Help on package cplex:

NAME
    cplex - The CPLEX Python API.

DESCRIPTION
    This package contains classes for accessing CPLEX from the Pytho
n
    programming language. The most important class defined by this p
ackage
    is the `Cplex` class, which provides methods for creating, modif
ying,
    querying, or solving an optimization problem, and for querying a
spects of
    a solution.

    The `exceptions` module defines the exception classes that are r
aised
    during abnormal operation by the CPLEX Python API.

    The `callbacks` module defines callback classes that can be used
 to alter
    the behavior of the algorithms used by CPLEX.

    The constant `infinity`, defined in the cplex package, should be
 used to
    set infinite upper and lower bounds.

    The classes `SparsePair` and `SparseTriple` are used as input an
d output
    classes for sparse vector and sparse matrix output, respectivel
y. See
    the documentation for individual methods for details about the u
sage
    of these classes.

PACKAGE CONTENTS
    _internal (package)
    aborter
    callbacks
    constant_class
    exceptions (package)
    model_info
    paramset

```
SUBMODULES
    _const
    _proc

CLASSES
    builtins.object
        Cplex
        Stats
        cplex._internal._matrices.SparsePair
        cplex._internal._matrices.SparseTriple
        cplex.aborter.Aborter
        cplex.paramset.ParameterSet

    class Aborter(builtins.object)
     |  Gracefully terminates the solve and tuning methods of CPLEX.
     |
     |  You can pass an instance of this class to one or more Cplex o
bjects.
     |
     |  Calling the method abort() will then terminate the solve or
tuning
     |  method of the Cplex object.
     |
     |  Methods defined here:
     |
     |  __del__(self)
     |
     |  __enter__(self)
     |      Enter the runtime context related to this object.
     |
     |      The with statement will bind this method's return value t
o the
     |      target specified in the as clause of the statement, if an
y.
     |
     |      Aborter objects return themselves.
     |
     |  __exit__(self, exc_type, exc_value, traceback)
     |      Exit the runtime context.
     |
     |      When we exit the with block, the end() method is called.
     |
     |  __init__(self)
```

```
    |      Constructor of the Aborter class.
    |
    |      The Aborter object is a context manager and can be used,
like so:
    |
    |      with Aborter() as aborter:
    |          # do stuff
    |
    |      When the with block is finished, the end() method will be
 called
    |      automatically.
    |
    |  abort(self)
    |      Aborts the solving and tuning methods.
    |
    |      Example usage:
    |
    |      >>> aborter = cplex.Aborter()
    |      >>> aborter.abort()
    |
    |  clear(self)
    |      Clears the invoking aborter.
    |
    |      Example usage:
    |
    |      >>> aborter = cplex.Aborter()
    |      >>> aborter.clear()
    |
    |  end(self)
    |      Ends the invoking aborter.
    |
    |      Example usage:
    |
    |      >>> aborter = cplex.Aborter()
    |      >>> aborter.end()
    |
    |  is_aborted(self)
    |      Returns True if the method to abort has been called.
    |
    |      Example usage:
    |
    |      >>> aborter = cplex.Aborter()
    |      >>> aborter.is_aborted()
    |      False
```

```
     |
     |  ----------------------------------------------------------------
-----------
     |  Data descriptors defined here:
     |
     |  __dict__
     |      dictionary for instance variables (if defined)
     |
     |  __weakref__
     |      list of weak references to the object (if defined)

   class Cplex(builtins.object)
     |  A class encapsulating a CPLEX Problem.
     |
     |  An instance of the Cplex class provides methods for creatin
g,
     |  modifying, and querying an optimization problem, solving it,
 and
     |  querying aspects of the solution.
     |
     |  Most of the methods are provided within subinterfaces: for e
xample,
     |  methods for adding, modifying, and querying data associated
with
     |  variables are within the `Cplex.variables` interface, and me
thods for
     |  querying the solution are within the `Cplex.solution` catego
ry.
     |
     |  Methods defined here:
     |
     |  __del__(self)
     |      non-public
     |
     |  __enter__(self)
     |      Enter the runtime context related to this object.
     |
     |      The "with" statement will bind this method's return value
 to the
     |      target specified in the as clause of the statement, if an
y.
     |
     |      Cplex objects return themselves.
     |
```

```
     |      Example usage:
     |
     |      >>> import cplex
     |      >>> with cplex.Cplex() as cpx:
     |      ...     # do stuff
     |      ...     pass
     |
     |  __exit__(self, exc_type, exc_value, traceback)
     |      Exit the runtime context.
     |
     |      When we exit the with-block, the `end()` method is called
     |      automatically.
     |
     |  __init__(self, *args)
     |      Constructor of the Cplex class.
     |
     |      The Cplex constructor accepts four types of argument list
s.
     |
     |      >>> cpx = cplex.Cplex()  # doctest: +SKIP
     |
     |      cpx is a new problem with no data
     |
     |      >>> cpx = cplex.Cplex("filename")  # doctest: +SKIP
     |
     |      cpx is a new problem containing the data in filename. If
filename
     |      does not exist, an exception is raised.
     |
     |      >>> cpx = cplex.Cplex("filename", "filetype")  # doctes
t: +SKIP
     |
     |      same as form 2, but cplex reads the file filename as a fi
le of
     |      type filetype, rather than inferring the file type from i
ts
     |      extension.
     |
     |      >>> cpx = cplex.Cplex(old_cpx)  # doctest: +SKIP
     |
     |      cpx contains the same problem data as old_cpx, but is a d
ifferent
     |      object and contains no solution data. Future changes to o
ne do
```

```
     |      not affect the other.
     |
     |      The Cplex object is a context manager and can be used, li
ke so:
     |
     |      >>> import cplex
     |      >>> with cplex.Cplex() as cpx:
     |      ...     # do stuff
     |      ...     pass
     |
     |      When the with-block is finished, the `end()` method will
be
     |      called automatically.
     |
     |  cleanup(self, epsilon)
     |      Deletes values from the problem data with absolute value
     |      smaller than epsilon.
     |
     |      See :cpxapi:`CPXcleanup` in the Callable Library Referen
ce Manual
     |      for more detail.
     |
     |      Example usage:
     |
     |      >>> import cplex
     |      >>> c = cplex.Cplex()
     |      >>> indices = c.variables.add(obj=[1.0, 1e-10, 1.0])
     |      >>> c.objective.get_linear()
     |      [1.0, 1e-10, 1.0]
     |      >>> c.cleanup(epsilon=1e-6)
     |      >>> c.objective.get_linear()
     |      [1.0, 0.0, 1.0]
     |
     |  copy_parameter_set(self, source)
     |      Returns a deep copy of a parameter set.
     |
     |      In a sense, this a convenience function; it is equivalent
 to
     |      querying what parameters are in the source parameter set,
     |      querying their values, and then adding those parameters t
o the
     |      target parameter set.
     |
     |      Note
```

```
    |       The source parameter set must have been created by this
 CPLEX
    |       problem object. Mixing parameter sets from different CP
LEX
    |       problem objects is not supported.
    |
    |     Note
    |       When this CPLEX problem object is destroyed, the parame
ter set
    |       object returned by this function will also be destoyed.
    |
    |     See `ParameterSet`.
    |
    |     See :cpxapi:`CPXparamsetcopy` in the Callable Library Re
ference
    |     Manual for more detail.
    |
    |     Example usage:
    |
    |     >>> import cplex
    |     >>> c = cplex.Cplex()
    |     >>> source = c.create_parameter_set()
    |     >>> source.add(c.parameters.advance,
    |     ...           c.parameters.advance.values.none)
    |     >>> len(source)
    |     1
    |     >>> target = c.copy_parameter_set(source)
    |     >>> len(target)
    |     1
    |
    |  copy_vmconfig(self, xmlstring)
    |     Read a virtual machine configuration from a string and in
stall
    |     it in this instance.
    |
    |     The content of the string passed to this function must co
nform to
    |     the VMC file specification. If the string can be successf
ully
    |     parsed, then the virtual machine configuration specified
 by it is
    |     installed in this instance. In case of error, a previousl
y
    |     installed virtual machine configuration is not touched.
```

```
     |
     |      See :distmipapi:`CPXcopyvmconfig` in the Callable Librar
y
     |      Reference Manual for more detail.
     |
     |  copylp(self, numcols, numrows, objsense=1, obj=None, rhs=No
ne, senses='', matbeg=None, matcnt=None, matind=None, matval=None,
 lb=None, ub=None, range_values=None, colnames=None, rownames=Non
e)
     |      Copies LP data into a CPLEX problem object.
     |
     |      The arguments define an objective function, constraint m
atrix,
     |      variable bounds, righthand side, constraint senses, rang
e values,
     |      names of constraints, and names of variables.
     |
     |      Note
     |        This method can give better performance when building a
 model,
     |        but it may not be as user friendly as using other metho
ds. To
     |        compare different techniques, see the lpex1.py example.
     |
     |      Note
     |        Calling this method destroys any existing data associat
ed with
     |        the problem object.
     |
     |      numcols : the number of columns in the constraint matrix,
 or
     |      equivalently, the number of variables in the problem obje
ct.
     |
     |      numrows : the number of rows in the constraint matrix, no
t
     |      including the objective function or bounds on the variabl
es.
     |
     |      objsense : sets the sense of the objective function. Must
 be
     |      either Cplex.objective.sense.minimize or
     |      Cplex.objective.sense.maximize.
     |
```

```
    |     obj : a list of floats of length at least `numcols` conta
ining
    |     the objective function coefficients. Required if `numcol
s` > 0.
    |
    |     rhs : a list of floats of length at least `numrows` conta
ining
    |     the righthand side value for each constraint in the const
raint
    |     matrix. Required if `numrows` > 0.
    |
    |     senses : A list of single-character strings or a string
    |     containing the sense of each constraint in the constraint
 matrix.
    |     Must be of length at least `numrows`. Each entry must be
one of
    |     'G', 'L', 'E', and 'R', indicating greater-than-or-equal
-to (>=),
    |     less-than-or-equal-to (<=), equality (=), and ranged con
straints,
    |     respectively. Required if `numrows` > 0.
    |
    |     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 arguments are required if
    |     `numcols` > 0 and `numrows` > 0.
    |
    |     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 entries:
    |     matval[matbeg[j]],.., matval[matbeg[j]+matcnt[j]-1]
    |
    |     The corresponding row indices are:
    |     matind[matbeg[j]],.., matind[matbeg[j]+matcnt[j]-1]
    |
    |     lb : a list of length at least `numcols` containing the l
ower
```

```
    |     bound on each of the variables. Required if `numcols` >
0.
    |
    |     ub : a list of length at least `numcols` containing the u
pper
    |     bound on each of the variables. Required if `numcols` >
0.
    |
    |     range_values : a list of floats, specifying the differenc
e
    |     between lefthand side and righthand side of each linear
    |     constraint. If range_values[i] > 0 (zero) then the constr
aint i
    |     is defined as rhs[i] <= rhs[i] + range_values[i]. If
    |     range_values[i] < 0 (zero) then constraint i is defined a
s
    |     rhs[i] + range_value[i] <= a*x <= rhs[i].
    |
    |     colnames : a list of strings of length at least `numcols`
    |     containing the names of the matrix columns or, equivalent
ly, the
    |     constraint names.
    |
    |     rownames : a list of strings of length at least `numrows`
    |     containing the names of the matrix rows or, equivalently,
 the
    |     constraint names.
    |
    |     See :cpxapi:`CPXcopylpwnames` in the Callable Library Re
ference
    |     Manual for more detail.
    |
    |     Example usage:
    |
    |     >>> import cplex
    |     >>> c = cplex.Cplex()
    |     >>> c.copylp(numcols=3,
    |     ...         numrows=2,
    |     ...         objsense=c.objective.sense.maximize,
    |     ...         obj=[1.0, 2.0, 3.0],
    |     ...         rhs=[20.0, 30.0],
    |     ...         senses="LL",
    |     ...         matbeg=[0, 2, 4],
    |     ...         matcnt=[2, 2, 2],
```

```
     |      ...         matind=[0, 1, 0, 1, 0, 1],
     |      ...         matval=[-1.0, 1.0, 1.0, -3.0, 1.0, 1.0],
     |      ...         lb=[0.0, 0.0, 0.0],
     |      ...         ub=[40.0, cplex.infinity, cplex.infinity],
     |      ...         range_values=[0.0, 0.0],
     |      ...         colnames=["x1", "x2", "x3"],
     |      ...         rownames=["c1", "c2"])
     |
     |  create_parameter_set(self)
     |      Returns a new CPLEX parameter set object that is associat
ed
     |      with this CPLEX problem object.
     |
     |      Note
     |        When this CPLEX problem object is destroyed, the parame
ter set
     |        object returned by this function will also be destoyed.
     |
     |      See `ParameterSet`.
     |
     |      Example usage:
     |
     |      >>> import cplex
     |      >>> c = cplex.Cplex()
     |      >>> ps = c.create_parameter_set()
     |      >>> ps.add(c.parameters.advance,
     |      ...       c.parameters.advance.values.none)
     |      >>> len(ps)
     |      1
     |
     |  del_vmconfig(self)
     |      Delete the virtual machine configuration in this instanc
e (if
     |      there is any).
     |
     |      See :distmipapi:`CPXdelvmconfig` in the Callable Library
     |      Reference Manual for more detail.
     |
     |      Example usage:
     |
     |      >>> import cplex
     |      >>> c = cplex.Cplex()
     |      >>> c.del_vmconfig()
     |
```

```
     |  end(self)
     |      Releases the Cplex object.
     |
     |      Frees all data structures associated with CPLEX. After a
call of
     |      the method end(), the invoking Cplex object and all objec
ts that
     |      have been created with it (such as variables and constrai
nts) can
     |      no longer be used. Attempts to use them subsequently rais
e a
     |      ValueError.
     |
     |      Note
     |        The Cplex object is a context manager. Thus, rather tha
n
     |        calling this method explicitly, the best practice shoul
d be to
     |        use a Cplex object in a "with" statement (see `__enter_
_` and
     |        `__exit__`).
     |
     |      Example usage:
     |
     |      >>> import cplex
     |      >>> cpx = cplex.Cplex()
     |      >>> cpx.end()
     |
     |  get_aborter(self)
     |      Returns the `Aborter` being used by the invoking object.
     |
     |      Example usage:
     |
     |      >>> import cplex
     |      >>> c = cplex.Cplex()
     |      >>> aborter = c.use_aborter(cplex.Aborter())
     |      >>> aborter = c.get_aborter()
     |
     |  get_dettime(self)
     |      Returns a deterministic time stamp in ticks.
     |
     |      To measure elapsed deterministic time in ticks between a
starting
```

```
     |      point and ending point of an operation, take the determin
istic
     |      time stamp at the starting point; take the deterministic
time
     |      stamp at the ending point; subtract the starting determin
istic
     |      time stamp from the ending deterministic time stamp.
     |
     |      The absolute value of the deterministic time stamp is not
     |      meaningful.
     |
     |      See :cpxapi:`CPXgetdettime` in the Callable Library Refe
rence
     |      Manual for more detail.
     |
     |      Example usage:
     |
     |      >>> import cplex
     |      >>> c = cplex.Cplex()
     |      >>> out = c.set_results_stream(None)
     |      >>> out = c.set_log_stream(None)
     |      >>> c.read("lpex.mps")
     |      >>> start = c.get_dettime()
     |      >>> c.solve()
     |      >>> solve_dettime = c.get_dettime() - start
     |
     |  get_num_cores(self)
     |      Returns the number of cores on this machine.
     |
     |      See :cpxapi:`CPXgetnumcores` in the Callable Library Ref
erence
     |      Manual for more detail.
     |
     |      Example usage:
     |
     |      >>> import cplex
     |      >>> c = cplex.Cplex()
     |      >>> num_cores = c.get_num_cores()
     |
     |  get_parameter_set(self)
     |      Returns a parameter set containing parameters that have b
een
     |      changed from their default values in the environment.
     |
```

```
     |      Example usage:
     |
     |      >>> import cplex
     |      >>> c = cplex.Cplex()
     |      >>> c.parameters.advance.set(c.parameters.advance.value
s.none)
     |      >>> ps = c.get_parameter_set()
     |      >>> val = ps.get(c.parameters.advance)
     |      >>> val == c.parameters.advance.values.none
     |      True
     |
     |  get_problem_name(self)
     |      Returns the problem name.
     |
     |      See :cpxapi:`CPXgetprobname` in the Callable Library Ref
erence
     |      Manual for more detail.
     |
     |      Example usage:
     |
     |      >>> import cplex
     |      >>> c = cplex.Cplex()
     |      >>> c.set_problem_name("prob1")
     |      >>> c.get_problem_name()
     |      'prob1'
     |
     |  get_problem_type(self)
     |      Returns the problem type.
     |
     |      See :cpxapi:`CPXgetprobtype` in the Callable Library Ref
erence
     |      Manual for more detail.
     |
     |      The return value is an attribute of `problem_type`.
     |
     |      Example usage:
     |
     |      >>> import cplex
     |      >>> c = cplex.Cplex()
     |      >>> out = c.set_results_stream(None)
     |      >>> out = c.set_log_stream(None)
     |      >>> c.read("lpex.mps")
     |      >>> c.get_problem_type()
     |      0
```

```
    |      >>> c.problem_type[c.get_problem_type()]
    |      'LP'
    |
    |  get_stats(self)
    |      Returns a `Stats` object containing problem statistics.
    |
    |      Note
    |        Printing the `Stats` object will give a nice summary of
 the
    |        problem statistics in human readable form (e.g. as with
 the
    |        "display problem statistics" command in the CPLEX inter
active).
    |
    |      Example usage:
    |
    |      >>> import cplex
    |      >>> c = cplex.Cplex()
    |      >>> out = c.set_results_stream(None)
    |      >>> out = c.set_log_stream(None)
    |      >>> c.read("lpex.mps")
    |      >>> stats = c.get_stats()
    |      >>> stats.num_variables
    |      32
    |      >>> stats.num_linear_constraints
    |      27
    |
    |  get_time(self)
    |      Returns a time stamp in seconds.
    |
    |      To measure time spent between a starting point and ending
 point
    |      of an operation, take the result of this method at the st
arting
    |      point; take the result of this method at the end point; s
ubtract
    |      the starting time stamp from the ending time stamp; the
    |      subtraction yields elapsed time in seconds.
    |
    |      The interpretation of this value as wall clock time or CP
U time
    |      is controlled by the parameter clocktype.
    |
    |      The absolute value of the time stamp is not meaningful.
```

```
     |
     |      See :cpxapi:`CPXgettime` in the Callable Library Referen
ce Manual
     |      for more detail.
     |
     |      Example usage:
     |
     |      >>> import cplex
     |      >>> c = cplex.Cplex()
     |      >>> out = c.set_results_stream(None)
     |      >>> out = c.set_log_stream(None)
     |      >>> c.read("lpex.mps")
     |      >>> start = c.get_time()
     |      >>> c.solve()
     |      >>> solve_time = c.get_time() - start
     |
     |  get_version(self)
     |      Returns a string specifying the version of CPLEX.
     |
     |      See :cpxapi:`CPXversion` in the Callable Library Referen
ce Manual
     |      for more detail.
     |
     |      Example usage:
     |
     |      >>> import cplex
     |      >>> c = cplex.Cplex()
     |      >>> version = c.get_version()
     |
     |  get_versionnumber(self)
     |      Returns an integer specifying the version of CPLEX.
     |
     |      The version of CPLEX is in the format vvrrmmff, where vv
is the
     |      version, rr is the release, mm is the modification, and f
f is the
     |      fixpack number. For example, for CPLEX version 12.5.0.1 t
he
     |      returned value is 12050001.
     |
     |      See :cpxapi:`CPXversionnumber` in the Callable Library R
eference
     |      Manual for more detail.
     |
```

```
     |      Example usage:
     |
     |      >>> import cplex
     |      >>> c = cplex.Cplex()
     |      >>> versionnumber = c.get_versionnumber()
     |
     |  has_vmconfig(self)
     |      Test whether this instance has a virtual machine configur
ation
     |      installed.
     |
     |      See `copy_vmconfig`, `read_copy_vmconfig`, and `del_vmco
nfig`.
     |
     |      Example usage:
     |
     |      >>> import cplex
     |      >>> c = cplex.Cplex()
     |      >>> c.has_vmconfig()
     |      False
     |
     |  populate_solution_pool(self)
     |      Generates a variety of solutions to a discrete problem (M
IP, MIQP, MIQCP).
     |
     |      The algorithm that populates the solution pool works in t
wo
     |      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 u
sing
     |      the information computed and stored in the first phase an
d by
     |      continuing to explore the tree.
     |
     |      For more information, see the function :mipapi:`CPXpopul
ate` in the
```

```
    |      Callable Library Reference Manual and the topic solution
 pool
    |      in the CPLEX User's Manual.
    |
    |  read(self, filename, filetype='')
    |      Reads a problem from file.
    |
    |      The first argument is a string specifying the filename fr
om which
    |      the problem will be read.
    |
    |      If the method is called with two arguments, the second ar
gument
    |      is a string specifying the file type. If this argument is
    |      omitted, filetype is taken to be the extension of the fil
ename.
    |
    |      See :cpxapi:`CPXreadcopyprob` in the Callable Library Re
ference
    |      Manual for more detail.
    |
    |      Example usage:
    |
    |      >>> import cplex
    |      >>> c = cplex.Cplex()
    |      >>> out = c.set_results_stream(None)
    |      >>> out = c.set_log_stream(None)
    |      >>> c.read("lpex.mps")
    |
    |  read_annotations(self, filename)
    |      Reads annotations from a file.
    |
    |      See :cpxapi:`CPXreadcopyannotations` in the Callable Lib
rary
    |      Reference Manual for more detail.
    |
    |      Example usage:
    |
    |      >>> import cplex
    |      >>> c = cplex.Cplex()
    |      >>> idx = c.long_annotations.add('ann1', 0)
    |      >>> objtype = c.long_annotations.object_type.variable
    |      >>> indices = c.variables.add(names=['v1', 'v2', 'v3'])
    |      >>> c.long_annotations.set_values(idx, objtype,
```

```
    |      ...                            [(i, 1) for i in indices])
    |      >>> idx = c.double_annotations.add('ann1', 0)
    |      >>> objtype = c.double_annotations.object_type.variable
    |      >>> indices = c.variables.add(names=['v1', 'v2', 'v3'])
    |      >>> c.double_annotations.set_values(idx, objtype,
    |      ...                              [(i, 1) for i in indices])
    |      >>> c.write_annotations('example.ann')
    |      >>> c.long_annotations.delete()
    |      >>> c.double_annotations.delete()
    |      >>> c.long_annotations.get_num()
    |      0
    |      >>> c.double_annotations.get_num()
    |      0
    |      >>> c.read_annotations('example.ann')
    |      >>> c.long_annotations.get_num()
    |      1
    |      >>> c.double_annotations.get_num()
    |      1
    |
    |  read_copy_vmconfig(self, filename)
    |      Read a virtual machine configuration from a file and inst
all
    |      it in this instance.
    |
    |      The filename argument to this function must specify a fil
e that
    |      conforms to the VMC file format. If the file can be succe
ssfully
    |      parsed, then the virtual machine configuration specified
 by it is
    |      installed in this instance. In case of error, a previousl
y
    |      installed virtual machine configuration is not touched.
    |
    |      See :distmipapi:`CPXreadcopyvmconfig` in the Callable Li
brary
    |      Reference Manual for more detail.
    |
    |  register_callback(self, callback_class)
    |      Registers a callback class for use during optimization.
    |
    |      callback_class must be a proper subclass of one of the ca
llback
```

```
    |      classes defined in the module `callbacks`. To implement c
ustom
    |      logic, override the __call__ method with a method that ha
s
    |      signature __call__(self) -> None. If callback_class is a
 subclass
    |      of more than one callback class, it will only be called w
hen its
    |      first superclass is called. register_callback returns th
e
    |      instance of callback_class registered for use. Any previ
ously
    |      registered callback of the same class will no longer be
    |      registered.
    |
    |      Returns an instance of callback_class.
    |
    |      Example usage:
    |
    |      >>> import cplex
    |      >>> c = cplex.Cplex()
    |      >>> class MyMIPInfoCallback(cplex.callbacks.MIPInfoCallb
ack):
    |      ...    pass
    |      >>> cb = c.register_callback(MyMIPInfoCallback)
    |
    |  remove_aborter(self)
    |      Removes the `Aborter` being used by the invoking object.
    |
    |      Returns the aborter that was removed or None.
    |
    |      Example usage:
    |
    |      >>> import cplex
    |      >>> c = cplex.Cplex()
    |      >>> aborter = c.use_aborter(cplex.Aborter())
    |      >>> aborter = c.remove_aborter()
    |
    |  runseeds(self, cnt=30)
    |      Evaluates the variability of the problem.
    |
    |      Solves the same problem instance multiple times using dif
ferent
```

```
     |      random seeds allowing the user to evaluate the variabilit
y of the
     |      problem class the instance belongs to.
     |
     |      The optional cnt argument specifies the number of times
     |      optimization should be performed (the default is 30).
     |
     |      A problem must be an MILP, MIQP, or MIQCP and must exist
in
     |      memory.
     |
     |  set_callback(self, functor=None, contextmask=0)
     |      Set callback function to use during optimization.
     |
     |      Sets the callback that CPLEX invokes during optimizatio
n. If
     |      functor is None then contextmask will be treated as 0 and
 the
     |      callback is effectively cleared from CPLEX.
     |
     |      In all other cases functor must be a reference to an obje
ct that
     |      has a callable member called 'invoke' (if that does not e
xist, or
     |      is not a callable, an exception will occur the first time
 CPLEX
     |      attempts to invoke the callback). Whenever CPLEX needs to
 invoke
     |      the callback it calls this member with exactly one argume
nt: an
     |      instance of `cplex.callbacks.Context`.
     |
     |      Note that in the 'invoke' function you must not invoke an
y
     |      functions of the Cplex instance that is performing the cu
rrent
     |      solve. All functions that can be invoked from a callback
are
     |      members of the `cplex.callbacks.Context` class.
     |
     |      contextmask must be the bitwise OR of values from
     |      `cplex.callbacks.Context.id` and specifies in which cont
exts
```

```
    |      CPLEX shall invoke the callback: the callback is invoked
in all
    |      contexts for which the corresponding bit is set in contex
tmask.
    |
    |      Note about cplex.callbacks.Context.id.thread_down: This
is
    |      considered a "destructor" function and should not raise a
ny
    |      exception. Any exception raised from the callback in this
 context
    |      will just be ignored.
    |
    |      See `cplex.callbacks.Context`.
    |
    |      See :cpxapi:`CPXcallbacksetfunc` in the Callable Library
    |      Reference Manual for more detail.
    |
    |      Example usage:
    |
    |      >>> import cplex
    |      >>> c = cplex.Cplex()
    |      >>> class GenericCB():
    |      ...     def invoke(self, context):
    |      ...         pass  # Do something here.
    |      >>> cb = GenericCB()
    |      >>> c.set_callback(cb)  # Register callback.
    |      >>> c.set_callback(None)  # Clear callback.
    |
    |  set_error_stream(self, error_file, fn=None)
    |      Specifies where errors will be printed.
    |
    |      The first argument must be a file-like object (i.e., an o
bject
    |      with a write method and a flush method). Use None as the
first
    |      argument to suppress output.
    |
    |      The second optional argument is a function that takes a s
tring as
    |      input and returns a string. If specified, strings sent to
 this
    |      stream will be processed by this function before being wr
itten.
```

```
    |
    |      Returns the stream to which errors will be written. To wr
ite to
    |      this stream, use this object's write() method.
    |
    |      Example usage:
    |
    |      >>> import cplex
    |      >>> with cplex.Cplex() as c, open("output.txt", "w") as
f:
    |      ...     output = c.set_error_stream(f)
    |      ...     output.write("this is an example")
    |
    |  set_log_stream(self, log_file, fn=None)
    |      Specifies where the log will be printed.
    |
    |      The first argument must be a file-like object (i.e., an o
bject
    |      with a write method and a flush method). Use None as the
first
    |      argument to suppress output.
    |
    |      The second optional argument is a function that takes a s
tring as
    |      input and returns a string. If specified, strings sent to
 this
    |      stream will be processed by this function before being wr
itten.
    |
    |      Returns the stream to which the log will be written. To w
rite to
    |      this stream, use this object's write() method.
    |
    |      >>> import cplex
    |      >>> with cplex.Cplex() as c, open("output.txt", "w") as
f:
    |      ...     output = c.set_log_stream(f)
    |      ...     output.write("this is an example")
    |
    |  set_modeling_assistance_callback(self, functor=None)
    |      Set callback function to use for modeling assistance warn
ings.
    |
    |      Sets the callback that CPLEX invokes before and after
```

```
    |     optimization (once for every modeling issue detected). I
f functor
    |     is None then the callback is effectively cleared from CPL
EX. The
    |     callback function will only be invoked if the CPLEX param
eter
    |     Cplex.parameters.read.datacheck is set to
    |     Cplex.parameters.read.datacheck.values.assist (2). In ad
dition,
    |     the parameter Cplex.parameters.read.warninglimit control
s the
    |     number of times each type of modeling assistance warning
will be
    |     reported (the rest will be ignored). See CPX_PARAM_DATAC
HECK and
    |     CPX_PARAM_WARNLIM in the Parameters of CPLEX Reference M
anual.
    |
    |     In all other cases functor must be a reference to an obje
ct that
    |     has a callable attribute named 'invoke' (if that does not
 exist,
    |     or is not a callable, an exception will occur the first t
ime CPLEX
    |     attempts to invoke the callback). Whenever CPLEX needs to
 invoke
    |     the callback it calls this member with two argument: the
modeling
    |     issue ID and the associated warning message.
    |
    |     See `model_info`.
    |
    |     See :cpxapi:`CPXmodelasstcallbacksetfunc` in the Callabl
e Library
    |     Reference Manual for more detail.
    |
    |     Example usage:
    |
    |     >>> import cplex
    |     >>> c = cplex.Cplex()
    |     >>> c.parameters.read.datacheck.set(
    |     ...     c.parameters.read.datacheck.values.assist)
    |     >>> class ModelAsstCB():
    |     ...     def invoke(self, issueid, message):
```

```
    |      ...        pass  # Do something here.
    |      >>> cb = ModelAsstCB()
    |      >>> c.set_modeling_assistance_callback(cb)  # Register c
allback.
    |      >>> c.set_modeling_assistance_callback(None)  # Clear ca
llback.
    |
    |  set_parameter_set(self, source)
    |      Applies the parameter values in the paramset to the
    |      environment.
    |
    |      Note
    |        The source parameter set must have been created by this
 CPLEX
    |        problem object. Mixing parameter sets from different CP
LEX
    |        problem objects is not supported.
    |
    |      See :cpxapi:`CPXparamsetapply` in the Callable Library R
eference
    |      Manual for more detail.
    |
    |      Example usage:
    |
    |      >>> import cplex
    |      >>> c = cplex.Cplex()
    |      >>> ps = c.create_parameter_set()
    |      >>> ps.add(c.parameters.advance,
    |      ...       c.parameters.advance.values.none)
    |      >>> c.set_parameter_set(ps)
    |      >>> value = c.parameters.advance.get()
    |      >>> value == c.parameters.advance.values.none
    |      True
    |
    |  set_problem_name(self, name)
    |      Sets the problem name.
    |
    |      See :cpxapi:`CPXchgprobname` in the Callable Library Ref
erence
    |      Manual for more detail.
    |
    |      Example usage:
    |
    |      >>> import cplex
```

```
     |      >>> c = cplex.Cplex()
     |      >>> c.set_problem_name("prob1")
     |      >>> c.get_problem_name()
     |      'prob1'
     |
     |  set_problem_type(self, type, soln=None)
     |      Changes the problem type.
     |
     |      If only one argument is given, that argument specifies th
e new
     |      problem type (see `problem_type`). It must be one of the
     |      following:
     |
     |      * Cplex.problem_type.LP
     |      * Cplex.problem_type.MILP
     |      * Cplex.problem_type.fixed_MILP
     |      * Cplex.problem_type.QP
     |      * Cplex.problem_type.MIQP
     |      * Cplex.problem_type.fixed_MIQP
     |      * Cplex.problem_type.QCP
     |      * Cplex.problem_type.MIQCP
     |
     |      If an optional second argument is given, it is taken to b
e an
     |      identifier of a member of the solution pool. In this cas
e, the
     |      first argument must be one of the following:
     |
     |      * Cplex.problem_type.fixed_MILP
     |      * Cplex.problem_type.fixed_MIQP
     |
     |      See :cpxapi:`CPXchgprobtype` and :cpxapi:`CPXchgprobtype
solnpool`
     |      in the Callable Library Reference Manual for more detail.
     |
     |      Example usage:
     |
     |      >>> import cplex
     |      >>> c = cplex.Cplex()
     |      >>> c.set_problem_type(c.problem_type.LP)
     |
     |  set_results_stream(self, results_file, fn=None)
     |      Specifies where results will be printed.
     |
```

```
    |      The first argument must be a file-like object (i.e., an o
bject
    |      with a write method and a flush method). Use None as the
first
    |      argument to suppress output.
    |
    |      The second optional argument is a function that takes a s
tring as
    |      input and returns a string. If specified, strings sent to
 this
    |      stream will be processed by this function before being wr
itten.
    |
    |      Returns the stream to which results will be written. To w
rite to
    |      this stream, use this object's write() method.
    |
    |      Example usage:
    |
    |      >>> import cplex
    |      >>> with cplex.Cplex() as c, open("output.txt", "w") as
f:
    |      ...     output = c.set_results_stream(f)
    |      ...     output.write("this is an example")
    |
    |  set_warning_stream(self, warning_file, fn=None)
    |      Specifies where warnings will be printed.
    |
    |      The first argument must be a file-like object (i.e., an o
bject
    |      with a write method and a flush method). Use None as the
first
    |      argument to suppress output.
    |
    |      The second optional argument is a function that takes a s
tring as
    |      input and returns a string. If specified, strings sent to
 this
    |      stream will be processed by this function before being wr
itten.
    |
    |      Returns the stream to which warnings will be written. To
write to
    |      this stream, use this object's write() method.
```

```
     |
     |      Example usage:
     |
     |      >>> import cplex
     |      >>> with cplex.Cplex() as c, open("output.txt", "w") as
f:
     |      ...     output = c.set_warning_stream(f)
     |      ...     output.write("this is an example")
     |
     |  solve(self, paramsets=None)
     |      Solves the problem.
     |
     |      The optional paramsets argument can only be specified whe
n
     |      multiple objectives are present (otherwise, a ValueError
 is
     |      raised). paramsets must be a sequence containing `Parame
terSet`
     |      objects (see `Cplex.create_parameter_set`) or None. See
     |      :cpxapi:`CPXmultiobjopt` in the Callable Library Referen
ce Manual
     |      for more detail.
     |
     |      Note
     |        The solve method returning normally (i.e., without rais
ing an
     |        exception) does not necessarily mean that an optimal or
     |        feasible solution has been found. Use
     |        `SolutionInterface.get_status()` to query the status o
f the
     |        current solution.
     |
     |      Example usage:
     |
     |      >>> import cplex
     |      >>> c = cplex.Cplex()
     |      >>> out = c.set_results_stream(None)
     |      >>> out = c.set_log_stream(None)
     |      >>> c.read("lpex.mps")
     |      >>> c.solve()
     |      >>> status = c.solution.get_status()
     |
     |  unregister_callback(self, callback_class)
     |      Stops a callback class from being used.
```

```
    |
    |     callback_class must be one of the callback classes define
d in the
    |     module `callbacks` or a subclass of one of them. This met
hod
    |     unregisters any previously registered callback of the sa
me class.
    |     If callback_class is a subclass of more than one callback
 class,
    |     this method will unregister only the callback of the same
 type as
    |     its first superclass.
    |
    |     Returns the instance of callback_class just unregistere
d.
    |
    |     Example usage:
    |
    |     >>> import cplex
    |     >>> c = cplex.Cplex()
    |     >>> class MyMIPInfoCallback(cplex.callbacks.MIPInfoCallb
ack):
    |     ...    pass
    |     >>> cb = c.register_callback(MyMIPInfoCallback)
    |     >>> cb = c.unregister_callback(MyMIPInfoCallback)
    |
    |  use_aborter(self, aborter)
    |     Use an `Aborter` to control termination of solve methods.
    |
    |     Instructs the invoking object to use the aborter to contr
ol
    |     termination of its solving and tuning methods.
    |
    |     If another aborter is already being used by the invoking
object,
    |     then this method overrides the previously used aborter.
    |
    |     Returns the aborter installed in the invoking object or N
one.
    |
    |     Example usage:
    |
    |     >>> import cplex
    |     >>> c = cplex.Cplex()
```

```
    |      >>> aborter = c.use_aborter(cplex.Aborter())
    |
    |  write(self, filename, filetype='')
    |      Writes a problem to a file.
    |
    |      The first argument is a string specifying the filename to
 which
    |      the problem will be written.
    |
    |      If the filename ends with .bz2 (for BZip2) or .gz (for GN
U Zip),
    |      a compressed file is written.
    |
    |      If the method is called with two arguments, the second ar
gument
    |      is a string specifying the file type. If this argument is
    |      omitted, filetype is taken to be the extension of the fil
ename.
    |
    |      If filetype is any of "sav", "mps", "lp", the problem is
written
    |      in the corresponding format. If filetype is either "rew"
or "rlp"
    |      the problem is written with generic names in mps or lp fo
rmat,
    |      respectively. If filetype is "alp" the problem is written
 with
    |      generic names in lp format, where the variable names are
    |      annotated to indicate the type and bounds of each variabl
e.
    |
    |      If filetype is "dua", the dual problem is written to a fi
le. If
    |      filetype is "emb", an embedded network problem is written
 to a
    |      file. If filetype is "ppe", the perturbed problem is writ
ten to a
    |      file. If filetype is "dpe", the perturbed dual problem is
 written
    |      to a file.
    |
    |      For documentation of the file types, see the CPLEX File F
ormat
    |      Reference Manual.
```

```
    |
    |      See :cpxapi:`CPXwriteprob`, :cpxapi:`CPXdualwrite`,
    |      :cpxapi:`CPXembwrite`, :cpxapi:`CPXdperwrite`, and
    |      :cpxapi:`CPXpperwrite` in the Callable Library Reference
 Manual
    |      for more detail.
    |
    |      Example usage:
    |
    |      >>> import cplex
    |      >>> c = cplex.Cplex()
    |      >>> indices = c.variables.add(names=['x1', 'x2', 'x3'])
    |      >>> c.write("example.lp")
    |
    |  write_annotations(self, filename)
    |      Writes the annotations to a file.
    |
    |      See :cpxapi:`CPXwriteannotations` in the Callable Librar
y
    |      Reference Manual for more detail.
    |
    |      Example usage:
    |
    |      >>> import cplex
    |      >>> c = cplex.Cplex()
    |      >>> idx = c.long_annotations.add('ann1', 0)
    |      >>> objtype = c.long_annotations.object_type.variable
    |      >>> indices = c.variables.add(names=['v1', 'v2', 'v3'])
    |      >>> c.long_annotations.set_values(idx, objtype,
    |      ...                             [(i, 1) for i in indices])
    |      >>> c.write_annotations('example.ann')
    |
    |  write_as_string(self, filetype='LP', comptype='')
    |      Writes a problem as a string in the given file format.
    |
    |      For an explanation of the filetype and comptype argument
s, see
    |      `Cplex.write_to_stream`.
    |
    |      Note
    |        When SAV format is specified for filetype or a compress
ed file
    |        format is specified for comptype, the return value will
 be a
```

```
     |       byte string.
     |
     |     Example usage:
     |
     |     >>> import cplex
     |     >>> c = cplex.Cplex()
     |     >>> indices = c.variables.add(names=['x1', 'x2', 'x3'])
     |     >>> lp_str = c.write_as_string("lp")
     |     >>> len(lp_str) > 0
     |     True
     |
     |  write_benders_annotation(self, filename)
     |     Writes the annotation of the auto-generated decompositio
n.
     |
     |     Writes the annotation of the decompostion CPLEX automati
cally
     |     generates for the model of the CPLEX problem object to th
e
     |     specified file.
     |
     |     See :cpxapi:`CPXwritebendersannotation` in the Callable
Library
     |     Reference Manual for more detail.
     |
     |     Example usage:
     |
     |     >>> import cplex
     |     >>> c = cplex.Cplex()
     |     >>> out = c.set_results_stream(None)
     |     >>> out = c.set_log_stream(None)
     |     >>> c.read('UFL_25_35_1.mps')
     |     >>> c.write_benders_annotation('UFL_25_35_1.ann')
     |
     |  write_to_stream(self, stream, filetype='LP', comptype='')
     |     Writes a problem to a file-like object in the given file
format.
     |
     |     The filetype argument can be any of "sav" (a binary forma
t), "lp"
     |     (the default), "mps", "rew", "rlp", or "alp" (see `Cplex.
write`
     |     for an explanation of these).
     |
```

```
     |      If comptype is "bz2" (for BZip2) or "gz" (for GNU Zip), a
     |      compressed file is written.
     |
     |      See :cpxapi:`CPXwriteprob` in the Callable Library Refer
ence
     |      Manual for more detail.
     |
     |      Example usage:
     |
     |      >>> import cplex
     |      >>> c = cplex.Cplex()
     |      >>> indices = c.variables.add(names=['x1', 'x2', 'x3'])
     |      >>> class NoOpStream():
     |      ...     def __init__(self):
     |      ...         self.was_called = False
     |      ...     def write(self, bytes):
     |      ...         self.was_called = True
     |      ...         pass
     |      ...     def flush(self):
     |      ...         pass
     |      >>> stream = NoOpStream()
     |      >>> c.write_to_stream(stream)
     |      >>> stream.was_called
     |      True
     |
     |  ----------------------------------------------------------
-----------
     |  Data descriptors defined here:
     |
     |  __dict__
     |      dictionary for instance variables (if defined)
     |
     |  __weakref__
     |      list of weak references to the object (if defined)
     |
     |  ----------------------------------------------------------
-----------
     |  Data and other attributes defined here:
     |
     |  problem_type = <cplex._internal.ProblemType object>

    class ParameterSet(builtins.object)
     |  A parameter set object for use with multi-objective optimiza
tion.
```

```
    |
    |  A parameter set consists of key-value pairs where the key is
 a CPLEX
    |  parameter ID (e.g., CPX_PARAM_ADVIND) and the value is the a
ssociated
    |  parameter value.
    |
    |  When adding, getting, or deleting items from a parameter set
 the
    |  param argument can be either a Parameter object (e.g,
    |  Cplex.parameters.advance) or an integer ID (e.g., CPX_PARAM_
ADVIND
    |  (1001)).
    |
    |  For more details see the section on multi-objective optimiza
tion in
    |  the CPLEX User's Manual.
    |
    |  See `Cplex.create_parameter_set` and `Cplex.copy_parameter_
set`.
    |
    |  Example usage:
    |
    |  >>> import cplex
    |  >>> c = cplex.Cplex()
    |  >>> ps = c.create_parameter_set()
    |  >>> ps.add(c.parameters.advance, c.parameters.advance.value
s.none)
    |  >>> len(ps)
    |  1
    |
    |  Methods defined here:
    |
    |  __del__(self)
    |      Destructor of the ParameterSet class.
    |
    |      When a ParameterSet object is destoyed, the end() method
is
    |      called.
    |
    |      Example usage:
    |
    |      >>> import cplex
    |      >>> c = cplex.Cplex()
```

```
     |      >>> ps = c.create_parameter_set()
     |      >>> del ps
     |
     |  __enter__(self)
     |      Enter the runtime context related to this object.
     |
     |      The with statement will bind this method's return value t
o the
     |      target specified in the as clause of the statement, if an
y.
     |
     |      ParameterSet objects return themselves.
     |
     |      Example usage:
     |
     |      >>> import cplex
     |      >>> c = cplex.Cplex()
     |      >>> with c.create_parameter_set():
     |      ...     pass  # do something here
     |
     |  __exit__(self, exc_type, exc_value, traceback)
     |      Exit the runtime context.
     |
     |      When we exit the with block, the end() method is called.
     |
     |  __init__(self, env)
     |      Constructor of the ParameterSet class.
     |
     |      This class is not meant to be instantiated directly nor u
sed
     |      externally.
     |
     |  __len__(self)
     |      Return the number of items in the parameter set.
     |
     |      Example usage:
     |
     |      >>> import cplex
     |      >>> c = cplex.Cplex()
     |      >>> ps = c.create_parameter_set()
     |      >>> len(ps)
     |      0
     |
     |  add(self, param, value)
```

```
|      Add a parameter ID and value to a parameter set.
|
|      Example usage:
|
|      >>> import cplex
|      >>> c = cplex.Cplex()
|      >>> ps = c.create_parameter_set()
|      >>> ps.add(c.parameters.advance,
|      ...       c.parameters.advance.values.none)
|
|  clear(self)
|      Clears all items from the parameter set.
|
|      Example Usage:
|
|      >>> import cplex
|      >>> c = cplex.Cplex()
|      >>> ps = c.create_parameter_set()
|      >>> ps.add(c.parameters.advance,
|      ...       c.parameters.advance.values.none)
|      >>> ps.clear()
|      >>> len(ps)
|      0
|
|  delete(self, param)
|      Deletes a parameter from a parameter set.
|
|      Example usage:
|
|      >>> import cplex
|      >>> c = cplex.Cplex()
|      >>> ps = c.create_parameter_set()
|      >>> ps.add(c.parameters.advance,
|      ...       c.parameters.advance.values.none)
|      >>> len(ps)
|      1
|      >>> ps.delete(c.parameters.advance)
|      >>> len(ps)
|      0
|
|  end(self)
|      Releases the ParameterSet object.
|
```

```
    |      Frees all data structures associated with a ParameterSe
t. After
    |      a call of the method end(), the invoking object can no lo
nger be
    |      used. Attempts to use them subsequently raise a ValueErro
r.
    |
    |      Example usage:
    |
    |      >>> import cplex
    |      >>> c = cplex.Cplex()
    |      >>> ps = c.create_parameter_set()
    |      >>> ps.end()
    |
    |  get(self, param)
    |      Gets a parameter value.
    |
    |      Example usage:
    |
    |      >>> import cplex
    |      >>> c = cplex.Cplex()
    |      >>> ps = c.create_parameter_set()
    |      >>> ps.add(c.parameters.advance,
    |      ...       c.parameters.advance.values.none)
    |      >>> val = ps.get(c.parameters.advance)
    |      >>> val == c.parameters.advance.values.none
    |      True
    |
    |  get_ids(self)
    |      Gets the parameter IDs contained in a parameter set.
    |
    |      Returns an iterator containing the parameter IDs in a par
ameter
    |      set.
    |
    |      Example usage:
    |
    |      >>> import cplex
    |      >>> c = cplex.Cplex()
    |      >>> ps = c.create_parameter_set()
    |      >>> ps.add(c.parameters.advance,
    |      ...       c.parameters.advance.values.none)
    |      >>> list(ps.get_ids())
    |      [1001]
```

```
     |
     |  read(self, filename)
     |      Reads parameter names and settings from the file specifie
d by
     |      filename and copies them into the parameter set.
     |
     |      Note that the content of the parameter set is not cleared
 out
     |      before the parameters in the file are copied into the par
ameter
     |      set. The parameters are read from the file one by one and
 are
     |      added to the parameter set, or, if the parameter was alre
ady
     |      present in the set, then its value is updated.
     |
     |      This routine reads and copies files in the PRM format, as
 created
     |      by Cplex.parameters.write. The PRM format is documented
in the
     |      CPLEX File Formats Reference Manual.
     |
     |      Example usage:
     |
     |      >>> import cplex
     |      >>> c = cplex.Cplex()
     |      >>> c.parameters.advance.set(c.parameters.advance.value
s.none)
     |      >>> c.parameters.write_file('example.prm')
     |      >>> ps = c.create_parameter_set()
     |      >>> ps.read('example.prm')
     |      >>> value = ps.get(c.parameters.advance)
     |      >>> value == c.parameters.advance.values.none
     |      True
     |
     |  write(self, filename)
     |      Writes a parameter file that contains the parameters in t
he
     |      parameter set.
     |
     |      This routine writes a file in a format suitable for readi
ng by
     |      ParameterSet.read or by Cplex.parameters.read.
     |
```

```
    |      The file is written in the PRM format which is documented
 in the
    |      CPLEX File Formats Reference Manual.
    |
    |      Example usage:
    |
    |      >>> import cplex
    |      >>> c = cplex.Cplex()
    |      >>> ps = c.create_parameter_set()
    |      >>> ps.add(c.parameters.advance,
    |      ...       c.parameters.advance.values.none)
    |      >>> ps.write('example.prm')
    |      >>> c.parameters.read_file('example.prm')
    |      >>> value = c.parameters.advance.get()
    |      >>> value == c.parameters.advance.values.none
    |      True
    |
    |  ----------------------------------------------------------
-----------
    |  Data descriptors defined here:
    |
    |  __dict__
    |      dictionary for instance variables (if defined)
    |
    |  __weakref__
    |      list of weak references to the object (if defined)

   class SparsePair(builtins.object)
    |  A class for storing sparse vector data.
    |
    |  An instance of this class has two attributes, ind and val.  i
nd
    |  specifies the indices and val specifies the values.  ind and
 val
    |  must be sequences of the same length.  In general, ind may co
ntain
    |  any identifier; for example, when a SparsePair object is pas
sed to
    |  Cplex.linear_constraints.add, its ind attribute may be a lis
t
    |  containing both variable names and variable indices.
    |
    |  Methods defined here:
    |
```

```
     |  __init__(self, ind=None, val=None)
     |      Constructor for SparsePair.
     |
     |      Takes two arguments, ind and val; ind specifies the indic
es that
     |      the SparsePair refers to, and val specifies the float val
ues
     |      associated with those indices; ind and val must have the
same
     |      length.  If ind or val is omitted, they will default to a
n empty
     |      list.
     |
     |      >>> spair = SparsePair(ind=[0], val=[1.0])
     |
     |  __repr__(self)
     |      Representation method of SparsePair.
     |
     |      Example usage:
     |
     |      >>> SparsePair(ind=[0], val=[1.0])
     |      SparsePair(ind = [0], val = [1.0])
     |
     |  isvalid(self)
     |      Tests that ind and val have the same length.
     |
     |      Example usage:
     |
     |      >>> spair = SparsePair(ind=[0, 1, 2], val=[1.0, 1.0, 1.
0])
     |      >>> spair.isvalid()
     |      True
     |
     |  unpack(self)
     |      Extracts the indices and values sequences as a tuple.
     |
     |      Returns ind and val as given in __init__.
     |
     |      >>> spair = SparsePair(ind=[0, 1, 2], val=[1.0, 1.0, 1.
0])
     |      >>> ind, val = spair.unpack()
     |
     |  ----------------------------------------------------------
-----------
```

```
     |  Data descriptors defined here:
     |
     |  __dict__
     |      dictionary for instance variables (if defined)
     |
     |  __weakref__
     |      list of weak references to the object (if defined)

    class SparseTriple(builtins.object)
     |  A class for storing sparse matrix data.
     |
     |  An instance of this class has three attributes, ind1, ind2,
and val.
     |  ind1 and ind2 specify the indices and val specifies the valu
es.
     |  ind1, ind2, and val must be sequences of the same length.  In
     |  general, ind1 and ind2 may contain any identifier; for examp
le, when
     |  a SparseTriple object is passed to Cplex.quadratic_constrain
ts.add,
     |  its ind1 attribute may be a list containing both variable na
mes and
     |  variable indices.
     |
     |  Methods defined here:
     |
     |  __init__(self, ind1=None, ind2=None, val=None)
     |      Constructor for SparseTriple.
     |
     |      Takes three arguments, ind1, ind2 and val, specifying the
     |      indices that the SparseTriple refers to and the float val
ues
     |      associated with those indices, respectively.  ind1, ind
2, and
     |      val must all have the same length.  If ind1, ind2, or val
 is
     |      omitted, they will default to an empty list.
     |
     |      >>> striple = SparseTriple(ind1=[0], ind2=[0], val=[1.
0])
     |
     |  __repr__(self)
     |      Representation method of SparseTriple.
     |
```

```
     |      Example usage:
     |
     |      >>> SparseTriple(ind1=[0], ind2=[0], val=[1.0])
     |      SparseTriple(ind1 = [0], ind2 = [0], val = [1.0])
     |
     |  isvalid(self)
     |      Tests that ind1, ind2, and val have the same length.
     |
     |      Example usage:
     |
     |      >>> striple = SparseTriple(ind1=[0, 1], ind2=[0, 1],
     |      ...                       val=[1.0, 1.0])
     |      >>> striple.isvalid()
     |      True
     |
     |  unpack(self)
     |      Extracts the indices and values sequences as a tuple.
     |
     |      Returns ind1, ind2, and val as given in __init__.
     |
     |      >>> striple = SparseTriple(ind1=[0, 1], ind2=[0, 1],
     |      ...                       val=[1.0, 1.0])
     |      >>> ind1, ind2, val = striple.unpack()
     |
     |  ----------------------------------------------------------
-----------
     |  Data descriptors defined here:
     |
     |  __dict__
     |      dictionary for instance variables (if defined)
     |
     |  __weakref__
     |      list of weak references to the object (if defined)

    class Stats(builtins.object)
     |  A class whose data members reflect statistics about a CPLEX
     |  problem.
     |
     |  An instance of this class is returned by the `Cplex.get_stat
s()`
     |  method.
     |
     |  The __str__ method of this class returns a string containing
 a
```

```
     |  summary of the problem statistics in human readable form.
     |
     |  An instance of this class always has the following integer m
embers:
     |
     |  * num_objectives
     |  * num_variables
     |  * num_nonnegative
     |  * num_fixed
     |  * num_boxed
     |  * num_free
     |  * num_other
     |  * num_binary
     |  * num_integer
     |  * num_semicontinuous
     |  * num_semiinteger
     |  * num_quadratic_variables
     |  * num_linear_objective_nz
     |  * num_quadratic_objective_nz
     |  * num_linear_constraints
     |  * num_linear_less
     |  * num_linear_equal
     |  * num_linear_greater
     |  * num_linear_range
     |  * num_linear_nz
     |  * num_linear_rhs_nz
     |  * num_indicator_constraints
     |  * num_indicator_less
     |  * num_indicator_equal
     |  * num_indicator_greater
     |  * num_indicator_complemented
     |  * num_indicator_nz
     |  * num_indicator_rhs_nz
     |  * num_quadratic_constraints
     |  * num_quadratic_less
     |  * num_quadratic_greater
     |  * num_quadratic_linear_nz
     |  * num_quadratic_nz
     |  * num_quadratic_rhs_nz
     |  * num_SOS_constraints
     |  * num_SOS1
     |  * num_SOS1_members
     |  * type_SOS1
     |  * num_SOS2
```

```
     |  * num_SOS2_members
     |  * type_SOS2
     |  * num_lazy_constraints
     |  * num_lazy_nnz
     |  * num_lazy_lt
     |  * num_lazy_eq
     |  * num_lazy_gt
     |  * num_lazy_rhs_nnz
     |  * num_user_cuts
     |  * num_user_cuts_nnz
     |  * num_user_cuts_lt
     |  * num_user_cuts_eq
     |  * num_user_cuts_gt
     |  * num_user_cuts_rhs_nnz
     |  * num_pwl_constraints
     |  * num_pwl_breaks
     |
     |  An instance of this class always has the following float mem
bers:
     |
     |  * min_lower_bound
     |  * max_upper_bound
     |  * min_linear_objective
     |  * max_linear_objective
     |  * min_linear_constraints
     |  * max_linear_constraints
     |  * min_linear_constraints_rhs
     |  * max_linear_constraints_rhs
     |
     |  An instance of this class returned by an instance of the Cpl
ex
     |  class with a quadratic objective also has the following floa
t
     |  members:
     |
     |  * min_quadratic_objective
     |  * max_quadratic_objective
     |
     |  An instance of this class returned by an instance of the Cpl
ex
     |  class with ranged constraints also has the following float
     |  members:
     |
     |  * min_linear_range
```

```
     |  * max_linear_range
     |
     |  An instance of this class returned by an instance of the Cpl
ex
     |  class with quadratic constraints also has the following floa
t
     |  members:
     |
     |  * min_quadratic_linear
     |  * max_quadratic_linear
     |  * min_quadratic
     |  * max_quadratic
     |  * min_quadratic_rhs
     |  * max_quadratic_rhs
     |
     |  An instance of this class returned by an instance of the Cpl
ex
     |  class with indicator constraints also has the following floa
t
     |  members:
     |
     |  * min_indicator
     |  * max_indicator
     |  * min_indicator_rhs
     |  * max_indicator_rhs
     |
     |  An instance of this class returned by an instance of the Cpl
ex
     |  class with lazy constraints also has the following float mem
bers:
     |
     |  * min_lazy_constraint
     |  * max_lazy_constraint
     |  * min_lazy_constraint_rhs
     |  * max_lazy_constraint_rhs
     |
     |  An instance of this class returned by an instance of the Cpl
ex
     |  class with user cuts also has the following float members:
     |
     |  * min_user_cut
     |  * max_user_cut
     |  * min_user_cut_rhs
     |  * max_user_cut_rhs
```

```
     |
     |  Methods defined here:
     |
     |  __init__(self, c)
     |      Initialize self.  See help(type(self)) for accurate sign
ature.
     |
     |  __str__(self)
     |      Returns a string containing a summary of the problem
     |      statistics in human readable form.
     |
     |  ----------------------------------------------------------
-----------
     |  Data descriptors defined here:
     |
     |  __dict__
     |      dictionary for instance variables (if defined)
     |
     |  __weakref__
     |      list of weak references to the object (if defined)

DATA
    __all__ = ['Cplex', 'Stats', 'Aborter', 'callbacks', 'exception
s', 'in...
    infinity = 1e+20

VERSION
    12.10.0.0

FILE
    c:\program files (x86)\microsoft visual studio\shared\anaconda3
_64\lib\site-packages\cplex\__init__.py
```