**IBM**

# IBM ILOG CPLEX V12.1

# File formats supported by CPLEX

# Legal terms

(end of license terms of `dtoa` routine of the `gdtoa` package)

# *Table of contents*

        

# *For support*

Explains the prerequisites and procedure for technical support of the product.

## In this section

### Contacting IBM Support

Contains information on how to obtain technical support from IBM worldwide, should you encounter any problems in using IBM products.

# Contacting IBM Support

## IBM Software Support Handbook

This guide contains important information on the procedures and practices followed in the service and support of your IBM products. It does not replace the contractual terms and conditions under which you acquired specific IBM Products or Services. Please review it carefully. You may want to bookmark the site so you can refer back as required to the latest information. The "IBM Software Support Handbook" can be found on the web at *http://www14.software.ibm.com/webapp/set2/sas/f/handbook/home.html*.

## Accessing Software Support

When calling or submitting a problem to IBM Software Support about a particular service request, please have the following information ready:

♦ IBM Customer Number

♦ The machine type/model/serial number (for Subscription and Support calls)

♦ Company name

♦ Contact name

♦ Preferred means of contact (voice or email)

♦ Telephone number where you can be reached if request is voice

♦ Related product and version information

♦ Related operating system and database information

♦ Detailed description of the issue

♦ Severity of the issue in relationship to the impact of it affecting your business needs

## Contact by Web

*Open service requests* is a tool to help clients find the right place to open any problem, hardware or software, in any country where IBM does business. This is the starting place when it is not evident where to go to open a service request.

*Service Request (SR)* tool offers Passport Advantage clients for distributed platforms online problem management to open, edit and track open and closed PMRs by customer number. Timesaving options: create new PMRs with prefilled demographic fields; describe problems yourself and choose severity; submit PMRs directly to correct support queue; attach troubleshooting files directly to PMR; receive alerts when IBM updates PMR; view reports on open and closed PMRs. You can find information about assistance for SR at *http://www.ibm.com/software/support/help-contactus.html*

*System Service Request (SSR)* tool is similar to Electronic Service request in providing online problem management capability for clients with support offerings in place on System

i, System p, System z, TotalStorage products, Linux, Windows, Dynix/PTX, Retail, OS/2, Isogon, Candle on OS/390 and Consul z/OS legacy products.

*IBMLink* SoftwareXcel support contracts offer clients on the System z platform the IBMLink online problem management tool to open problem records and ask usage questions on System z software products. You can open, track, update, and close a defect or problem record; order corrective/preventive/toleration maintenance; search for known problems or technical support information; track applicable problem reports; receive alerts on high impact problems and fixes in error; and view planning information for new releases and preventive maintenance.

## Contact by phone

If you have an active service contract maintenance agreement with IBM , or are covered by Program Services, you may contact customer support teams by telephone. For individual countries, please visit the Technical Support section of the *IBM Directory of worldwide contacts*.

# *File Formats Reference Manual*

This manual documents the file formats supported by IBM® ILOG® CPLEX® . It begins with a brief description of the file formats in alphabetic order. This manual continues with longer explanations of the following topics and formats.

## In this section

**Brief descriptions of file formats**
Defines the file formats CPLEX® recognizes and supports.

**Reading and entering file formats in the Interactive Optimizer**
Describes the file formats appropriate for the Interactive Optimizer

**Saving problems in the Interactive Optimizer**
Explains which format to use to save a problem in the Interactive Optimizer.

**LP file format: matrix models**
Summarizes the rules for the LP file format.

**MPS file format: industry standard**
Describes the industry standard MPS file format and CPLEX® extensions to it.

**Special records in MPS files: CPLEX extensions**
CPLEX® extends the MPS standard in several ways.

**NET file format: network flow models**
Describes NET file format.

**PRM file format: parameter settings**
Describes PRM file format for nondefault parameter settings.

**BAS file format: advanced basis**
Describes BAS file format to support advanced basis.

**MST file format: MIP starts**
Describes the MST file format for MIP starts.

**ORD file format: priorities and branching orders**
Describes ORD file format to support priorities and branching orders.

**SOL file format: solution files**
Describes SOL file format to support solution files.

**FLT file format: filter files for the solution pool**
Describes FLT file format to support filters of the solution pool.

**CSV file format: comma separated values**
Describes CSV file format for comma separated values.

**XML file format: serialized models and solutions**
Describes XML file format to support serialized models and solutions.

# Brief descriptions of file formats

## BAS

files are text files governed by Mathematical Programming System (MPS) conventions (that is, they are not binary) for saving a problem basis. They are documented in *BAS file format: advanced basis*.

## BZ2

is not a file format specific to CPLEX® . Rather, this file extension indicates that a file (possibly in one of the formats that CPLEX® reads) has been compressed by BZIP2. On most platforms, CPLEX® can automatically uncompress such a file and then read data from the file in one of the formats briefly described here

## CLP

is the format CPLEX® uses to represent the conflicting constraints and bounds (a subset of an infeasible model) that were found by the conflict refiner.

## CSV

files contain comma-separated values. Concert Technology offers facilities in CPLEX® for reading and writing such files. See the *CPLEX C++ API Reference Manual* for details, especially the classes `IloCsvReader`, `IloCsvLine`, and `IloCsvReader::Iterator`.

## DPE

is the format CPLEX® uses to write a problem in a binary SAV file after the objective function of the problem has been perturbed.

## DUA

format, governed by MPS conventions, writes the dual formulation of a problem currently in memory so that the MPS file can later be read back in and the dual formulation can then be optimized explicitly. This file format is largely obsolete now since you can use the command `set presolve dual` in the Interactive Optimizer to tell CPLEX® to solve the dual formulation of an LP automatically. (You no longer have to tell CPLEX® to write the dual formulation to a DUA file and then tell CPLEX® to read the file back in and solve it.)

## EMB

is the format CPLEX® uses to save an embedded network it extracts from a problem. EMB files are written in MPS format.

## FLT

is the format CPLEX® uses to save filters for solution pools.

## GZ

is not a file format specific to CPLEX® . Rather, this file extension indicates that a file (possibly in one of the formats that CPLEX® reads) has been compressed by `gzip` , the GNU zip program. On most platforms, CPLEX® can automatically uncompress a gzipped file and then read data from a file in one of the formats briefly described here.

## LP (Linear Programming)

is a CPLEX® -specific file formatted for entering problems in an algebraic, row-oriented form. In other words, LP format allows you to enter problems in terms of their constraints. When you enter problems interactively in the Interactive Optimizer, you are implicitly using LP format. CPLEX® also reads files in LP format. The section *LP file format: matrix models* describes the conventions and use of this format.

## MIN

format for representing minimum-cost network-flow problems was introduced by DIMACS in 1991. More information about DIMACS network file formats is available via anonymous ftp from: `ftp://dimacs.rutgers.edu/pub/netflow/general-info/specs.tex`

## MPS

is an industry-standard, ASCII-text file format for mathematical programming problems. This file format is documented in *MPS file format: industry standard*. Besides the industry conventions, CPLEX® also supports extensions to this format for CPLEX® -specific cases, such as names of more than eight characters, blank space as delimiters between columns, etc. The extensions are documented in *Special records in MPS files: CPLEX extensions*.

## MST

is an XML format available with the CPLEX® MIP optimizer. It is a text format CPLEX® uses to enter a starting solution for a MIP. *MST file format: MIP starts* documents this file format.

## NET

is a CPLEX® -specific ASCII format for network-flow problems. It supports named nodes and arcs. *NET file format: network flow models* offers a fuller description of this file format.

## ORD

is a format available with the CPLEX® MIP optimizer. It is used to enter and to save priority orders for branching. It may contain branching instructions for individual variables. *ORD file format: priorities and branching orders* documents this file format.

## PPE

is the format CPLEX®  uses to write a problem in a binary SAV file after the righthand side has been perturbed.

## PRE

is the format CPLEX®  uses to write a presolved, reduced problem formulation to a binary SAV file. Since a presolved problem has been reduced, it does not correspond to the original problem.

## PRM

is the format CPLEX®  uses to read and write nondefault values of parameters in a file. *PRM file format: parameter settings* documents the format and conventions for reading and writing such files through the Callable Library.

## REW

is a format to write a problem in MPS format with disguised row and column names. This format is simply an MPS file format with all variable (column) and constraint (row) names converted to generic names. Variables are relabeled `x1` through `xn` , and rows are renamed `c1` through `cm` . This format may be useful, for example, for problems that you consider highly proprietary.

## RLP

is the LP format using generic names in the Interactive Optimizer.

## SAV

is a CPLEX® -specific binary format for reading and writing problems and their associated basis information. CPLEX®  includes the basis in a SAV file only if the problem currently in memory has been optimized and a basis exists. This format offers the advantage of being numerically accurate (to the same degree as your platform) in contrast to text file formats that may lose numerical accuracy. It also has the additional benefit of being efficient with respect to read and write time. However, since a SAV file is binary, you cannot read nor edit it with your favorite text editor.

## SOL

files are XML formatted files that contain solution information; they may also provide an advanced start for an optimization.

## XML

as a file format is available to C++ users of Concert Technology to serialize models and solutions (that is, instances of `IloModel` and `IloSolution`). *XML file format: serialized models and solutions* explains more about this serialization API.

# Reading and entering file formats in the Interactive Optimizer

The Interactive Optimizer accepts problems that you read in from files by means of the `read` command or that you enter interactively by means of the `enter` command. When you enter a problem interactively, CPLEX® uses the LP file format; you may save the problem in any supported file format that you choose.

The `read` command of the Interactive Optimizer accepts problem files in LP, MPS, and SAV formats. It also accepts basis files in BAS format. Problems previously saved in DUA, EMB, or REW formats are actually in MPS format. Presolved problems saved with the `pre` option are in SAV format. Problems in which the objective function has been perturbed and the problem saved with the `dpe` option are in SAV format. Problems in which the righthand side has been perturbed and the problem saved with the `ppe` option are in SAV format. Normally, CPLEX® automatically detects which of these file types it is reading; you may also designate the correct file type if CPLEX® does not detect the type automatically.

When CPLEX® reads LP or MPS files, it automatically allocates enough physical memory (if available) to read the problem.

CPLEX® reallocates memory automatically as it is reading from LP and MPS files so it usually **not** necessary to set values for the parameters `ColReadLim`, `RowReadLim`, and `NzReadLim`, but you can set these parameters to the problem sizes so no reallocations need be done. When CPLEX® reads a SAV file, it is not necessary for you to reset these parameters. SAV files contain sufficient information about the size of the problem for CPLEX® to allocate adequate space. For more information on the read limit parameters, see variable (column) read limit, constraint (row) read limit, and nonzero element read limit in the *CPLEX Parameters Reference Manual*.

# Saving problems in the Interactive Optimizer

In the Interactive Optimizer, you save information about the problem currently in memory as a file in the LP, MPS, or SAV formats by means of the `write` command and its options. For a complete list of file formats that CPLEX® supports, see *Brief descriptions of file formats*. Here are the options in the Interactive Optimizer for frequently used formats:

♦ Use the `bas` option to save a problem basis in MPS format.

♦ Use the `clp` option to write a conflict subproblem.

♦ Use the `flt` option to write filters for the solution pool.

♦ Use the `mst` option to write MIP start files. This option also applies to solutions in the solution pool. In fact, an additional option specifies whether to write a single solution specified by its index, or to write all solutions from the solution pool as MIP starts.

♦ Use the `pre` option to write a SAV file for the reduced, presolved problem formulation.

♦ Use the `prm` option to write a file of nondefault parameter settings.

♦ Use the `sol` option to write solution files. This option also applies to solutions in the solution pool. In fact, an additional option specifies whether to write a single solution specified by its index, or to write all solutions from the solution pool.

The SAV file format, because it is binary, is the format that preserves the greatest degree of precision in data. It can be effective in reducing read and write time for repetitively solved problems. However, because it is a binary format, it cannot be readily viewed or edited in standard text editors.

As a naming convention, we recommend that you use the file format for reading the file as the file extension when you write or save the file (for instance, `example.bas` , `example.lp` , `example.mps` , `example.sav`). When you follow this convention, CPLEX® automatically recognizes the file type and eliminates additional prompts for you to specify a file type.

# LP file format: matrix models

CPLEX® provides a facility for entering a problem in a natural, algebraic LP formulation from the keyboard. The problem can be modified and saved from within CPLEX® . This procedure is one way to create a file in a format that CPLEX® can read. An alternative technique is to create a similar file using a standard text editor and to read it into CPLEX® .

The CPLEX® LP format is provided as an input alternative to the MPS file format. An LP format file may be easier to generate than an MPS file if your problem already exists in an algebraic format or if you have an application which generates the problem file more readily in algebraic format (such as a C application). Working with LP files in the *CPLEX User's Manual* in the chapter about managing input and output explains the implications of using LP format rather than MPS format.

CPLEX® accepts any problem saved in an ASCII file provided that it adheres to the following syntax rules.

## Comments

Anything that follows a backslash (\) is a comment and is ignored until a return is encountered. Blank lines are also ignored. Blank lines and comment lines may be placed anywhere and as frequently as you want in the file.

## White space and line length

In general, white space between characters is irrelevant as it is skipped when a file is read. However, white space is not allowed in the keywords used to introduce a new section, such as MAX , MIN , ST , or BOUNDS . Also the keywords must be separated by white space from the rest of the file and must be at the beginning of a line. The maximum length for any name is 255. The maximum length of any line of input is 560.

Skipping spaces may cause CPLEX® to misinterpret (and accept) an invalid entry, such as the following:

```
x1 x2 = 0
```

If the user intended to enter that example as a nonlinear constraint, CPLEX® would instead interpret it as a constraint specifying that one variable named x1x2 must be equal to zero.

To indicate a quadratic constraint in this section, use explicit notation for multiplication and exponentiation (not space).

## Problem sense

The problem statement must begin with the word MINIMIZE or MAXIMIZE , MINIMUM or MAXIMUM , or the abbreviations MIN or MAX , in any combination of upper- and lower-case characters. The word introduces the objective function section.

## Variables

Variables can be named anything provided that the name does not exceed 255 characters, all of which must be alphanumeric (a-z, A-Z, 0-9) or one of these symbols: ! " # $ % & ( ) , . ; ? @ _ ' ' { } ~. Longer names are truncated to 255 characters. A variable name can not begin with a number or a period.

The letter `E` or `e` , alone or followed by other valid symbols, or followed by another `E` or `e` , should be avoided as this notation is reserved for exponential entries. Thus, variables can not be named `e9` , `E-24` , `E8cats` , or other names that could be interpreted as an exponent. Even variable names such as `eels` or `example` can cause a read error, depending on their placement in an input line.

## Objective function

The objective function definition must follow `MINIMIZE` or `MAXIMIZE` . It may be entered on multiple lines as long as no variable, constant, or sense indicator is split by a return. For example, this objective function `1x1 + 2x2 +3x3` can be entered like this:

```
1x1 + 2x2
+ 3x3
```

but not like this:

```
1x1 + 2x
2 + 3x3          \ a bad idea
```

because the second style splits the variable name `x2` with a return.

The objective function may be named by typing a name and a colon before the objective function. The objective function name and the colon must appear on the same line. Objective function names must conform to the same guidelines as variable names. (See the rule about *Variables*). If no objective function name is specified, CPLEX®  assigns the name `obj` .

An objective function may be quadratic. For an example and details about formatting a quadratic objective function, see the rule about *Quadratic terms*.

## Constraints

The constraints section is introduced by the keyword `subject to` . This expression can also appear as `such that` , `st` , `S.T.` , or `ST.` in any mix of upper- and lower-case characters. One of these expressions must precede the first constraint and be separated from it by at least one space.

Each constraint definition must begin on a new line. A constraint may be named by typing a name and a colon before the constraint. The constraint name and the colon must appear on the same line. Constraint names must adhere to the same guidelines as variable names. (See the rule about names of *Variables*.) If no constraint names are specified, CPLEX® assigns the names `c1`, `c2`, `c3`, etc.

The constraints are entered in the same way as the objective function; however, a constraint must be followed by an indication of its sense and a righthand side coefficient. The righthand

side coefficient must be typed on the same line as the sense indicator. Acceptable sense indicators are <, <=, =<, >, >=, =>, and =. These are interpreted as ≤, ≤, ≤, ≥, ≥, ≥, and =, respectively.

For example, here is a named constraint:

```
time: x1 + x2 <= 10
```

Quadratic constraints are allowed in this section. Quadratic terms are specified inside square brackets `[]` as detailed in the rule about *Quadratic terms*. The specification of a quadratic constraint differs from the specification of a quadratic objective in one important way: in a quadratic constraint, the terms are not divided by two; that is, they are not multiplied by 1/2, as they must be in a quadratic objective.

Indicator constraints are also allowed in this section. The rule about *MIP indicator constraints* explains how to specify indicator constraints.

## Bounds

The optional `bounds` section follows the mandatory constraint section. It is preceded by the word `bounds` or `bound` in any mix of lower- and upper-case characters.

Each bound definition must begin on a new line. The format for a bound is $l_n \leq x_n \leq u_n$ except in the following cases.

Upper and lower bounds may also be entered separately as

$l_n \leq x_n$

$x_n \leq u_n$

with the default lower bound of 0 (zero) and the default upper bound of $+\infty$ remaining in effect until the bound is explicitly changed.

Bounds that fix a variable can be entered as simple equalities. For example, `x5 = 5.6` is equivalent to `5.6 <= x5 <= 5.6`.

The bounds $+\infty$ (positive infinity) and $-\infty$ (negative infinity) must be entered as words: `+infinity`, `-infinity`, `+inf`, `-inf`.

A variable with a negative infinity lower bound and positive infinity upper bound may be entered as `free`, in any mix of upper- and lower-case characters, with a space separating the variable name and the word `free`. For example, `x7 free` is equivalent to `- infinity <= x7 <= + infinity`.

## End of file

The file must end with the word `end` in any combination of upper- and lower-case characters, alone on a line, when it is created with the `enter` command. This word is not required for files that are read in to CPLEX® , but it is strongly recommended. Files that have been corrupted can frequently be detected by a missing last line.

## MIP integer variables

This rule applies to the CPLEX®  MIP optimizer.

To specify any of the variables as general integer variables, add a GENERAL section; to specify any of the variables as binary integer variables, add a BINARY section. The GENERAL and BINARY sections follow the BOUNDS section, if one is present; otherwise, they follow the constraints section. Either of the GENERAL or BINARY sections can precede the other. The GENERAL section is preceded by the word GENERAL , GENERALS , or GEN in any mix of upper- and lower-case characters which must appear alone on a line. The following line or lines should list the names of all variables which are to be restricted to general integer values, separated by at least one space. The BINARY section is preceded by the word BINARY , BINARIES , or BIN in any mix of upper- and lower-case characters which must appear alone on a line. The following line or lines should list the names of all variables which are to be restricted to binary integer values, separated by at least one space. Binary variables are automatically given bounds of 0 (zero) and 1 (one), unless alternative bounds are specified in the BOUNDS section, in which case a warning message is issued.

Here is an example of a problem formulation in LP format where x4 is a general integer:

```
Maximize
 obj: x1 + 2 x2 + 3 x3 + x4
Subject To
 c1: - x1 + x2 + x3 + 10 x4 <= 20
 c2: x1 - 3 x2 + x3 <= 30
 c3: x2 - 3.5 x4 = 0
Bounds
 0 <= x1 <= 40
 2 <= x4 <= 3
General
 x4
End
```

If branching priorities or branching directions exist, enter this information through ORD files, as documented in *ORD file format: priorities and branching orders*.

## MIP semi-continuous variables

This rule applies to the CPLEX® MIP optimizer.

To specify any of the variables as semi-continuous variables, that is as variables that may take the value 0 or values between the specified lower and upper bounds, use a SEMI -CONTINUOUS section. This section must follow the BOUNDS , GENERALS , and BINARIES sections. The SEMI-CONTINUOUS section is preceded by the keyword SEMI-CONTINUOUS , SEMI , or SEMIS . The following line or lines should list the names of all the variables which are to be declared semi-continuous, separated by at least one space.

```
Semi-continuous
```

```
x1 x2 x3
```

## MIP special ordered sets

This rule applies to the CPLEX® MIP optimizer. To specify special ordered sets, use an SOS section, which is preceded by the SOS keyword. The SOS section should follow the Bounds, General, Binaries and Semi-Continuous sections. Special ordered sets of type 1 require that, of the variables in the set, one at most may be nonzero. Special ordered sets of type 2 require that at most two variables in the set may be nonzero, and if there are two

nonzeros, they must be adjacent. Adjacency is defined by the weights, which must be unique within a set given to the variables. The sorted weights define the order of the special ordered set. For MIP branch and cut, the order is used to decide how the variables are branched upon. See the *CPLEX User's Manual* for more information. The set is specified by an optional set name followed by a colon and then either of the S1 or S2 keywords (specifying the type) followed by a double colon. The set member names are listed on this line or lines, with their weights. Variable names and weights are separated by a colon, for example:

```
SOS

set1: S1:: x1:10 x2:13
```

## MIP indicator constraints

This rule applies to CPLEX® MIP optimizer.

To specify an indicator constraint, enter it among any other constraints in the model, like this:

```
[constraintname:]  binaryvariable = value  ->  linear constraint
```

The constraint name, followed by a colon, is optional. The hyphen followed by the greater-than symbol (->), separates the indicator variable and its value from the linear constraint that is controlled. The indicator variable must be declared as a binary variable, and the value it is compared to must be either 0 (zero) or 1 (one).

## Quadratic terms

This rule applies to applications licensed to solve problems with quadratic terms in them, that is, quadratic programming problems and quadratically constrained programs (QPs and QCPs). Quadratic coefficients may appear in the objective function. Quadratic coefficients may also appear in constraints under certain conditions. If there are quadratically constrained variables in the problem, see also rules about *Variables*, *Constraints*, and Solving problems with quadratic constraints (QCP) in the *CPLEX User's Manual*.

The algebraic coefficients of the function x'Qx are specified inside square brackets `[]` . The square brackets must be followed by a divide sign followed by the number 2. This convention denotes that all coefficients inside the square brackets will be divided by 2 in evaluating the quadratic terms of the objective function. All quadratic coefficients must appear inside square brackets. Multiple square bracket sections may be specified.

Inside of the square brackets, two variables are multiplied by an asterisk (`*`). For example, `[4x*y]` indicates that the coefficients of both of the off-diagonal terms of Q, corresponding to the variables `x` and `y` in the model are 2, since `4x*y` equals `2x*y + 2x*y` . Each pair of off-diagonal terms of Q is specified only once. CPLEX® automatically creates both off-diagonal entries of Q. Diagonal terms in Q (that is, terms with an exponent of 2) are indicated by the caret (`^` ) followed by 2. For example, `4x^2` indicates that the coefficient of the diagonal term of Q corresponding to the variable `x` in the model is 4.

For example, this problem

Minimize $a + b + 1/2(a^2 + 4ab + 7b^2)$

subject to $a + b \geq 10$ and $a, b \geq 0$

in LP format looks like this:

```
Minimize
obj: a + b + [ a^2 + 4 a * b + 7 b^2 ]/2
Subject To
c1: a + b >= 10
End
```

## Pools of lazy constraints and user-defined cuts

This rule is of interest only to advanced users.

It is possible to include pools of lazy constraints and user defined cuts in an LP file. A pool of lazy constraints or of user defined cuts must not contain any quadratic constraints. For more about these concepts, see User-cut and lazy-constraint pools in the *CPLEX User's Manual*.

# *MPS file format: industry standard*

Describes the industry standard MPS file format and CPLEX®  extensions to it.

## In this section

**Overview of MPS**
Introduces MPS file format.

**Records in MPS format**
Describes indicator records and data records in MPS file format.

**Example of MPS file format**
Shows formatted sample MPS file.

# Overview of MPS

MPS format, long established on mainframe LP systems, has become a widely accepted standard for defining LP problems. In contrast to the CPLEX® LP format, MPS format is a column-oriented format: problems are specified by column (variable) rather than by row (constraint).

# Records in MPS format

MPS data files are analogous to a deck of computer input cards: each line of the MPS file represents a single card record. Records in an MPS data file consist of two types: indicator records and data records. The records contain fields delimited by blank spaces.

## Indicator records

Indicator records separate the individual sections of the MPS file. Each indicator record contains a single word that begins in the first column. There are seven kinds of indicator records, each corresponding to sections of the MPS file. They are listed in *Indicator records* .

*Indicator records*

| Section name/indicator record | Purpose |
|---|---|
| NAME | specifies the problem name; unlike other indicator records, the name record contains data |
| ROWS | specifies name and sense for each constraint |
| COLUMNS | specifies the name assigned to each variable (column) and the nonzero constraint coefficients corresponding to that variable |
| RHS | specifies the names of righthand side vectors and values for each constraint (row) |
| RANGES | specifies constraints that are restricted to lie in the interval between two values; interval endpoints are also specified |
| BOUNDS | specifies the limits within which each variable (column) must remain |
| ENDATA | signals the end of the data; always the last entry in an MPS file |

Each section of the MPS file except the RANGES and BOUNDS sections is mandatory. If no BOUNDS section is present, all variables have their bounds set from 0 (zero) to $+\infty$ (positive infinity). Failure to include an RHS section causes CPLEX® to generate a warning message and set all righthand side values to 0 (zero). Variables and constraints must be declared in the ROWS and COLUMNS sections before they are referenced in the RHS , RANGES , and BOUNDS sections.

## Data records

Data records contain the information that describes the LP problem. Each data record comprises six fields, as in *Fields of a data record in MPS file format.* The fields must be separated by white space (that is, blank space, tab, etc.), and the first field must begin in column 2 or beyond. Not all fields are used within each section of the input file.

*Fields of a data record in MPS file format*

|          | Field 1   | Field 2 | Field 3 | Field 4 | Field 5 | Field 6 |
|----------|-----------|---------|---------|---------|---------|---------|
| Contents | Indicator | Name    | Name    | Value   | Name    | Value   |

Any ASCII character (32 through 126) is legal, but names must contain no embedded blanks. In addition, names over 255 characters are truncated. CPLEX® issues an error message if truncation causes the names to lose their uniqueness. Numeric fields can be at most 25 characters long.

If the first character in Field 3 or 5 is a dollar sign ($), the remaining characters in the record are treated as a comment. Another method for inserting comments is to place an asterisk (*) in column 1. Everything on such a line is treated as a comment.

Values may be defined with decimal or exponential notation and may utilize 25 characters. In exponential notation, plus (+) and minus (-) signs must precede the exponent value. If an exponent value is missing where one is expected, it is assigned a value of 0 (zero).

## The ROWS section

In the ROWS section, each row of the problem is specified with its name and sense, one row per record.

Field 1 contains a single letter designating the sense of each row. Acceptable values are:

♦ N indicates a free row.

♦ G indicates a greater-than-or-equal-to row.

♦ L indicates a less-than-or-equal-to row.

♦ E indicates an equality row.

Field 2 contains a character identifier, maximum length of 255 characters, specifying the name of the row.

Fields 3-6 are not used in the ROWS section.

If more than one free row is specified, the first one is used as the objective function and the others are discarded.

The ROWS section of our example looks like this:

```
ROWS
 N  obj
 L  c1
 L  c2
```

## The COLUMNS section

In the COLUMNS section, all the columns of the constraint matrix are specified with their name and all of the nonzero elements. Multiple records may be required to completely specify a given column.

Field 1: Blank

Field 2: Column identifier

Field 3: Row identifier

Field 4: Value of matrix coefficient specified by Fields 2 and 3

Field 5: Row identifier (optional)

Field 6: Value of matrix coefficient specified by Fields 2 and 5 (optional)

After a matrix element is specified for a column, all other nonzero elements in that same column should be specified.

The COLUMNS section of our example looks like this:

```
COLUMNS
    x1        obj                 -1   c1                  -1
    x1        c2                   1
    x2        obj                 -2   c1                   1
    x2        c2                  -3
    x3        obj                 -3   c1                   1
    x3        c2                   1
```

## The RHS section

In the RHS section, the nonzero righthand-side values of the constraints are specified.

Field 1: Blank

Field 2: RHS identifier

Field 3: Row identifier

Field 4: Value of RHS coefficient specified by Field 2 and 3

Field 5: Row identifier (optional)

Field 6: Value of RHS coefficient specified by Field 2 and 5 (optional)

Several RHS vectors can exist. The name of each RHS vector appears in Field 2. However, only the first RHS vector is selected when a problem is read. Additional RHS vectors are discarded.

The RHS section of our example looks like this:

```
RHS
    rhs       c1                  20   c2                  30
```

## The RANGES section

In the RANGES section, RHS range values to be applied to constraints may be specified.

Field 1: Blank

Field 2: Righthand side range vector identifier

Field 3: Row identifier

Field 4: Value of the range applied to row specified by Field 3

Field 5: Row identifier (optional)

Field 6: Value of the range applied to row specified by Field 5 (optional)

The effect of specifying a righthand side range depends on the sense of the specified row and whether the range has a positive or negative coefficient. *How range values are interpreted in data records of MPS files* specifies how range values are interpreted. For a given row, `rhs` is the righthand side value and `range` is the corresponding range value.

***How range values are interpreted in data records of MPS files***

| Row type | Range value sign | Resulting rhs upper limit | Resulting rhs lower limit |
|---|---|---|---|
| G | + or - | rhs + \|range\| | rhs |
| L | + or - | rhs | rhs - \|range\| |
| E | + | rhs + range | rhs |
| E | - | rhs | rhs + range |

The name of each range vector appears in Field 2. More than one range vector can be specified within an MPS file. However, only the first range vector is selected when a problem is read. Additional range vectors are discarded.

In our example, there are no ranged rows, but suppose we want to add the following constraint to our problem:

```
x1 - 3x2 + x3 >= 15
```

Instead of explicitly adding another row to the problem, we can represent this additional constraint by modifying row 2 of the example to make it a ranged row in this way:

```
15 <= x1 - 3x2 + x3 <= 30
```

The `RANGES` section of the MPS file to support this modification looks like this:

```
RANGES
    rhs         c2                      15
```

The name of each range vector appears in Field 2. However, only the first range vector is selected when a problem is read. Additional range vectors are discarded.

## The BOUNDS section

In the `BOUNDS` section, bound values for variables may be specified.

Field 1: Type of bound. Acceptable values are:

♦ `LO` Lower bound

♦ `UP` Upper bound

♦ FX Fixed value (upper and lower bound the same)

♦ FR Free variable (lower bound -∞ and upper bound +∞)

♦ MI Minus infinity (lower bound = -∞)

♦ PL Plus infinity (upper bound = +∞)

Field 2: Bound identifier

Field 3: Column identifier to be bounded

Field 4: Value of the specified bound

Fields 5 and 6 are not used in the BOUNDS section.

In our example, the BOUNDS section looks like this:

```
BOUNDS
 UP BOUND      x1                    40
```

If no bounds are specified, CPLEX® assumes a lower bound of 0 (zero) and an upper bound of +∞. If only a single bound is specified, the unspecified bound remains at 0 or +∞, whichever applies, with one exception. If an upper bound of less than 0 is specified and no other bound is specified, the lower bound is automatically set to -∞. CPLEX® deviates slightly from a convention used by some MPS readers when it encounters an upper bound of 0 (zero). Rather than automatically set this variable's lower bound to -∞, CPLEX® accepts both a lower and upper bound of 0, effectively fixing that variable at 0. CPLEX® resets the lower bound to -∞ only if the upper bound is less than 0. A warning message is issued when this exception is encountered.

More than one bound vector may exist. The name of each bound vector appears in Field 2. However, only the first bound vector is selected when a problem is read. Additional bound vectors are discarded.

# Example of MPS file format

```
NAME          example2.mps
ROWS
 N  obj
 L  c1
 L  c2
COLUMNS
    x1        obj             -1   c1              -1
    x1        c2               1
    x2        obj             -2   c1               1
    x2        c2              -3
    x3        obj             -3   c1               1
    x3        c2               1
RHS
    rhs       c1              20   c2              30
BOUNDS
 UP BOUND     x1              40
ENDATA
```

# *Special records in MPS files: CPLEX extensions*

CPLEX®  extends the MPS standard in several ways.

## In this section

### Overview of MPS extension
Introduces extensions to the MPS file format to support special features of CPLEX® .

### Objective sense and name in MPS files
Describes the CPLEX®  extension of MPS files to support name and sense of an objective function.

### Integer variables in MPS files
Describes CPLEX®  extensions of MPS file format to support integer variables.

### Special ordered sets (SOS) in MPS files
Describes CPLEX®  extensions of MPS file format to support special ordered sets.

### Quadratic objective information in MPS files
Describes CPLEX®  extensions of MPS file format to support an objective function containing quadratic terms.

### Quadratically constrained programs (QCP) in MPS files
Describes CPLEX®  extensions of MPS file format to support quadratically constrained programs.

### Indicator constraints in MPS files
Describes CPLEX®  extensions of MPS file format to support indicator constraints.

**User defined cuts in MPS files**
Describes CPLEX® extensions of MPS file format to support user defined cuts.

**Lazy constraints in MPS files**
Describes CPLEX® extensions of MPS file format to support lazy constraints.

# Overview of MPS extension

Historically, MPS format (including CPLEX® MPS format for CPLEX® version 2.1 and earlier releases) included restrictions, such as requiring input fields to occupy fixed columnar positions and limiting all names to a length of 8 characters or fewer. In CPLEX® version 3.0 and subsequent releases, these restrictions were relaxed. The current CPLEX® MPS format is actually an extended version of the historical MPS format. To allow for these extensions, certain practices which were accepted in MPS files for older CPLEX® releases and other systems are no longer permitted. For example, since CPLEX® no longer requires fixed columnar positions, blank spaces are interpreted as delimiters. Older MPS files containing names with embedded spaces therefore become unreadable. To maintain compatibility with earlier versions as well as MPS files from other systems, CPLEX® provides an MPS file conversion utility which translates older files into the newer CPLEX® MPS format. The section Converting file formats in the *CPLEX User's Manual* explains how to use the file conversion utility.

# Objective sense and name in MPS files

CPLEX® extends the MPS standard by allowing two additional sections: OBJSEN and OBJNAME . They may be specified after the NAME section. OBJSEN sets the objective function sense, and OBJNAME selects an objective function from among the free rows within the file. If neither of these sections appears in the MPS file, CPLEX® assumes that the problem is a minimization and that the objective function is the first free row encountered in the ROWS section. If these options are used, they must appear in order and as the first and second sections after the NAME section. The values for OBJSENSE can be MAX or MIN .

Here is an example of these optional sections:

```
NAME            example.mps
OBJSENSE
  MAX
OBJNAME
  rowname
```

# Integer variables in MPS files

If you use the CPLEX® mixed integer optimizer, then you may restrict any or all variables to integer values. CPLEX® accepts two commonly used ways of extending the MPS file format to include integer variables: in the COLUMNS section or in the BOUNDS section.

In the first way, integer variables are identified within the COLUMNS section of the MPS file by marker lines. A marker line is placed at the beginning and end of a range of integer variables. Multiple sets of marker lines are allowed. Integer marker lines have a field format consisting of Fields 2 through 4.

Field 2: Marker name

Field 3: 'MARKER' (including the single quotation marks)

Field 4: Keyword 'INTORG' and 'INTEND' to mark beginning and end respectively (including the single quotation marks)

Fields 5 and 6 are ignored.

The marker name must differ from the preceding and succeeding column names.

If no bounds are specified for the variables within markers, bounds of 0 (zero) and 1 (one) are assumed.

In the following example, column x4 is an integer variable and looks like this in the COLUMNS section of an MPS file, according to this first way of treating integer variables:

```
NAME
ROWS
 N  obj
 L  c1
 L  c2
 E  c3
COLUMNS
    x1        obj                -1   c1                 -1
    x1        c2                  1
    x2        obj                -2   c1                  1
    x2        c2                 -3   c3                  1
    x3        obj                -3   c1                  1
    x3        c2                  1
    MARK0000  'MARKER'                'INTORG'
    x4        obj                -1   c1                 10
    x4        c3               -3.5
    MARK0001  'MARKER'                'INTEND'
RHS
    rhs       c1                 20   c2                 30
BOUNDS
 UP BOUND     x1                 40
 LO BOUND     x4                  2
 UP BOUND     x4                  3
ENDATA
```

In the second way of treating integer variables, integer variables are declared in the BOUNDS section with special bound types in Field 1. The acceptable special bound types appear in *Special bound types for handling integer variables in MPS files*.

***Special bound types for handling integer variables in MPS files***

| Type | Purpose | Special Considerations |
|------|---------|------------------------|
| BV | Binary variable | Field 4 must be 1.0 or blank |
| LI | Integer lower bound | Field 4 is the lower bound value and must be an integer |
| SC | Semi-continuous variable | Field 4 is the upper bound and must be specified |
| UI | Integer upper bound | Field 4 is the upper bound value and must be an integer |

To specify general integers with no upper bounds, use LI with the value 0.0.

For example, column x4 is an integer variable declared in the BOUNDS section of an MPS file, according to this second way of treating integer variables:

```
NAME
ROWS
 N   obj
 L   c1
 L   c2
 E   c3
COLUMNS
     x1        obj                  -1    c1                  -1
     x1        c2                    1
     x2        obj                  -2    c1                   1
     x2        c2                   -3    c3                   1
     x3        obj                  -3    c1                   1
     x3        c2                    1
     x4        obj                  -1    c1                  10
     x4        c3                 -3.5
RHS
     rhs       c1                   20    c2                  30
BOUNDS
 UP BOUND     x1                   40
 LI BOUND     x4                    2
 UI BOUND     x4                    3
ENDATA
```

# Special ordered sets (SOS) in MPS files

If you use the CPLEX® mixed integer optimizer (that is, the MIP optimizer), then you may define special ordered sets (SOS) in MPS format.

The convention for SOS uses set declaration lines and member declaration lines, both of which begin in column 2 or beyond. In a set declaration line, columns 2 and 3 specify S1 or S2. Optionally, the name of a set is specified in column 4. In a member declaration line, column 5 or beyond specifies a variable name. Note that in an MPS file, the SOS section must follow the BOUNDS section.

If weighting information is provided, it follows the member name in a member declaration line.

In the following example, an SOS section is placed after the BOUNDS section:

```
NAME
ROWS
 N  obj
 L  c1
 L  c2
 E  c3
COLUMNS
    x1        obj              -1   c1              -1
    x1        c2                1
    x2        obj              -2   c1               1
    x2        c2               -3   c3               1
    x3        obj              -3   c1               1
    x3        c2                1
    x4        obj              -1   c1              10
    x4        c3             -3.5
RHS
    rhs       c1               20   c2              30
BOUNDS
 UP BOUND     x1               40
 LI BOUND     x4                2
 UI BOUND     x4                3
SOS
 S1 set1
    x1                      10000
    x2                      20000
    x4                      40000
ENDATA
```

## 'MARKER' lines for SOS in MPS files

'MARKER' lines are used to delimit SOS in the COLUMNS section of an MPS file, much like using integer markers. (The single quotation mark before and after the term is necessary.) The names of the sets are specified in the second field, starting in column 4 or beyond. Names of sets must be unique. The 'MARKER' lines must come in pairs of an 'SOSORG' and 'SOSEND' surrounding the columns that are in the SOS. Optionally, in Field 1 of a 'MARKER' .
. . . 'SOSORG' line, either S1 or S2 may be specified to indicate the type of the SOS. An

SOS `'MARKER'` line without an S1 or S2 indicator is assumed to denote an S1 set. Members of an SOS may or may not be integer or binary variables.

There is no requirement that there be a constraint that all members of an SOS sum to 1.0 (nor is any such constraint implicit). However, providing such a constraint in your formulation may be desirable as it may strengthen the LP relaxation of the mixed integer problem, as for example in the case of an S1 set consisting of binary variables.

In the following example, the excerpt from the COLUMNS section of an MPS file defines an SOS Type 1 set consisting of x5 and x6. which may be continuous or integer variables.

```
S1  NAME1       'MARKER'                 'SOSORG'
    x5          obj              -9      c1      5
    x5          c2               3
    x6          obj              -6      c1      8
    x6          c3               -4.5
    NAME1       'MARKER'                 'SOSEND'
```

The SOS `'MARKER'` lines can appear between integer `'MARKER'` lines (if all members of the SOS are integer), or integer `'MARKER'` lines can appear between SOS `'MARKER'` lines (if some members of the SOS are non-integer).

The MARKER format cannot accommodate overlapping SOSs. That is, a variable cannot be a member of two special ordered sets. Overlapping SOSs can, however, be specified by the CPLEX® SOS format, documented in *Special ordered sets (SOS) in MPS files*.

## REFROW section for SOS in MPS files

A REFROW section may be included immediately before the ROWS section. It consists of exactly one record line with the name of the reference row starting in Field 2. The specified row must also be defined in the ROWS section. The nonzeros of the reference row are used as weights within an SOS. All weights within one SOS must be unique values. A REFROW section is optional; if no reference row is specified, the weighting values 1, . . . , n is given to the n members of an SOS in the order in which they are read. In other words, without specific reference row information, it is assumed that the user has ordered the SOS variables in ascending order with respect to some relevant criterion (for example, in importance, capacity, objective weighting, or cost).

# Quadratic objective information in MPS files

If you use the CPLEX® barrier optimizer for quadratic programming problems (QPs), then you can specify quadratic objective coefficients in MPS format in a QMATRIX section.

Following the BOUNDS section, a QMATRIX section may be specified. Each line of this section defines one nonzero coefficient of the matrix Q. Each line should contain two variable names (which must have been specified in the COLUMNS section) in Fields 2 and 3, followed by a nonzero coefficient value in Field 4. For each off-diagonal coefficient, two lines must appear: one for the lower triangular element, and one for the upper triangular element. CPLEX® evaluates the quadratic part of the objective function as 0.5 x'Qx, when the coefficients of Q are specified in an MPS file.

For example, consider the following problem:

 Minimize

$$a + b + 1/2 \ (a2 + 4ab + 7b2)$$

 subject to

$$a + b \quad 10$$

$$a, \ b \quad 0$$

In MPS format, you may enter the problem in the following way:

```
NAME          problem
ROWS
 N  obj
 G  c1
COLUMNS
    a         obj              1   c1                  1
    b         obj              1   c1                  1
RHS
    rhs       c1              10
QMATRIX
    a         a                1
    a         b                2
    b         a                2
    b         b                7
ENDATA
```

You can also enter the quadratic objective coefficients by using a QUADOBJ section. In this format, only the upper diagonal elements of the Q matrix are entered. For the same example, the input with a QUADOBJ section looks like this:

```
NAME          problem
ROWS
```

```
 N  obj
 G  c1
COLUMNS
    a         obj              1  c1                    1
    b         obj              1  c1                    1
RHS
    rhs       c1              10
QUADOBJ
    a         a                1
    a         b                2
    b         b                7
ENDATA
```

If you have a model with quadratic objective information in MPS format in a QUADOBJ section of the following form, you do not have to convert your file in order for CPLEX® to make use of it.

```
varname1 varname2 value
```

CPLEX® can read that file and interpret the QUADOBJ section correctly. However, the MPS file writers of CPLEX® do not produce a QUADOBJ section themselves. Instead, they produce a QMATRIX section, as explained here.

# Quadratically constrained programs (QCP) in MPS files

As explained in the *CPLEX User's Manual* in Solving problems with quadratic constraints (QCP), CPLEX® can solve problems with quadratic terms among the constraints if the Q matrix for the quadratic term is positive semi-definite and the quadratic function defines a convex region. CPLEX® has extended the MPS format to accommodate QCP models.

The quadratic constraints of such a model are listed in the ROWS section, and their linear coefficients appear in the COLUMNS section, just the same as coefficients from the linear constraints.

The quadratic terms go in QCMATRIX sections, one QCMATRIX per quadratic constraint. QCMATRIX sections appear after the optional SOS section. They may appear either after or before the QMATRIX (objective) section.

The name of the constraint appears on the same line after QCMATRIX .

The quadratic terms of the quadratic expression must be given as a symmetric matrix. That is, if there is an entry for $Q_{ij}$, then there must be an identical entry for $Q_{ji}$ when i is not equal to j. This requirement is the same as for the QMATRIX section, where any quadratic terms in the objective function are declared. The formats of the Q parts are the same.

# Indicator constraints in MPS files

Indicator constraints provide a way for you to express relations among variables by identifying a binary (0-1) variable to control whether or not a given constraint is active. CPLEX® has extended the MPS format to express indicator constraints. The constraints to be controlled by the binary variable are listed in the ROWS section; their linear coefficients appear in the COLUMNS section (that is, the same as coefficients from linear constraints). Only rows of types E, L, and G may be part of indicator constraints. In other words, a row of type N cannot appear as a constraint controlled by a binary variable in this sense (that is, an indicator constraint).

The binary variables that control the linear constraints are specified in the BOUNDS section or with MARKER lines (that is, like any other binary variable). The relationship between the binary variables and the constraints they control is specified in the INDICATORS section. The INDICATORS section follows any quadratic constraint section and any quadratic objective section. Each line of the INDICATORS section has a type field starting in column 2 or beyond; the type must be "IF" followed by the name of the row of the indicator constraint, the name of the binary variable, and finally the value 0 (zero) or 1 (one) to indicate when the constraint should be active.

Rows that appear in the INDICATORS section cannot be ranged rows. In other words, a row that appears in the RANGES section cannot appear also in the INDICATORS section.

Here is an example of an INDICATORS section:

```
NAME          ind1.mps
ROWS
 N  obj
 L  row2
 L  row4
 E  row1
 E  row3
COLUMNS
    x         obj                    -1
    x         row2                    1
    x         row4                    1
    x         row1                    1
    y         row4                    1
    z         row4                    1
    z         row3                    1
RHS
    rhs       row2                   10
    rhs       row4                   15
BOUNDS
 UI bnd       y                       1
INDICATORS
 IF row1      y                       1
 IF row3      y                       0
ENDATA
```

That declaration represents the following model:

```
Minimize
 obj: - x
Subject To
 row2: x <= 10
 row4: x + y + z <= 15
 row1: y = 1 -> x  = 0
 row3: y = 0 -> z  = 0
Bounds
 0 <= y <= 1
Binaries
 y
End
```

# User defined cuts in MPS files

The advanced feature user defined cuts can be declared in a special section following the ROWS section. The title of this section is USERCUTS . The order of sections must be ROWS USERCUTS . The format of the USERCUTS section is the same as the format of the ROWS section with this exception: the type must be one of E, L, or G; the row must not be ranged. For more information about user defined cuts, see User-cut and lazy-constraint pools in the *CPLEX User's Manual*.

# Lazy constraints in MPS files

The advanced feature lazy constraints can be declared in a special section following the ROWS and USERCUTS sections. The title of this section is LAZYCONS . The order of sections must be ROWS USERCUTS LAZYCONS . The format of the LAZYCONS section is the same as the format of the ROWS section with this exception: the type must be one of E, L, or G; the row must not be ranged. For more information about lazy constraints and an example of an MPS file extended to include them, see User-cut and lazy-constraint pools in the *CPLEX User's Manual*.

# NET file format: network flow models

The NET file format is an ASCII file format specific to CPLEX® for network-flow problems. It is the recommended file format for representing pure network problems within CPLEX® . This format is supported by Concert Technology, by the Callable Library, and by the Interactive Optimizer. In particular, it works with `CPXNETptr` objects (not `CPXLPptr` objects).

## Comments

This is a free-format file; that is, line breaks or column positions are irrelevant to the interpretation of the file. The only exceptions to this convention are comments: anything from a backslash (\) character to the end of a line is a comment and does not contribute to the network specified by the file. Comments are allowed anywhere in the file.

## Keywords

The NET format recognizes the following keywords in a file:

♦ MAXIMIZE

♦ MINIMIZE

♦ NETWORK

♦ ENDNETWORK

♦ SUPPLY

♦ DEMAND

♦ ARCS

♦ BOUNDS

♦ OBJECTIVE

♦ INFINITY

♦ FREE

Keywords are independent of character case. Keywords must be separated by white space from other symbols in the file.

## White space

White space consists of one or more of the following:

♦ the space character

♦ the tab character (\t),

♦ the new line character (\n)

♦ a comment (that is, all characters following a backslash to the end of a line)

## Abbreviations of keywords

Also, the NET format recognizes the abbreviations summarized in *Abbreviations of Keywords in NET File Format*.

***Abbreviations of Keywords in NET File Format***

| Keyword | Abbreviation |
|---------|--------------|
| INFINITY | INF |
| MINIMIZE | MIN |
| MAXIMIZE | MAX |

## Start of a NET file

A NET file must start with one of the following keywords:

♦ MAXIMIZE NETWORK

♦ MINIMIZE NETWORK

Both may be followed optionally by the name of a problem. If no name is specified, the filename will be used instead. This part of a NET file is referred to as the **start** of a NET file.

## Names in a NET file

Names must follow the same conventions as they do for CPLEX® LP format files. They must consist of a sequence of alphanumeric characters (a-z, A-Z, or 0-9) or one of the symbols: ! " # $ % & ( ) / , . ; @ _ ' ' { } | ~. However, the first character may **not** be a digit or period (.). No names corresponding to the keywords are allowed. There is no restriction on the number of characters in a name within a NET file.

## End of a NET file

The network specification of a NET file must end with the keyword ENDNETWORK . Anything following the keyword ENDNETWORK will be ignored. This keyword is referred to as the end of a NET file.

## Sections of a NET file

Between its start and end, a NET file is divided into sections. Each section is introduced by its keyword and continues until the next section begins or the NET file ends. The keywords introducing sections are SUPPLY , DEMAND , ARCS , BOUNDS , and OBJECTIVE . Each section keyword may appear more than once in a NET file. They need not be in any order.

## The SUPPLY section

In this section, supply values for nodes are specified. Each supply value is specified with the following sequence:

```
node-name : value
```

where `node-name` specifies the name of the node for which to set a supply value, and `value` is the value that will be assigned to node `node-name` as its supply value. If a node with this name does not already exist, a new node will be created with this name. If the node has been previously assigned a supply value, the new value overrides the previous value, and a warning will be issued.

## The DEMAND section

This section corresponds to the `SUPPLY` section except that it specifies demand values instead of supply values. That is, instead of specifying a supply value s in the `SUPPLY` section, you can specify the negative of s in the `DEMAND` section and vice versa. The format for doing so is exactly the same: `node-name : value` . There is no requirement to use both a `SUPPLY` and a `DEMAND` section in a given model. You can fully specify any model using either of the section types alone by correctly using positive and negative values. The availability of either or both section types simply offers flexibility in model formulation.

## The ARCS section

In this section, the arcs from-node (or tail) and to-node (or head) are specified. For each arc, the format is:

```
arc-name : from-node -> to-node
```

where `arc-name` specifies the name for the arc from `from-node` to `to-node` . If `arc-name` already exists, a warning message is issued, and the specified nodes override the previous ones. The nodes are referred to by node names. If a node does not yet exist, a new node with this name will be created with supply value 0 (zero). Otherwise, the existing node of the specified name will be used.

## The OBJECTIVE section

This section is used to assign objective values to arcs in the format:

```
arc-name : value
```

where `arc-name` must be the name of an arc that has previously been specified in an `ARCS` section. This arc will be assigned the objective value indicated by `value` . If an arc is assigned an objective value more than once, a warning message will be issued, and the most recently assigned objective value for that arc in the file will be used. If no objective value is specified for an arc, 0 (zero) will be used by default.

## The BOUNDS section

In this section, bounds on the flow through an arc are specified in a variety of ways, similar to specifying bounds on variables in LP format. The general format is:

```
value1 <= arc-name <= value2
```

That general statement assigns a lower bound of `value1` and an upper bound of `value2` to the arc named `arc-name` . This arc must have previously been defined in an `ARCS` section.

Only one bound at a time may be specified for an arc. That is, the following are valid inputs: `value <= arc-name` to set the lower bound of the specified arc to `value` or `arc -name <= value` to set the upper bound of the specified arc to `value`. If the upper and lower bound for an arc are identical, you can write `arc-name = value` instead.

Bound values may be `INFINITY` or `-INFINITY`. An arc with lower bound `-INFINITY` and upper bound `INFINITY` may be entered as `FREE`, like this: `arc-name free`

If a bound is not specified for an arc, 0 (zero) will be used as the default lower bound and infinity as the default upper bound.

## Example of NET file format

```
\ Except for this comment, this is the example network file
\ created by netex1.c
\
MINIMIZE NETWORK netex1
SUPPLY
      n1 : 20
      n4 : -15
      n5 : 5
      n8 : -10
ARCS
      a1 :        n1 ->        n2
      a2 :        n2 ->        n3
      a3 :        n3 ->        n4
      a4 :        n4 ->        n7
      a5 :        n7 ->        n6
      a6 :        n6 ->        n8
      a7 :        n5 ->        n8
      a8 :        n5 ->        n2
      a9 :        n3 ->        n2
     a10 :        n4 ->        n5
     a11 :        n4 ->        n6
     a12 :        n6 ->        n4
     a13 :        n6 ->        n5
     a14 :        n2 ->        n6
OBJECTIVE
      a1 : 3
      a2 : 3
      a3 : 4
      a4 : 3
      a5 : 5
      a6 : 6
      a7 : 7
      a8 : 4
      a9 : 2
     a10 : 6
     a11 : 5
     a12 : 4
     a13 : 3
     a14 : 6
BOUNDS
```

```
18 <= a1  <= 24
 0 <= a2  <= 25
      a3   = 12
 0 <= a4  <= 10
 0 <= a5  <= 9
      a6     free
 0 <= a7  <= 20
 0 <= a8  <= 10
 0 <= a9  <= 5
 0 <= a10 <= 15
 0 <= a11 <= 10
 0 <= a12 <= 11
 0 <= a13 <= 6
ENDNETWORK
```

# PRM file format: parameter settings

It is possible to read and write a file of parameter settings with the Callable Library. This kind of file is known as a PRM file. The file extension for a PRM file is `.prm` . The Callable Library routine `CPXreadcopyparam` reads parameter values from a file with the `.prm` extension. The routine `CPXwriteparam` writes a file of the current nondefault parameter settings to a file with the `.prm` extension. Here is the format of such a file:

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

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

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

**Tip**: The first line of a PRM file is significant to CPLEX® . An easy way to produce a correctly formatted PRM file is to have CPLEX®  write it for you.

The comment character in a parameter file is **#**. CPLEX®  ignores the rest of the line.

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

Here is an example of such a file:

```
CPLEX Parameter File Version 11.0.0
CPX_PARAM_EPPER                 3.45000000000000e-06
CPX_PARAM_OBJULIM               1.23456789012345e+05
CPX_PARAM_PERIND                1
CPX_PARAM_SCRIND                1
CPX_PARAM_WORKDIR               "tmp"
```

# BAS file format: advanced basis

An MPS basis file, known as a BAS file, contains the information needed by CPLEX® to define an advanced basis. Like an MPS file, the BAS file begins with a `NAME` indicator record and ends with an `ENDATA` record.

> **Tip**: A BAS file is a text-based format that relies on each variable and each constraint having a name. If names do not exist, they will be created automatically, as needed, during a write operation. If you anticipate reading and writing BAS files, it is a good idea to assign a name to each variable (column) and to each constraint (row) yourself when you create the model.

A basis defines a list of basic structural variables and row variables. A structural variable is one of the variables (columns) defined in the MPS problem file. A row variable is actually the slack, surplus, or artificial variable associated with a row.

For linear programs, the total number of basic variables—both structural and row—is equal to the number of rows in the constraint matrix. Additionally, the number of basic structural variables is equal to the number of nonbasic row variables. By convention, an MPS basis file is built on the assumption that all row variables are basic and that all structural variables are nonbasic with values at their lower bound. The data records in a BAS file list structural and row variables that violate this assumption. This convention minimizes the size of the BAS file.

For quadratic programs, the total number of basic variables can exceed the number of rows and so not all basic variables can be paired with a nonbasic row variable.

*Status indicators for variables in a BAS file*

| Value | Status |
|-------|--------|
| XU | Variable 1 is basic; variable 2 is nonbasic at its upper bound |
| XL | Variable 1 is basic; variable 2 is nonbasic at its lower bound |
| UL | Variable 1 is nonbasic and is at its upper bound |
| LL | Variable 1 is nonbasic and is at its lower bound |
| BS | Variable 1 is basic. |

Field 1: Indicator specifying status of the named variables in Fields 2 and 3. Acceptable values appear in *Status indicators for variables in a BAS file*.

Field 2: Variable 1 identifier

Field 3: Variable 2 identifier (ignored if Field 1 is `UL`, `LL` or `BS`)

Variable 1 specifies a structural variable identifier which has entered the basis. By convention, this structural variable must displace one of the row variables. Variable 2 is a row variable

that has left the basis. No relationship between structural variables entering the basis and row variables leaving the basis is implied within the BAS file.

In the *Example of MPS file format*, variables x2 and x3 are basic and the two constraints (row variables) are nonbasic. Also, x1 was forced to its upper limit of 40. The optimal basis for that example appears in the following sample. CPLEX® adds the number of iterations to the NAME record. The iteration count is useful if the basis file was automatically generated during a previously aborted run. The XL indicator in the first two data records indicates that x3 and x2 are basic and that the row variables for c1 and c2 are nonbasic at their lower bound. The third record shows that structural variable x1 is nonbasic and at its upper bound.

```
NAME            example2.bas   Iterations 3   Rows 2   Cols 3
 XL x3          c1
 XL x2          c2
 UL x1
ENDATA
```

# MST file format: MIP starts

If you use the CPLEX® MIP optimizer, the MST file format is available to specify MIP start values for particular variables, most commonly the integer variables. MST files are of the same format as SOL files. While SOL files have values for all variables, MST files written by CPLEX® at the default write level of detail have values only for the integer variables and members of special ordered sets (SOS); at higher write levels of detail, MST files may also contain both discrete and continuous variables. For more information about the write level for MST, SOL files, see the *CPLEX Parameters Reference Manual* (WriteLevel, CPX_PARAM_WRITELEVEL).

MST file format also supports MIP starts from members of the solution pool. See details about the Concert Technology methods cplex.writeMIPStarts and the Callable Library routine CPXmstwritemipstarts, documented in their respective reference manuals.

CPLEX® uses the start values in an MST file only if the advanced indicator parameter is on (that is, set to 1 (one) its default).

♦ In Concert Technology, use the method IloCplex::setParam(AdvInd 1).

♦ In the Callable Library, use the routine CPXsetintparam(env, CPX_PARAM_ADVIND, 1).

♦ In the Interactive Optimizer, use the command set advance 1.

Here is an example of an MST file:

```
<?xml version = "1.0" standalone="yes"?>
<?xml-stylesheet href="https://www.ilog.com/products/cplex/xmlv1.0/solution.
xsl" type="text/xsl"?>
<CPLEXSolution version="1.0">
 <header
   problemName="../../../examples/data/mexample.mps"
   objectiveValue="-122.5"
   solutionTypeValue="3"
   solutionTypeString="primal"
   solutionStatusValue="101"
   solutionStatusString="integer optimal solution"
   MIPNodes="0"
   MIPIterations="3"/>
 <quality
   epInt="1e-05"
   epRHS="1e-06"
   maxIntInfeas="0"
   maxPrimalInfeas="0"
   maxX="40"
   maxSlack="2"/>
 <variables>
  <variable name="x4" index="3" value="3"/>
 </variables>
</CPLEXSolution>
```

# ORD file format: priorities and branching orders

If you use the CPLEX® MIP optimizer, the ORD file format is available to indicate priority orders and branching directions for specific variables. Variables that are not given an explicit priority or that do not appear in an ORD file are assigned 0 (zero) priority. An ORD file begins with a NAME indicator record and ends with an ENDATA record.

Integer variables are specified, one per line, with an optional branching direction (UP or DN) beginning in column 2 and 3. Names begin in column 5 or beyond. The variable name and its priority must be separated by one or more blank spaces.

Here is an example of an ORD file:

```
NAME
     x3                               10
 DN x5                                 5
 UP x7
ENDATA
```

ORD files created using CPLEX® versions 2.1 or earlier used a fixed format in which the various data fields were limited to eight characters in length and restricted to specific columnar positions in each line. The extensions provided in the new CPLEX® ORD file reader allow for more descriptive names and greater overall input flexibility. Most fixed-format ORD files conform to the new format. Any files that do not conform can be converted to the new format using the convert utility that comes with the standard CPLEX® distribution. Converting file formats explains how to use that utility.

# SOL file format: solution files

CPLEX®  enables you to read and write solution files, formatted in XML, for all problem types, for all application programming interfaces (APIs). These solution files, known as SOL files, carry the file extension `.sol`. The XML solution file format makes it possible for you to display and view these solution files in most browsers as well as to pass the solution to XML-aware applications. CPLEX®  also provides a stylesheet and schema in *yourCplexinstallation* `/include/ilcplex` to facilitate your use of this format in your applications.

♦ `solution.xsl` stylesheet

♦ `solution.xsd` schema

> **Note**: Browsers may have specific conventions regarding accessing remote style sheets and XML files with extensions other than .xml. For example, your browser may not read the header of an SOL file with the extension .sol or may not consult the stylesheet designated there; consequently, your browser may display such a file as raw XML code, rather than the formatted display you expect. In such a situation, try copying your SOL file to a file with the extension .xml that your browser recognizes and processes correctly.

CPLEX®  can also read SOL files as an advanced start. SOL files contain basis statuses, if they are available, and solution values. The basis statuses can be used for advanced starts with simplex optimizers; the solution values can be used for a crossover from a barrier solution or as a MIP start from a mixed integer solution. A mixed integer solution may be from a conventional MIP optimization or from a member of the solution pool.

SOL files contain XML formatted information, as do MST files. Unlike MST files, at the default write level of detail, SOL files contain solution values for all variables rather than only the variables that define the integer feasible solution. SOL files are thus larger, take longer to input, and (in unusual cases of numerically difficult models) are less likely to provide a feasible starting point. Consequently, CPLEX®  generally recommends using MST files to restart the optimization instead of SOL files. However, if you already use SOL files for other purposes, you can also use them to provide a feasible starting point.

SOL files also carry an optional name attribute, useful when the problem has names. SOL files also include an index, corresponding to the constraint index or variable index of the problem.

The SOL header gives information about the status of the solution. For example, the optimization status appears as a string and the numeric value of the CPLEX®  symbolic constant.

The SOL quality gives information about the quality of the solution. For example, the maximum primal infeasibility, the values of the tolerance parameters in effect during the optimization, and other quality information appears in this part.

There are, of course, methods and routines for reading and writing SOL files.

♦ In Concert Technology, use these methods:

- In the C++ API, see the methods `IloCplex::readSolution` and `IloCplex::writeSolution`.

- In the Java API, see the methods `IloCplex.readSolution` and `IloCplex.writeSolution`.

- In the .NET API, see the methods `Cplex.ReadSolution` and `Cplex.WriteSolution`.

♦ In the Callable Library, use the routine `CPXreadcopysol` to read a SOL file and the routine `CPXsolwrite` to write SOL files.

Here is an example of a SOL file.

```
<?xml version = "1.0" standalone="yes"?>
<?xml-stylesheet href="https://www.ilog.com/products/cplex/xmlv1.0/solution.
xsl" type="text/xsl"?>
<CPLEXSolution version="1.1">
 <header
   problemName="../../../examples/data/mexample.mps"
   solutionName="incumbent"
   solutionIndex="-1"
   objectiveValue="-122.5"
   solutionTypeValue="3"
   solutionTypeString="primal"
   solutionStatusValue="101"
   solutionStatusString="integer optimal solution"
   MIPNodes="0"
   MIPIterations="3"/>
 <quality
   epInt="1e-05"
   epRHS="1e-06"
   maxIntInfeas="0"
   maxPrimalInfeas="0"
   maxX="40"
   maxSlack="2"/>
 <linearConstraints>
  <constraint name="c1" index="0" slack="0"/>
  <constraint name="c2" index="1" slack="2"/>
  <constraint name="c3" index="2" slack="0"/>
 </linearConstraints>
 <variables>
  <variable name="x1" index="0" value="40"/>
  <variable name="x2" index="1" value="10.5"/>
  <variable name="x3" index="2" value="19.5"/>
  <variable name="x4" index="3" value="3"/>
 </variables>
</CPLEXSolution>
```

# *FLT file format: filter files for the solution pool*

Describes FLT file format to support filters of the solution pool.

## In this section

**Overview of FLT**
Introduces FLT file format for specifying filters of the solution pool.

**Reading and writing filter files**
Cites methods and routines for reading and writing filter files.

**Syntax of a filter file**
Describes the syntax of a filter file.

**Diversity filters**
Describes diversity filters for the solution pool.

**Range filters**
Describes range filters for the solution pool.

# Overview of FLT

FLT denotes a file format for specifying filters (either diversity filters or range filters) associated with the solution pool of an application of CPLEX® . This section documents that format.

For more general information about the solution pool, see Solution pool: generating and keeping multiple solutions of the *CPLEX User's Manual*. For an introduction to these filters and their purpose, along with examples of their use, see also Diversity filters and Range filters of the *CPLEX User's Manual*.

# Reading and writing filter files

An existing filter file (such as one you create in your favorite text editor, or one you have saved from a previous session) can be added to your application and associated with the solution pool by one of these means:

♦ Concert Technology

  ● `readFilters` in the C++ API

  ● `readFilters(String)` in the Java API

  ● `Cplex.ReadFilters` in the .NET API

♦ `CPXreadcopysolnpoolfilters` routine of the Callable Library;

♦ `read` *filename* `flt` command of the Interactive Optimizer.

The diversity and range filters already associated with a solution pool can be saved in a formatted file, as explained in Filter files in the *CPLEX User's Manual*.

♦ In Concert Technology,

  ● in the C++ API, use the method `writeFilters`.

  ● in the Java API, use the method `writeFilters(String)`.

  ● in the .NET API, use the method `CPLEX.WriteFilters`.

♦ In the Callable Library, use the routine `CPXfltwrite`.

♦ `write` *filename* `flt` command of the Interactive Optimizer, where *filename* is the name of a file that the user supplies, and `flt` specifies the format of the file.

# Syntax of a filter file

A filter file may contain one or more diversity filters, one or more range filters, or a combination of both types of filter.

The filters in a file are associated with a particular model. The name of the model follows the keyword NAME.

In a formatted filter file, the strings -inf and inf may be used to denote "no limit" in each of these contexts:

♦ the lower bound of a diversity or range filter,

♦ the upper bound of a diversity or range filter.

Here is a sample filter file. This filter is suitable for use with the model in Example: simple facility location problem in the *CPLEX User's Manual*. An explanation of this file appears in *Diversity filters* and *Range filters*.

```
NAME location
DIVFILTER f1 2 inf
x1 1.0 1
x2 1.0 1
x3 1.0 0
x4 1.0 0
RNGFILTER f2 -inf 0
transport 1.0
fixed -1.0
ENDATA
```

# Diversity filters

A *diversity filter* allows you to control which solutions are generated and stored in the solution pool, according to their divergence from a reference solution. Only binary variables can be used to define a diversity filter.

The format of a diversity filter file lets you specify the names of the binary variables of interest, the weights to be assigned to those variables, and the reference values of those variables with which to compare all solutions. It also allows you to specify a lower bound and an upper bound on the divergence from the reference value(s).

The filter enforces the following constraint for a variable `x`:

```
lower bound <= diff(x) <= upper bound
```

This information is used to compute the *diversity measure*. The diversity measure is computed by summing the weighted absolute differences from the reference values, like this:

```
diff(x) = sum {weights[i] * |x[varind[i]] - refval[i]|};
```

The keyword DIVFILTER designates the beginning of a diversity filter in the file.

The name of the filter follows the keyword DIVFILTER on the same line. For example, in the sample in *Syntax of a filter file*, the name of the filter is `f1` and follows the keyword DIVFILTER.

The lower and upper bounds on the diversity function follow the name of the filter on the same line as the keyword DIVFILTER. In this example, the lower bound on diversity is 2, and the upper bound is infinity (that is, there is no limit).

Each successive line shows the name of a variable followed by the weight and the reference value for that variable.

In the example, equal weight (1.0) is given to the diversity of the four specified variables. The reference values are:

♦ 1 (one) for `x1` and `x2`

♦ 0 (zero) for `x3` and `x4`

# Range filters

A *range filter* adds a constraint over a linear expression, like this:

```
lower bound <= linear expression <= upper bound
```

where the linear expression is a sum of weights multiplied by the value of a variable. That is,

```
sum{weights[i]*x[varind[i]]}
```

Range filters can be defined by any type of variables (binary, integer, continuous, semi-integer, semi-continuous).

In the sample filter file, the range filter corresponds to this constraint that the transportation cost must be no more than the fixed cost:

```
1.0 * transport - 1.0 * fixed <= 0
```

In a formatted FLT file, a range filter is specified by the keyword RNGFILTER.

The name of the filter (in this example, f2 ) follows the keyword RNGFILTER on the same line.

The lower and upper bounds on the linear expression follow the name of the filter on the same line as the keyword RNGFILTER. In this example, the lower bound on the expression is negative infinity (that is, no lower limit) and the upper bound is 0 (zero).

Each successive line shows the name of the variable and its coefficient in the linear expression.

# CSV file format: comma separated values

CPLEX®  supports the file format known as CSV through XML facilities in Concert Technology. CSV is a file format consisting of lines of comma-separated values in ordinary ASCII text. Concert Technology provides classes adapted to reading data into your application from a CSV file. The constructors and methods of these classes are documented more fully in the *CPLEX C++ API Reference Manual* as the group `optim.concert.extensions`.

## IloCsvReader

An object of this class is capable of reading data from a CSV file and passing the data to your application. There are methods in this class for recognizing the first line of the file as a header, for indicating whether or not to cache the data, for counting columns, for counting lines, for accessing lines by number or by name, for designating special characters, for indicating separators, and so forth.

## IloCsvLine

An object of this class represents a line of a CSV file. The constructors and methods of this class enable you to designate special characters, such as a decimal point, separator, line ending, and so forth.

## IloCsvReader::Iterator

An object of this embedded class is an iterator capable of accessing data in a CSV file line by line. This iterator is useful, for example, in programming loops of your application, such as while-statements.

# XML file format: serialized models and solutions

Concert Technology for C++ users offers a suite of classes for serializing CPLEX® models (that is, instances of `IloModel` ) and solutions (that is, instances of `IloSolution`) through XML. The *CPLEX C++ API Reference Manual* documents the XML serialization API in the group `optim.concert.xml`. That group includes these classes:

♦ `IloXmlContext` allows you to serialize an instance of `IloModel` or `IloSolution`. This class offers methods for reading and writing a model, a solution, or both a model and a solution together. There are examples of how to use this class in the reference manual.

♦ `IloXmlInfo` offers methods that enable you to validate the XML serialization of elements, such as numeric arrays, integer arrays, variables, and other extractables from your model or solution.

♦ `IloXmlReader` creates a reader in an environment (that is, in an instance of `IloEnv`). This class offers methods to check runtime type information (RTTI), to recognize hierarchic relations between objects, and to access attributes of objects in your model or solution.

♦ `IloXmlWriter` creates a writer in an environment (that is, in an instance of `IloEnv`). This class offers methods to access elements and to convert their types as needed in order to serialize elements of your model or solution.

**Note**: There is a fundamental difference between writing an XML file of a model and writing an LP/MPS/SAV file of the same extracted model. If the model contains piecewise linear elements (PWL), or other nonlinear features, the XML file will represent the model as such. In contrast, the LP/MPS/SAV file will represent only the transformed model. That transformed model obscures these nonlinear features because of the automatic transformation that took place.

# *Index*