



MOSEK Optimizer API for Java

Release 9.3.21

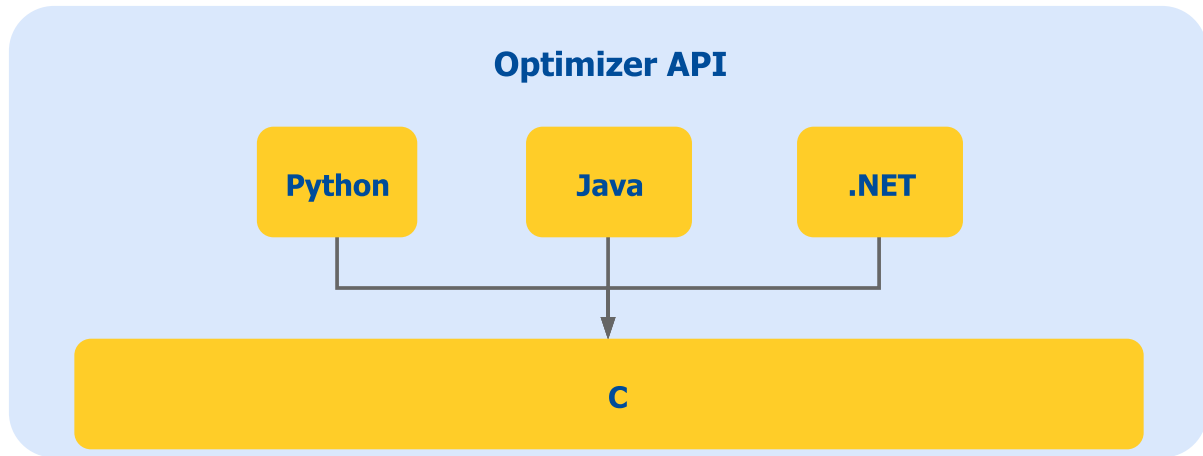
MOSEK ApS

08 August 2022

18.4	Constants	478
18.5	Response Codes	479
	Bibliography	482
	Symbol Index	483
	Index	498

1.1 Why the Optimizer API for Java?

The Optimizer API for Java provides an object-oriented interface to the **MOSEK** optimizers. This object oriented design is common to Java, Python and .NET and is based on a thin class-based interface to the native C optimizer API. The overhead introduced by this mapping is minimal.



The Optimizer API for Java can be used with any application running on the Oracle Java platform (and possibly other Java implementations). It consists of a single class library `mosek.jar` and a set of library files that must be available at runtime.

The Optimizer API for Java provides access to:

- Linear Optimization (LO)
- Conic Quadratic (Second-Order Cone) Optimization (CQO, SOCO)
- Power Cone Optimization
- Conic Exponential Optimization (CEO)
- Convex Quadratic and Quadratically Constrained Optimization (QO, QCQO)
- Semidefinite Optimization (SDO)
- Mixed-Integer Optimization (MIO)

as well as to additional functions for

- problem analysis,
- sensitivity analysis,
- infeasibility diagnostics,
- BLAS/LAPACK linear algebra routines.

Chapter 2

Contact Information

Phone	+45 7174 9373	
Website	mosek.com	
Email		
	sales@mosek.com	Sales, pricing, and licensing
	support@mosek.com	Technical support, questions and bug reports
	info@mosek.com	Everything else.
Mailing Address		
	MOSEK ApS	
	Fruebjergvej 3	
	Symbion Science Park, Box 16	
	2100 Copenhagen O	
	Denmark	

You can get in touch with **MOSEK** using popular social media as well:

Blogger	https://blog.mosek.com/
Google Group	https://groups.google.com/forum/#!forum/mosek
Twitter	https://twitter.com/mosektw
Linkedin	https://www.linkedin.com/company/mosek-aps
Youtube	https://www.youtube.com/channel/UCvIyectEVLp31NXeD5mIbEw

In particular **Twitter** is used for news, updates and release announcements.

Chapter 3

License Agreement

Before using the **MOSEK** software, please read the license agreement available in the distribution at <MSKHOME>/mosek/9.3/mosek-eula.pdf or on the **MOSEK** website <https://mosek.com/products/license-agreement>.

MOSEK uses some third-party open-source libraries. Their license details follows.

zlib

MOSEK includes the *zlib* library obtained from the [zlib website](#). The license agreement for *zlib* is shown in [Listing 3.1](#).

Listing 3.1: *zlib* license.

```
zlib.h -- interface of the 'zlib' general purpose compression library
version 1.2.7, May 2nd, 2012

Copyright (C) 1995-2012 Jean-loup Gailly and Mark Adler

This software is provided 'as-is', without any express or implied
warranty. In no event will the authors be held liable for any damages
arising from the use of this software.

Permission is granted to anyone to use this software for any purpose,
including commercial applications, and to alter it and redistribute it
freely, subject to the following restrictions:

1. The origin of this software must not be misrepresented; you must not
   claim that you wrote the original software. If you use this software
   in a product, an acknowledgment in the product documentation would be
   appreciated but is not required.
2. Altered source versions must be plainly marked as such, and must not be
   misrepresented as being the original software.
3. This notice may not be removed or altered from any source distribution.

Jean-loup Gailly          Mark Adler
jloup@gzip.org            madler@alumni.caltech.edu
```


Compiling examples using make

The example directory contains makefiles for use with GNU Make. To build the examples, open a prompt and change directory to the examples directory <EXDIR>. To compile all examples type

```
make -f Makefile
```

This will compile all the classes into a **jar** file. To run all the examples type

```
make test
```

Chapter 5

Design Overview

5.1 Modeling

Optimizer API for Java is an interface for specifying optimization problems directly in matrix form. It means that an optimization problem such as:

$$\begin{array}{ll}\text{minimize} & c^T x \\ \text{subject to} & Ax \leq b, \\ & x \in \mathcal{K}\end{array}$$

is specified by describing the matrix A , vectors b, c and a list of cones \mathcal{K} directly.

The main characteristics of this interface are:

- **Simplicity**: once the problem data is assembled in matrix form, it is straightforward to input it into the optimizer.
- **Exploiting sparsity**: data is entered in sparse format, enabling huge, sparse problems to be defined and solved efficiently.
- **Efficiency**: the Optimizer API incurs almost no overhead between the user's representation of the problem and **MOSEK**'s internal one.

Optimizer API for Java does not aid with modeling. It is the user's responsibility to express the problem in **MOSEK**'s standard form, introducing, if necessary, auxiliary variables and constraints. See [Sec. 12](#) for the precise formulations of problems **MOSEK** solves.

5.2 “Hello World!” in MOSEK

Here we present the most basic workflow pattern when using Optimizer API for Java.

Creating an environment and task

Every interaction with **MOSEK** using Optimizer API for Java begins by creating a **MOSEK environment**. It coordinates the access to **MOSEK** from the current process.

In most cases the user does not interact directly with the environment, except for creating optimization **tasks**, which contain actual problem specifications and where optimization takes place. An environment can host multiple tasks.


```

    };
    double blc[] = {30.0,
                    15.0,
                    -infinity
    };
    double buc[] = {30.0,
                    +infinity,
                    25.0
    };

    mosek.boundskey
    bkc[] = {mosek.boundskey.lo,
             mosek.boundskey.ra,
             mosek.boundskey.lo,
             mosek.boundskey.lo
    };
    double blx[] = {0.0,
                    0.0,
                    0.0,
                    0.0
    };
    double bux[] = {+infinity,
                    10.0,
                    +infinity,
                    +infinity
    };
    double[] xx = new double[numvar];

    try (mosek.Env env = new Env();
         mosek.Task task = new Task(env, 0, 0)) {
        // Directs the log task stream to the user specified
        // method task_msg_obj.stream
        task.set_Stream(
            mosek.streamtype.log,
            new mosek.Stream()
        );
        { public void stream(String msg) { System.out.print(msg); } });

        // Append 'numcon' empty constraints.
        // The constraints will initially have no bounds.
        task.appendcons(numcon);

        // Append 'numvar' variables.
        // The variables will initially be fixed at zero (x=0).
        task.appendvars(numvar);

        for (int j = 0; j < numvar; ++j) {
            // Set the linear term c_j in the objective.
            task.putcj(j, c[j]);

            // Set the bounds on variable j.
            // blx[j] <= x_j <= bux[j]
            task.putvarbound(j, bkc[j], blx[j], bux[j]);

            // Input column j of A
            task.putacol(j,
                        asub[j],
                        aval[j]);
            /* Variable (column) index.*/
            /* Row index of non-zeros in column j.*/
            /* Non-zero Values of column j. */
        }
    }

```

(continues on next page)

```

}

// Set the bounds on constraints.
// blc[i] <= constraint_i <= buc[i]
for (int i = 0; i < numcon; ++i)
    task.putconbound(i, bkc[i], blc[i], buc[i]);

// Input the objective sense (minimize/maximize)
task.putobjsense(mosek.objsense.maximize);

// Solve the problem
task.optimize();

// Print a summary containing information
// about the solution for debugging purposes
task.solutionsummary(mosek.streamtype.msg);

// Get status information about the solution
mosek.solsta solsta[] = new mosek.solsta[1];
task.getsolsta(mosek.soltype.bas, solsta);

switch (solsta[0]) {
    case optimal:
        task.getxx(mosek.soltype.bas, // Request the basic solution.
            xx);

        System.out.println("Optimal primal solution\n");
        for (int j = 0; j < numvar; ++j)
            System.out.println ("x[" + j + "]: " + xx[j]);
        break;
    case dual_infeas_cer:
    case prim_infeas_cer:
        System.out.println("Primal or dual infeasibility certificate found.\n");
        break;
    case unknown:
        System.out.println("Unknown solution status.\n");
        break;
    default:
        System.out.println("Other solution status");
        break;
}
}
catch (mosek.Exception e) {
    System.out.println ("An error/warning was encountered");
    System.out.println (e.toString());
    throw e;
}
}
}

```



```

    };
    double[] bux = {infinity,
                    infinity,
                    infinity
    };

    int[][] asub = { {0}, {0}, {0} };
    double[][] aval = { {1.0}, {1.0}, {1.0} };
    double[] xx = new double[numvar];

    try (Env env = new Env();
         Task task = new Task(env, 0, 0)) {
        // Directs the log task stream to the user specified
        // method task_msg_obj.stream
        task.set_Stream(
            mosek.streamtype.log,
            new mosek.Stream()
        );
        { public void stream(String msg) { System.out.print(msg); } });
        /* Give MOSEK an estimate of the size of the input data.
        This is done to increase the speed of inputting data.
        However, it is optional. */
        /* Append 'numcon' empty constraints.
        The constraints will initially have no bounds. */
        task.appendcons(numcon);

        /* Append 'numvar' variables.
        The variables will initially be fixed at zero (x=0). */
        task.appendvars(numvar);

        for (int j = 0; j < numvar; ++j) {
            /* Set the linear term c_j in the objective.*/
            task.putcj(j, c[j]);
            /* Set the bounds on variable j.
            blx[j] <= x_j <= bux[j] */
            task.putvarbound(j, bkc[j], blx[j], bux[j]);
            /* Input column j of A */
            task.putacol(j,
                        asub[j], /* Variable (column) index.*/
                        aval[j]); /* Row index of non-zeros in column j.*/
                                   /* Non-zero Values of column j. */
        }
        /* Set the bounds on constraints.
        for i=1, ..., numcon : blc[i] <= constraint i <= buc[i] */
        for (int i = 0; i < numcon; ++i)
            task.putconbound(i, bkc[i], blc[i], buc[i]);

        /*
        The lower triangular part of the Q
        matrix in the objective is specified.
        */

        int[] qsubi = {0, 1, 2, 2 };
        int[] qsubj = {0, 1, 0, 2 };
        double[] qval = {2.0, 0.2, -1.0, 2.0};

        /* Input the Q for the objective. */
    }

```



```

double[] blx = {0.0,
                0.0,
                0.0
                };
double[] bux = {infinity,
                infinity,
                infinity
                };

int[][] asub = { {0}, {0}, {0} };
double[][] aval = { {1.0}, {1.0}, {1.0} };

double[] xx = new double[numvar];

try (mosek.Env env = new mosek.Env();
    mosek.Task task = new mosek.Task(env, 0, 0)) {
    // Directs the log task stream to the user specified
    // method task_msg_obj.stream
    task.set_Stream(
        mosek.streamtype.log,
        new mosek.Stream()
    {
        public void stream(String msg) { System.out.print(msg); }
    });

    /* Give MOSEK an estimate of the size of the input data.
    This is done to increase the speed of inputting data.
    However, it is optional. */
    /* Append 'numcon' empty constraints.
    The constraints will initially have no bounds. */
    task.appendcons(numcon);

    /* Append 'numvar' variables.
    The variables will initially be fixed at zero (x=0). */
    task.appendvars(numvar);

    for (int j = 0; j < numvar; ++j) {
        /* Set the linear term c_j in the objective.*/
        task.putcj(j, c[j]);
        /* Set the bounds on variable j.
        blx[j] <= x_j <= bux[j] */
        task.putvarbound(j, bkc[j], blx[j], bux[j]);
        /* Input column j of A */
        task.putacol(j,
                     asub[j], /* Variable (column) index.*/
                     aval[j]); /* Row index of non-zeros in column j.*/
    }
    /* Set the bounds on constraints.
    for i=1, ..., numcon : blc[i] <= constraint i <= buc[i] */
    for (int i = 0; i < numcon; ++i)
        task.putconbound(i, bkc[i], blc[i], buc[i]);
    /*
    * The lower triangular part of the Q
    * matrix in the objective is specified.
    */

    int[] qosubi = { 0, 1, 2, 2 };

```

```

int[]   qosubj = { 0, 1, 0, 2 };
double[] qoval = { 2.0, 0.2, -1.0, 2.0 };

/* Input the Q for the objective. */

task.putqobj(qosubi, qosubj, qoval);

/*
 * The lower triangular part of the Q~0
 * matrix in the first constraint is specified.
 * This corresponds to adding the term
 * x0^2 - x1^2 - 0.1 x2^2 + 0.2 x0 x2
 */

int[]   qsubi = {0, 1, 2, 2 };
int[]   qsubj = {0, 1, 2, 0 };
double[] qval = { -2.0, -2.0, -0.2, 0.2};

/* put Q~0 in constraint with index 0. */

task.putqconk (0,
               qsubi,
               qsubj,
               qval);

task.putobjsense(mosek.objsense.minimize);

/* Solve the problem */

try {
    mosek.rescode termcode = task.optimize();
} catch (mosek.Warning e) {
    System.out.println (" Mosek warning:");
    System.out.println (e.toString ());
}

// Print a summary containing information
// about the solution for debugging purposes
task.solutionsummary(mosek.streamtype.msg);

mosek.solsta solsta[] = new mosek.solsta[1];
/* Get status information about the solution */
task.getsolsta(mosek.soltype.itr, solsta);

task.getxx(mosek.soltype.itr, // Interior solution.
           xx);
switch (solsta[0]) {
case optimal:
    System.out.println("Optimal primal solution\n");
    for (int j = 0; j < numvar; ++j)
        System.out.println ("x[" + j + "]: " + xx[j]);
    break;
case dual_infeas_cer:
case prim_infeas_cer:
    System.out.println("Primal or dual infeasibility.\n");
    break;
case unknown:

```



```

mosek.boundkey[] bkc    = { mosek.boundkey.fx };
double[] blc = { 1.0 };
double[] buc = { 1.0 };

mosek.boundkey[] bkc
= {mosek.boundkey.lo,
   mosek.boundkey.lo,
   mosek.boundkey.lo,
   mosek.boundkey.fr,
   mosek.boundkey.fr,
   mosek.boundkey.fr
};
double[] blx = { 0.0,
                 0.0,
                 0.0,
                 -infinity,
                 -infinity,
                 -infinity
};
double[] bux = { +infinity,
                 +infinity,
                 +infinity,
                 +infinity,
                 +infinity,
                 +infinity
};

double[] c    = { 0.0,
                 0.0,
                 0.0,
                 1.0,
                 1.0,
                 1.0
};

double[][] aval = {
    {1.0},
    {1.0},
    {2.0}
};
int[][] asub = {
    {0},
    {0},
    {0}
};

int[] csub = new int[3];
double[] xx = new double[numvar];
// create a new environment object
try (Env env = new Env();
     Task task = new Task(env, 0, 0)) {
    // Directs the log task stream to the user specified
    // method task_msg_obj.stream
    task.set_Stream(
        mosek.streamtype.log,

```

```

    new mosek.Stream()
{ public void stream(String msg) { System.out.print(msg); }});

/* Give MOSEK an estimate of the size of the input data.
This is done to increase the speed of inputting data.
However, it is optional. */
/* Append 'numcon' empty constraints.
The constraints will initially have no bounds. */
task.appendcons(numcon);

/* Append 'numvar' variables.
The variables will initially be fixed at zero (x=0). */
task.appendvars(numvar);

/* Optionally add a constant term to the objective. */
task.putcfix(0.0);
for (int j = 0; j < numvar; ++j) {
    /* Set the linear term c_j in the objective.*/
    task.putcj(j, c[j]);
    /* Set the bounds on variable j.
       blx[j] <= x_j <= bux[j] */
    task.putvarbound(j, bkg[j], blx[j], bux[j]);
}

for (int j = 0; j < aval.length; ++j)
    /* Input column j of A */
    task.putacol(j,                /* Variable (column) index.*/
                 asub[j],          /* Row index of non-zeros in column j.*/
                 aval[j]);         /* Non-zero Values of column j. */

/* Set the bounds on constraints.
for i=1, ...,numcon : blc[i] <= constraint i <= buc[i] */
for (int i = 0; i < numcon; ++i)
    task.putconbound(i, bkg[i], blc[i], buc[i]);

csub[0] = 3;
csub[1] = 0;
csub[2] = 1;
task.appendcone(mosek.conetype.quad,
                0.0, /* For future use only, can be set to 0.0 */
                csub);

csub[0] = 4;
csub[1] = 5;
csub[2] = 2;
task.appendcone(mosek.conetype.rquad, 0.0, csub);

task.putobjsense(mosek.objsense.minimize);

System.out.println ("optimize");
/* Solve the problem */
mosek.rescode r = task.optimize();
System.out.println (" Mosek warning:" + r.toString());
// Print a summary containing information
// about the solution for debugging purposes
task.solutionsummary(mosek.streamtype.msg);

```



```
[ 0.06389298  0.78308564  2.30604283 ]
```

Source code

Listing 6.5: Source code solving problem (6.7).

```
package com.mosek.example;

import mosek.*;

public class pow1 {
    static final int numcon = 1;
    static final int numvar = 6;

    public static void main (String[] args) throws java.lang.Exception {
        // Since the value infinity is never used, we define
        // 'infinity' symbolic purposes only
        double infinity = 0;

        mosek.boundkey[] bkg = new mosek.boundkey[numvar];
        double[] blx      = new double[numvar];
        double[] bux      = new double[numvar];

        double[] val      = { 1.0, 1.0, -1.0 };
        int[]      sub      = { 3, 4, 0 };

        double[] aval     = { 1.0, 1.0, 0.5 };
        int[]      asub     = { 0, 1, 2 };

        int i;
        double[] xx = new double[numvar];

        // create a new environment object
        try (Env env = new Env();
             Task task = new Task(env, 0, 0)) {
            // Directs the log task stream to the user specified
            // method task_msg_obj.stream
            task.set_Stream(
                mosek.streamtype.log,
                new mosek.Stream()
            {
                public void stream(String msg) { System.out.print(msg); }
            });

            /* Append 'numcon' empty constraints.
               The constraints will initially have no bounds. */
            task.appendcons(numcon);

            /* Append 'numvar' variables.
               The variables will initially be fixed at zero (x=0). */
            task.appendvars(numvar);

            /* Define the linear part of the problem */
            task.putclist(sub, val);
            task.putarow(0, asub, aval);
            task.putconbound(0, mosek.boundkey.fx, 2.0, 2.0);
            for(i=0;i<5;i++) {
                bkg[i] = mosek.boundkey.fr;
            }
        }
    }
}
```

(continues on next page)

```

        blx[i] = -infinity;
        bux[i] = infinity;
    }
    bkg[5] = mosek.boundkey.fx;
    blx[5] = bux[5] = 1.0;
    task.putvarboundslice(0, numvar, bkg, blx, bux);

    /* Add a conic constraint */
    int[][] csub = { {0, 1, 3}, {2, 5, 4}};
    task.appendcone(mosek.conetype.ppow, 0.2, csub[0]);
    task.appendcone(mosek.conetype.ppow, 0.4, csub[1]);

    task.putobjsense(mosek.objsense.maximize);

    System.out.println ("optimize");
    /* Solve the problem */
    mosek.rescode r = task.optimize();
    System.out.println (" Mosek warning:" + r.toString());
    // Print a summary containing information
    // about the solution for debugging purposes
    task.solutionsummary(mosek.streamtype.msg);

    mosek.solsta solsta[] = new mosek.solsta[1];

    /* Get status information about the solution */
    task.getsolsta(mosek.soltype.itr, solsta);

    task.getxx(mosek.soltype.itr, // Interior solution.
               xx);

    switch (solsta[0]) {
        case optimal:
            System.out.println("Optimal primal solution\n");
            for (int j = 0; j < 3; ++j)
                System.out.println ("x[" + j + "]: " + xx[j]);
            break;
        case dual_infeas_cer:
        case prim_infeas_cer:
            System.out.println("Primal or dual infeasibility.\n");
            break;
        case unknown:
            System.out.println("Unknown solution status.\n");
            break;
        default:
            System.out.println("Other solution status");
            break;
    }
}
catch (mosek.Exception e) {
    System.out.println ("An error/warning was encountered");
    System.out.println (e.toString());
    throw e;
}
}
}

```



```

        0.0
    };

    double[] a    = { 1.0,
                      1.0,
                      1.0
    };

    int[] asub    = {0, 1, 2};
    int[] csub    = new int[numvar];
    double[] xx   = new double[numvar];

    // create a new environment object
    try (Env env = new Env();
         Task task = new Task(env, 0, 0)) {
        // Directs the log task stream to the user specified
        // method task_msg_obj.stream
        task.set_Stream(
            mosek.streamtype.log,
            new mosek.Stream()
        {
            public void stream(String msg) { System.out.print(msg); }
        });

        /* Append 'numcon' empty constraints.
           The constraints will initially have no bounds. */
        task.appendcons(numcon);

        /* Append 'numvar' variables.
           The variables will initially be fixed at zero (x=0). */
        task.appendvars(numvar);

        /* Define the linear part of the problem */
        task.putcslice(0, numvar, c);
        task.putarow(0, asub, a);
        task.putconbound(0, bkc, blc, buc);
        task.putvarboundslice(0, numvar, bkx, blx, bux);

        /* Add a conic constraint */
        csub[0] = 0;
        csub[1] = 1;
        csub[2] = 2;
        task.appendcone(mosek.conetype.pexp,
                       0.0, /* For future use only, can be set to 0.0 */
                       csub);

        task.putobjsense(mosek.objsense.minimize);

        System.out.println ("optimize");
        /* Solve the problem */
        mosek.rescode r = task.optimize();
        System.out.println (" Mosek warning:" + r.toString());
        // Print a summary containing information
        // about the solution for debugging purposes
        task.solutionsummary(mosek.streamtype.msg);

        mosek.solsta solsta[] = new mosek.solsta[1];

        /* Get status information about the solution */

```



```

{ public void stream(String msg) { System.out.print(msg); } });

/* Append 'NUMCON' empty constraints.
   The constraints will initially have no bounds. */
task.appendcons(numcon);

/* Append 'NUMVAR' variables.
   The variables will initially be fixed at zero (x=0). */
task.appendvars(numvar);

/* Append 'NUMBARVAR' semidefinite variables. */
task.appendbarvars(dimbarvar);

/* Optionally add a constant term to the objective. */
task.putcfix(0.0);

/* Set the linear term c_j in the objective.*/
task.putcj(0, 1.0);

for (int j = 0; j < numvar; ++j)
    task.putvarbound(j, mosek.boundkey.fr, -0.0, 0.0);

/* Set the linear term barc_j in the objective.*/
{
    long[] idx = new long[1];
    double[] falpha = { 1.0 };
    idx[0] = task.appendsparsesymmat(dimbarvar[0],
                                    barc_i,
                                    barc_j,
                                    barc_v);

    task.putbarcj(0, idx, falpha);
}

/* Set the bounds on constraints.
   for i=1, ..., numcon : blc[i] <= constraint i <= buc[i] */

for (int i = 0; i < numcon; ++i)
    task.putconbound(i,          /* Index of constraint.*/
                     bkc[i],     /* Bound key.*/
                     blc[i],     /* Numerical value of lower bound.*/
                     buc[i]);    /* Numerical value of upper bound.*/

/* Input A row by row */
for (int i = 0; i < numcon; ++i)
    task.putarow(i,
                 asub[i],
                 aval[i]);

/* Append the conic quadratic cone */
task.appendcone(mosek.conetype.quad,
               0.0,
               conesub);

/* Add the first row of barA */
{
    long[] idx = new long[1];

```

```

double[] falpha = {1.0};
task.appendsparsesymmat(dimbarvar[0],
                        bara_i[0],
                        bara_j[0],
                        bara_v[0],
                        idx);

task.putbaraij(0, 0, idx, falpha);
}

{
    long[] idx = new long[1];
    double[] falpha = {1.0};
    /* Add the second row of barA */
    task.appendsparsesymmat(dimbarvar[0],
                            bara_i[1],
                            bara_j[1],
                            bara_v[1],
                            idx);

    task.putbaraij(1, 0, idx, falpha);
}

/* Run optimizer */
task.optimize();

/* Print a summary containing information
   about the solution for debugging purposes*/
task.solutionsummary (mosek.streamtype.msg);

mosek.solsta[] solsta = new mosek.solsta[1];
task.getsolsta (mosek.soltype.itr, solsta);

switch (solsta[0]) {
    case optimal:
        double[] xx = new double[numvar];
        double[] barx = new double[lenbarvar[0]];

        task.getxx(mosek.soltype.itr, xx);
        task.getbarxj(mosek.soltype.itr,    /* Request the interior solution. */
                     0,
                     barx);
        System.out.println("Optimal primal solution");
        for (int i = 0; i < numvar; ++i)
            System.out.println("x[" + i + "] : " + xx[i]);

        for (int i = 0; i < lenbarvar[0]; ++i)
            System.out.println("barx[" + i + "]: " + barx[i]);
        break;
    case dual_infeas_cer:
    case prim_infeas_cer:
        System.out.println("Primal or dual infeasibility certificate found.");
        break;
    case unknown:
        System.out.println("The status of the solution could not be determined.");
        break;
}

```

(continues on next page)


```

/* Append numbarvar semidefinite variables. */
task.appendbarvars(dimbarvar);

/* Set objective (6 nonzeros).*/
task.putbarblocktriplet(6, Cj, Ck, Cl, Cv);

/* Set the equality constraint (6 nonzeros).*/
task.putbarblocktriplet(6, Ai, Aj, Ak, Al, Av);

/* Set the inequality constraint (1 nonzero).*/
task.putbarblocktriplet(1, A2i, A2j, A2k, A2l, A2v);

/* Set constraint bounds */
task.putconboundslice(0, 2, bkc, blc, buc);

/* Run optimizer */
task.optimize();
task.solutionsummary(mosek.streamtype.msg);

mosek.solsta[] solsta = new mosek.solsta[1];
task.getsolsta (mosek.soltype.itr, solsta);

switch (solsta[0]) {
    case optimal:

        /* Retrieve the soution for all symmetric variables */
        System.out.println("Solution (lower triangular part vectorized):");
        for(int i = 0; i < numbarvar; i++) {
            int dim = dimbarvar[i] * (dimbarvar[i] + 1) / 2;
            double[] barx = new double[dim];

            task.getbarxj(mosek.soltype.itr, i, barx);

            System.out.print("X" + (i+1) + ": ");
            for (int j = 0; j < dim; ++j)
                System.out.print(barx[j] + " ");
            System.out.println();
        }

        break;
    case dual_infeas_cer:
    case prim_infeas_cer:
        System.out.println("Primal or dual infeasibility certificate found.");
        break;
    case unknown:
        System.out.println("The status of the solution could not be determined.");
        break;
    default:
        System.out.println("Other solution status.");
        break;
}
}
}
}

```



```

int[][] asub    = { {0, 1}, {0, 1} };
double[][] aval = { {50.0, 3.0}, {31.0, -2.0} };

int[] ptrb = { 0, 2 };
int[] ptre = { 2, 4 };

double[] xx = new double[numvar];

try (Env env = new Env();
     Task task = new Task(env, 0, 0)) {
    // Directs the log task stream to the user specified
    // method task_msg_obj.stream
    task.set_Stream(
        mosek.streamtype.log,
        new mosek.Stream()
    {
        public void stream(String msg) { System.out.print(msg); });
    task.set_ItgSolutionCallback(
        new mosek.ItgSolutionCallback() {
            public void callback(double[] xx) {
                System.out.print("New integer solution: ");
                for (double v : xx) System.out.print(" " + v + " ");
                System.out.println("");
            }
        });
    /* Append 'numcon' empty constraints.
       The constraints will initially have no bounds. */
    task.appendcons(numcon);

    /* Append 'numvar' variables.
       The variables will initially be fixed at zero (x=0). */
    task.appendvars(numvar);

    for (int j = 0; j < numvar; ++j) {
        /* Set the linear term c_j in the objective.*/
        task.putcj(j, c[j]);
        /* Set the bounds on variable j.
           blx[j] <= x_j <= bux[j] */
        task.putvarbound(j, bkc[j], blx[j], bux[j]);
        /* Input column j of A */
        task.putacol(j,
                     asub[j], /* Variable (column) index.*/
                     aval[j]); /* Row index of non-zeros in column j.*/
    }
    /* Set the bounds on constraints.
       for i=1, ..., numcon : blc[i] <= constraint i <= buc[i] */
    for (int i = 0; i < numcon; ++i)
        task.putconbound(i, bkc[i], blc[i], buc[i]);

    /* Specify integer variables. */
    for (int j = 0; j < numvar; ++j)
        task.putvartype(j, mosek.variabletype.type_int);

    /* Set max solution time */
    task.putdparam(mosek.dparam.mio_max_time, 60.0);
}

```

```

/* A maximization problem */
task.putobjsense(mosek.objsense.maximize);
/* Solve the problem */
try {
    task.optimize();
} catch (mosek.Warning e) {
    System.out.println (" Mosek warning:");
    System.out.println (e.toString ());
}

// Print a summary containing information
// about the solution for debugging purposes
task.solutionsummary(mosek.streamtype.msg);
task.getxx(mosek.soltype.itg, // Integer solution.
           xx);
mosek.solsta solsta[] = new mosek.solsta[1];
/* Get status information about the solution */
task.getsolsta(mosek.soltype.itg, solsta);

switch (solsta[0]) {
    case integer_optimal:
        System.out.println("Optimal solution\n");
        for (int j = 0; j < numvar; ++j)
            System.out.println ("x[" + j + "]: " + xx[j]);
        break;
    case prim_feas:
        System.out.println("Feasible solution\n");
        for (int j = 0; j < numvar; ++j)
            System.out.println ("x[" + j + "]: " + xx[j]);
        break;

    case unknown:
        mosek.prosta prosta[] = new mosek.prosta[1];
        task.getprosta(mosek.soltype.itg, prosta);
        switch (prosta[0]) {
            case prim_infeas_or_unbounded:
                System.out.println("Problem status Infeasible or unbounded");
                break;
            case prim_infeas:
                System.out.println("Problem status Infeasible.");
                break;
            case unknown:
                System.out.println("Problem status unknown.");
                break;
            default:
                System.out.println("Other problem status.");
                break;
        }
        break;
    default:
        System.out.println("Other solution status");
        break;
}
}
catch (mosek.Exception e) {

```


(continued from previous page)

```
for(int i = 0; i < numExp; i++) sub[i] = expStart + 3*i + 2;
task.putvarboundlistconst(sub, boundkey.fr, -inf, inf);

// t_i = 1
for(int i = 0; i < numExp; i++) sub[i] = expStart + 3*i + 1;
task.putvarboundlistconst(sub, boundkey.fx, 1, 1);

// Every triple is in an exponential cone
conetype[] ct = new conetype[numExp];
int[] len = new int[numExp];
for(int i = 0; i < numExp; i++)
    { ct[i] = conetype.pexp; val[i] = 0.0; len[i] = 3; }
task.appendconesseq(ct, val, len, expStart);

// Solve and map to original h, w, d
task.optimize();
double[] xyz = new double[numvar];
double[] hwd = new double[numvar];
task.getxxslice(soltype.itr, 0, numvar, xyz);
for(int i = 0; i < numvar; i++) hwd[i] = Math.exp(xyz[i]);
return hwd;
}
}
```

Given sample data we obtain the solution h, w, d as follows:

Listing 6.15: Sample data for problem (6.19).

```
public static void main(String[] args)
{
    double Aw    = 200.0;
    double Af    = 50.0;
    double alpha = 2.0;
    double beta  = 10.0;
    double gamma = 2.0;
    double delta = 10.0;

    double[] hwd = max_volume_box(Aw, Af, alpha, beta, gamma, delta);

    System.out.format("h=%.4f w=%.4f d=%.4f\n", hwd[0], hwd[1], hwd[2]);
}
```

6.9 Library of basic functions

This section contains a library of small models of basic functions frequently appearing in optimization models. It is essentially an implementation of the mathematical models from the **MOSEK Modeling Cookbook** using Optimizer API for Java. These short code snippets can be seen as illustrative examples, can be copy-pasted to other code, and can even be directly called when assembling optimization models as we show in Sec. 6.9.6 (although this may be more suitable for prototyping; also note that additional variables and constraints will be introduced and there is no error checking).

6.9.1 Variable and constraint management

Append variables

Adds a number of new variables. Returns the index of the first variable in the sequence.

Listing 6.16: New variables.

```
public static int msk_newvar(Task task, int num) { // free
    int v = task.getnumvar();
    task.appendvars(num);
    for(int i=0; i<num; i++)
        task.putvarbound(v+i, boundkey.fr, -inf, inf);
    return v;
}
public static int msk_newvar_fx(Task task, int num, double val) { // fixed
    int v = task.getnumvar();
    task.appendvars(num);
    for(int i=0; i<num; i++)
        task.putvarbound(v+i, boundkey.fx, val, val);
    return v;
}
public static int msk_newvar_bin(Task task, int num) { // binary
    int v = task.getnumvar();
    task.appendvars(num);
    for(int i=0; i<num; i++) {
        task.putvarbound(v+i, boundkey.ra, 0.0, 1.0);
        task.putvartype(v+i, variabletype.type_int);
    }
    return v;
}
```

Variable duplication

Declares equality of two variables, or returns an index of a new duplicate of an existing variable.

p-norm

$$t \geq (\sum_i |x_i|^p)^{1/p}, p > 1$$

Listing 6.25: p -norm.

```
// t >= \|x\|_p (where p>1), x is a list of variables
public static void msk_pnorm(Task task, int t, int[] x, double p) {
    int n = x.length;
    int r = msk_newvar(task, n);
    for(int i=0; i<n; i++)
        task.appendcone(conetype.ppow, 1.0-1.0/p, new int[]{t, r+i, x[i]});
    int c = msk_newcon(task, 1);
    for(int i=0; i<n; i++)
        task.putaij(c, r+i, -1.0);
    task.putaij(c, t, 1.0);
    task.putconbound(c, boundkey.fx, 0.0, 0.0);
}
```

Geometric mean

$$t \leq (x_1 \cdots x_n)^{1/n}, x_i > 0$$

Listing 6.26: Geometric mean.

```
// |t| <= (x_1...x_n)^(1/n), x_i>=0, x is a list of variables of length >= 1
public static void msk_geo_mean(Task task, int t, int[] x) {
    int n = x.length;
    if (n==1) msk_abs(task, x[0], t);
    else {
        int t2 = msk_newvar(task, 1);
        task.appendcone(conetype.ppow, 1.0-1.0/n, new int[]{t2, x[n-1], t});
        msk_geo_mean(task, msk_dup(task, t2), Arrays.copyOfRange(x, 0, n-1));
    }
}
```

6.9.4 Exponentials and logarithms

log

$$t \leq \log x, x > 0$$

Listing 6.27: Logarithm.

```
// t <= log(x), x>=0
public static void msk_log(Task task, int t, int x) {
    task.appendcone(conetype.pexp, 0.0, new int[]{x, msk_newvar_fx(task, 1, 1.0), t});
}
```

exp

$$t \geq e^x$$

Listing 6.28: Exponential.

```
// t >= exp(x)
public static void msk_exp(Task task, int t, int x) {
    task.appendcone(conetype.pexp, 0.0, new int[]{t, msk_newvar_fx(task, 1, 1.0), x});
}
```

Entropy

$$t \geq x \log x, x > 0$$

Listing 6.29: Entropy.

```
// t >= x * log(x), x>=0
public static void msk_ent(Task task, int t, int x) {
    int v = msk_newvar(task, 1);
    int c = msk_newcon(task, 1);
    task.putaij(c, v, 1.0);
    task.putaij(c, t, 1.0);
    task.putconbound(c, boundkey.fx, 0.0, 0.0);
    task.appendcone(conetype.pexp, 0.0, new int[]{msk_newvar_fx(task, 1, 1.0), x, v});
}
```

Relative entropy

$$t \geq x \log x/y, x, y > 0$$

Listing 6.30: Relative entropy.

```
// t >= x * log(x/y), x,y>=0
public static void msk_relent(Task task, int t, int x, int y) {
    int v = msk_newvar(task, 1);
    int c = msk_newcon(task, 1);
    task.putaij(c, v, 1.0);
    task.putaij(c, t, 1.0);
    task.putconbound(c, boundkey.fx, 0.0, 0.0);
    task.appendcone(conetype.pexp, 0.0, new int[]{y, x, v});
}
```

Log-sum-exp

$$\log \sum_i e^{x_i} \leq t$$

Listing 6.31: Log-sum-exp.

```
// log( sum_i(exp(x_i)) ) <= t, where x is a list of variables
public static void msk_logsumexp(Task task, int t, int[] x) {
    int n = x.length;
    int u = msk_newvar(task, n);
    int z = msk_newvar(task, n);
    for(int i=0; i<n; i++) msk_exp(task, u+i, z+i);
    int c = msk_newcon(task, n);
    for(int i=0; i<n; i++) {
        task.putarow(c+i, new int[]{x[i], t, z+i}, new double[]{1.0, -1.0, -1.0});
        task.putconbound(c+i, boundkey.fx, 0.0, 0.0);
    }
    int s = msk_newcon(task, 1);
    for(int i=0; i<n; i++) task.putaij(s, u+i, 1.0);
    task.putconbound(s, boundkey.up, -inf, 1.0);
}
```

6.9.5 Integer Modeling

Semicontinuous variable

$$x \in \{0\} \cup [a, b], b > a > 0$$


```

// Perform optimization.
rescode trm = task.optimize();
task.solutionsummary(streamtype.log);

// Handle solution status. We expect Optimal
solsta solsta = task.getsolsta(soltype.itr);

switch ( solsta ) {
  case optimal:
    // Fetch and print the solution
    System.out.println("An optimal interior point solution is located.");
    int numvar = task.getnumvar();
    double[] xx = new double[numvar];
    task.getxx(soltype.itr, xx);
    for(int i = 0; i < numvar; i++)
      System.out.println("x[" + i + "] = " + xx[i]);
    break;

  case dual_infeas_cer:
    System.out.println("Dual infeasibility certificate found.");
    break;

  case prim_infeas_cer:
    System.out.println("Primal infeasibility certificate found.");
    break;

  case unknown:
    // The solutions status is unknown. The termination code
    // indicates why the optimizer terminated prematurely.
    System.out.println("The solution status is unknown.");
    Env.getcodedesc(trm, symname, desc);
    System.out.printf("  Termination code: %s %s\n", symname, desc);
    break;

  default:
    System.out.println("Unexpected solution status " + solsta + "\n");
    break;
}
}
catch (mosek.Error e) {
  System.out.println("Unexpected error (" + e.code + ") " + e.msg);
}
}
}

```



```

task.set_Stream (mosek.streamtype.log,
new mosek.Stream() {
    public void stream(String msg) { System.out.print(msg); }
});

long start = System.currentTimeMillis();

System.out.println("Starting polling loop...");

int i = 0;

while ( true ) {

    Thread.sleep(100);

    System.out.printf("poll %d...\n", i);

    rescode trm[] = new rescode[1];
    rescode resp[] = new rescode[1];

    boolean respavailable = task.asyncpoll( host,
                                           port,
                                           token,
                                           resp,
                                           trm);

    System.out.println("polling done");

    if (respavailable) {
        System.out.println("solution available!");

        task.asyncgetresult(host,
                           port,
                           token,
                           resp,
                           trm);

        task.solutionsummary (mosek.streamtype.log);
        break;
    }

    i++;

    if (i == numpolls) {
        System.out.println("max num polls reached, stopping host.");
        task.asyncstop (host, port, token);
        break;
    }

}

} catch (java.lang.Exception e) {
    System.out.println("Something unexpected happend...");
    throw e;
}
}

```

(continued from previous page)

```
}  
}  
}
```


Table 8.1: List of commands of the MOSEK Python Console.

Command	Description
help [command]	Print list of commands or info about a specific command
log filename	Save the session to a file
intro	Print MOSEK splashscreen
testlic	Test the license system
read filename	Load problem from file
reread	Reload last problem file
solve [options]	Solve current problem
write filename	Write current problem to file
param [name [value]]	Set a parameter or get parameter values
paramdef	Set all parameters to default values
paramdiff	Show parameters with non-default values
info [name]	Get an information item
anapro	Analyze problem data
hist	Plot a histogram of problem data
histsol	Plot a histogram of the solutions
spy	Plot the sparsity pattern of the A matrix
truncate epsilon	Truncate small coefficients down to 0
resobj [fac]	Rescale objective by a factor
anasol	Analyze solutions
removeitg	Remove integrality constraints
removecones	Remove all cones and leave just the linear part
infsub	Replace current problem with its infeasible subproblem
writesol basename	Write solution(s) to file(s) with given basename
del_sol	Remove all solutions from the task
optserver [url]	Use an OptServer to optimize
exit	Leave


```

double[] c      = {1.0, 1.0};

int[]   ptrb = {0, 2};
int[]   ptre = {2 , 4};

int[]   asub = {0, 1,
                0, 1
                };

double[] aval = {1.0, 1.0,
                 2.0, 1.0
                 };

mosek.boundkey[] bkc = {
    mosek.boundkey.up,
    mosek.boundkey.up
};
double[]  blc  = { -infinity,
                  -infinity
                  };

double[]  buc  = {2.0,
                  6.0
                  };

mosek.boundkey[] bkx = {
    mosek.boundkey.lo,
    mosek.boundkey.lo
};
double[]  blx  = {0.0,
                  0.0
                  };
double[]  bux  = { +infinity,
                  +infinity
                  };

int    numvar = 2;
int    numcon = 2;

double[] w1 = {2.0, 6.0};
double[] w2 = {1.0, 0.0};

try (Env env = new Env();
     Task task = new Task(env, 0, 0)) {
    task.inputdata(numcon, numvar,
                  c,
                  0.0,
                  ptrb,
                  ptre,
                  asub,
                  aval,
                  bkc,
                  blc,
                  buc,

```

```

        bkx,
        blx,
        bux);
task.putobjsense(mosek.objsense.maximize);

System.out.println("optimize");
try {
    task.optimize();
} catch (mosek.Warning e) {
    System.out.println("Mosek warning:");
    System.out.println(e.toString());
}

int[] basis = new int[numcon];
task.initbasissolve(basis);

//List basis variables corresponding to columns of B
int[] varsub = {0, 1};
for (int i = 0; i < numcon; i++) {
    System.out.println("Basis i:" + i + " Basis:" + basis[i]);
    if (basis[varsub[i]] < numcon) {
        System.out.println("Basis variable no " + i + " is xc" +
            basis[i]);
    } else {
        int index = basis[i] - numcon;
        System.out.println("Basis variable no " + i + " is x" +
            index);
    }
}

// solve Bx = w1
// varsub contains index of non-zeros in b.
// On return b contains the solution x and
// varsub the index of the non-zeros in x.

int[] nz = new int[1];
nz[0] = 2;

task.solvewithbasis(0, nz, varsub, w1);
System.out.println("nz =" + nz[0]);
System.out.println("\nSolution to Bx = w1:\n");

for (int i = 0; i < nz[0]; i++) {
    if (basis[varsub[i]] < numcon) {
        System.out.println("xc" + basis[varsub[i]] + "=" + w1[varsub[i]]);
    } else {
        int index = basis[varsub[i]] - numcon;
        System.out.println("x" + index + " = " + w1[varsub[i]]);
    }
}

// Solve B^Tx = w2
nz[0] = 2;
varsub[0] = 0;
varsub[1] = 1;

```



```

for (int i = 0 ; i < numvar ; ++i)
    task.putvarbound(
        i,
        mosek.boundkey.fr,
        -infinity,
        infinity);

/* Define a basic solution by specifying
   status keys for variables & constraints. */
task.deletesolution(mosek.soltype.bas);

task.putskcslice(mosek.soltype.bas, 0, numvar, skc);
task.putskxslice(mosek.soltype.bas, 0, numvar, skx);

task.initbasissolve(basis);
}

public static void main (String[] argv) {
    int numcon = 2;
    int numvar = 2;

    double[][] aval = {
        { -1.0 },
        { 1.0, 1.0 }
    };
    int[][] asub = {
        { 1 },
        { 0, 1 }
    };
    int[] ptrb = new int[] {0, 1};
    int[] ptre = new int[] {1, 3};

    int[] bsub = new int[numvar];
    double[] b = new double[numvar];
    int[] basis = new int[numvar];

    try (Env env = new Env();
        Task task = new Task(env, 0, 0)) {
        // Directs the log task stream to the user specified
        // method task_msg_obj.streamCB
        task.set_Stream(
            mosek.streamtype.log,
            new mosek.Stream()
            { public void stream(String msg) { System.out.print(msg); } });

        /* Put A matrix and factor A.
           Call this function only once for a given task. */

        setup(
            task,
            aval,
            asub,
            ptrb,
            ptre,
            numvar,

```

(continues on next page)

```

        basis
    );

    /* now solve rhs */
    b[0] = 1;
    b[1] = -2;
    bsub[0] = 0;
    bsub[1] = 1;
    int[] nz_ = { 2 };
    task.solvewithbasis(0, nz_, bsub, b);
    int nz = nz_[0];
    System.out.println("\nSolution to Bx = b:\n");

    /* Print solution and show correspondents
       to original variables in the problem */
    for (int i = 0; i < nz; ++i) {
        if (basis[bsub[i]] < numcon)
            System.out.println ("This should never happen");
        else
            System.out.println("x" + (basis[bsub[i]] - numcon) + " = " + b[bsub[i]]);
    }

    b[0] = 7;
    bsub[0] = 0;
    nz_[0] = 1;
    task.solvewithbasis(0, nz_, bsub, b);
    nz = nz_[0];

    System.out.println ("\nSolution to Bx = b:\n");
    /* Print solution and show correspondents
       to original variables in the problem */
    for (int i = 0; i < nz; ++i) {
        if (basis[bsub[i]] < numcon)
            System.out.println("This should never happen");
        else
            System.out.println("x" + (basis[bsub[i]] - numcon) + " = " + b[bsub[i]] );
    }
}
}
}

```

The most important step in the above example is the definition of the basic solution, where we define the status key for each variable. The actual values of the variables are not important and can be selected arbitrarily, so we set them to zero. All variables corresponding to columns in the linear system we want to solve are set to basic and the slack variables for the constraints, which are all non-basic, are set to their bound.

The program produces the output:

Solution to Bx = b:

x1 = 1
x0 = 3

Solution to Bx = b:

x1 = 7
x0 = 7


```

[/variables]

[objective minimize]
  - 2.2e+01 x0000_x0 - 1.45e+01 x0001_x1 + 1.2e+01 x0002_x2 + x0003
  + 1e+00
[/objective]

[constraints]
  [con c0000] 3.605551275463989e+00 x0000_x0 - 5.547001962252291e-01 x0002_x2 + 3.
  ↪328201177351375e+00 x0001_x1 - x0006 = 0e+00 [/con]
  [con c0001] 3.419401657060442e+00 x0002_x2 + 2.294598480395823e+00 x0001_x1 -
  ↪x0007 = 0e+00 [/con]
  [con c0002] 8.111071056538127e-01 x0001_x1 - x0008 = 0e+00 [/con]
  [con c0003] - x0003 + x0004 = 0e+00 [/con]
[/constraints]

[bounds]
  [b] -1e+00      <= x0000_x0,x0001_x1,x0002_x2 <= 1e+00 [/b]
  [b]              x0003,x0004 free [/b]
  [b]              x0005 = 1e+00 [/b]
  [b]              x0006,x0007,x0008 free [/b]
  [cone rquad k0000] x0004, x0005, x0006, x0007, x0008 [/cone]
[/bounds]

```

We can clearly see that constraints c0000, c0001 and c0002 represent the original linear constraints as in (9.11), while c0003 corresponds to (9.10). The cone roots are x0005 and x0004.

Chapter 10

Technical guidelines

This section contains some more in-depth technical guidelines for Optimizer API for Java, not strictly necessary for basic use of **MOSEK**.

10.1 Memory management and garbage collection

Users who experience memory leaks, especially:

- memory usage not decreasing after the solver terminates,
- memory usage increasing when solving a sequence of problems,

should make sure that the *Task* objects are properly garbage collected. Since each *Task* object links to a **MOSEK** task resource in a linked library, it is sometimes the case that the garbage collector is unable to reclaim it automatically. This means that substantial amounts of memory may be leaked. For this reason it is very important to make sure that the *Task* object is disposed of, either automatically or manually, when it is not used any more.

It is recommended to use a construction such as

```
try {
    env = new mosek.Env();
    task = new mosek.Task(env, 0,0);
    // ...
    // ... optimization ...
    // ...
}
finally {
    if (task != null) task.dispose();
    if (env != null) env.dispose();
}
```

This construction assures that the *Task.dispose* method is called when the object goes out of scope, even if an exception occurred. If this approach cannot be used, e.g. if the *Task* object is returned by a factory function, one should explicitly call the *Task.dispose* method when the object is no longer used. The same applies to the environment object.

Chapter 11

Case Studies

In this section we present some case studies in which the Optimizer API for Java is used to solve real-life applications. These examples involve some more advanced modeling skills and possibly some input data. The user is strongly recommended to first read the basic tutorials of [Sec. 6](#) before going through these advanced case studies.

- *Portfolio Optimization*
 - **Keywords:** Markowitz model, variance, risk, efficient frontier, transaction cost, market impact cost
 - **Type:** Conic Quadratic, Power Cone, Mixed-Integer Optimization
- *Logistic regression*
 - **Keywords:** machine learning, logistic regression, classifier, log-sum-exp, softplus, regularization
 - **Type:** Exponential Cone, Quadratic Cone
- *Concurrent Optimizer*
 - **Keywords:** Concurrent optimization
 - **Type:** Linear Optimization, Mixed-Integer Optimization

11.1 Portfolio Optimization

In this section the Markowitz portfolio optimization problem and variants are implemented using the **MOSEK** optimizer API.

- *Basic Markowitz model*
- *Efficient frontier*
- *Factor model and efficiency*
- *Market impact costs*
- *Transaction costs*
- *Cardinality constraints*


```

try ( mosek.Env env    = new mosek.Env ();
      mosek.Task task = new mosek.Task (env, 0, 0) ) {
    // Directs the log task stream to the user specified
    // method task_msg_obj.stream
    task.set_Stream(
        mosek.streamtype.log,
        new mosek.Stream()
    { public void stream(String msg) { System.out.print(msg); } });

    // Constraints.
    task.appendcons(numcon);

    // Constraint bounds. Compute total budget.
    totalBudget = w;
    for (int i = 0; i < n; ++i)
    {
        totalBudget += x0[i];
        /* Constraint bounds  $c^l = c^u = 0$  */
        task.putconbound(i + 1, mosek.boundkey.fx, 0.0, 0.0);
        task.putconname(i + 1, "GT[" + (i + 1) + "]");
    }
    /* The total budget constraint  $c^l = c^u = \text{totalBudget}$  in first row of A. */
    task.putconbound(0, mosek.boundkey.fx, totalBudget, totalBudget);
    task.putconname(0, "budget");

    // Variables.
    task.appendvars(numvar);

    /* x variables. */
    for (int j = 0; j < n; ++j)
    {
        /* Return of asset j in the objective */
        task.putcj(offsetx + j, mu[j]);
        /* Coefficients in the first row of A */
        task.putaij(0, offsetx + j, 1.0);
        /* No short-selling -  $x^l = 0$ ,  $x^u = \text{inf}$  */
        task.putvarbound(offsetx + j, mosek.boundkey.lo, 0.0, infinity);
        task.putvarname(offsetx + j, "x[" + (j + 1) + "]");
    }

    /* s variable is a constant equal to gamma. */
    task.putvarbound(offsets, mosek.boundkey.fx, gamma, gamma);
    task.putvarname(offsets, "s");

    /* t variables ( $t = GT \cdot x$ ). */
    for (int j = 0; j < n; ++j)
    {
        /* Copying the GT matrix in the appropriate block of A */
        for (int k = 0; k < n; ++k)
            if ( GT[k][j] != 0.0 )
                task.putaij(1 + k, offsetx + j, GT[k][j]);
        /* Diagonal -1 entries in a block of A */
        task.putaij(1 + j, offsett + j, -1.0);
        /* Free - no bounds */
        task.putvarbound(offsett + j, mosek.boundkey.fr, -infinity, infinity);
    }
}

```

(continues on next page)

(continued from previous page)

```
    task.putvarname(offsett + j, "t[" + (j + 1) + "]");
}

/* Define the cone spanned by (s, t), i.e. of dimension n + 1 */
int[] csub = new int[n + 1];
csub[0] = offsett;
for(int j = 0; j < n; j++) csub[j + 1] = offsett + j;
task.appendcone( mosek.conetype.quad,
                0.0, /* For future use only, can be set to 0.0 */
                csub );
task.putconename(0, "stddev");

/* A maximization problem */
task.putobjsense(mosek.objsense.maximize);

task.optimize();

/* Display solution summary for quick inspection of results */
task.solutionsummary(mosek.streamtype.log);

task.writedata("dump.opf");

/* Read the results */
double expret = 0.0;
double[] xx = new double[n + 1];

task.getxxslice(mosek.soltype.itr, 0, offsett + 1, xx);
for (int j = 0; j < n; ++j)
    expret += mu[j] * xx[j + offsett];

System.out.printf("\nExpected return %e for gamma %e\n", expret, xx[offsett]);
}
}
}
```

The above code produces the result:


```

public static void main (String[] args) {
    // Since the value infinity is never used, we define
    // 'infinity' symbolic purposes only
    double infinity = 0;
    double[] mu = {0.1073, 0.0737, 0.0627};
    double[][] GT = {
        {0.1667, 0.0232, 0.0013},
        {0.0000, 0.1033, -0.0022},
        {0.0000, 0.0000, 0.0338}
    };
    double[] x0 = {0.0, 0.0, 0.0};
    double w = 1.0;
    double[] alphas = {0.0, 0.25, 0.5, 0.75, 1.0, 1.5, 2.0, 2.5, 3.0, 3.5, 4.0, 4.5};
    int numalphas = 12;

    int numvar = 2 * n + 2;
    int numcon = n + 1;

    //Offset of variables into the API variable.
    int offsetx = 0;
    int offsets = n;
    int offsett = n + 1;
    int offsetu = 2 * n + 1;

    try ( Env env = new mosek.Env ();
          Task task = new mosek.Task (env, 0, 0) ) {

        // Directs the log task stream to the user specified
        // method task_msg_obj.stream
        task.set_Stream(
            mosek.streamtype.log,
            new mosek.Stream()
        { public void stream(String msg) { System.out.print(msg); } });

        task.appendcons(numcon);
        task.appendvars(numvar);

        //Constraints.
        for ( int i = 1; i <= n; ++i) {
            w += x0[i - 1];
            task.putconbound(i, mosek.boundkey.fx, 0., 0.);
            task.putconname(i, "GT[" + i + "]");
        }
        for (int i = 0; i < n; ++i)
            task.putaij(0, offsetx + i, 1.0);
        task.putconbound(0, mosek.boundkey.fx, w, w);
        task.putconname(0, "budget");

        // Objective coefficients
        for(int i = 0; i < n; i++)
            task.putcj(offsetx + i, mu[i]);

        for ( int i = 0; i < n; ++i) {
            // The constraint  $t = GT * x$ 
            for ( int j = i; j < n; ++j)

```

```

    task.putaij(i + 1, offsetx + j, GT[i][j]);

    task.putaij(i + 1, offsett + i, -1.0);

    // x variables
    task.putvarbound(offsetx + i, mosek.boundkey.lo, 0., 0.);

    task.putvarname(offsetx + i, "x[" + (i + 1) + "]");
    task.putvarname(offsett + i, "t[" + (i + 1) + "]");

    // t variable
    task.putvarbound(offsett + i, mosek.boundkey.fr, -infinity, infinity);
}

// s variable
task.putvarbound(offsets, mosek.boundkey.fr, -infinity, infinity);
task.putvarname(offsets, "s");

// u variable
task.putvarbound(offsetu, mosek.boundkey.fx, 0.5, 0.5);
task.putvarname(offsetu, "u");

//Cones.
int[] csub = new int[n+2];
csub[0] = offsets;
csub[1] = offsetu;
for(int i = 0; i < n; i++)
    csub[i+2] = offsett + i;

task.appendcone(mosek.conetype.rquad,
                0.0, /* For future use only, can be set to 0.0 */
                csub);
task.putconename(0, "variance");

/* A maximization problem */
task.putobjsense(mosek.objsense.maximize);

//task.writedata("dump.opf");

try {
    //Turn all log output off.
    task.putintparam(mosek.iparam.log, 0);

    System.out.printf("%-12s  %-12s  %-12s", "alpha", "exp ret", "variance");
    for (int k = 0; k < numalphas; ++k) {
        task.putcj(offsets, -alphas[k]);

        task.optimize();

        task.solutionsummary(mosek.streamtype.log);

        double expret = 0.0, stddev = 0.0;
        double[] xx = new double[numvar];

        task.getxx(mosek.soltype.itr, xx);
    }
}

```


(continued from previous page)

```
public int firstStop;
public CallbackProxy()
{
    stop = false;
    firstStop = -1;
}

public int progress(mosek.callbackcode caller)
{
    // Return non-zero implies terminate the optimizer
    return stop ? 1 : 0;
}
}
```

When all remaining tasks respond to the stop signal, response codes and statuses are returned to the caller, together with the index of the task which won the race.

Listing 11.13: A routine for parallel task race.

```
public static int optimize(mosek.Task[] tasks,
                          mosek.rescode[] res,
                          mosek.rescode[] trm)
{
    int n = tasks.length;
    Thread[] jobs = new Thread[n];

    // Set a callback function
    final CallbackProxy cb = new CallbackProxy();
    for (int i = 0; i < n; ++i)
        tasks[i].set_Progress(cb);

    // Initialize
    for (int i = 0; i < n; ++i)
    {
        res[i] = mosek.rescode.err_unknown;
        trm[i] = mosek.rescode.err_unknown;
    }

    // Start parallel optimizations, one per task
    for (int i = 0; i < n; ++i)
    {
        int num = i;
        jobs[i] = new Thread() { public void run() {
            try
            {
                trm[num] = tasks[num].optimize();
                res[num] = mosek.rescode.ok;
            }
            catch (mosek.Exception e)
            {
                trm[num] = mosek.rescode.err_unknown;
                res[num] = e.code;
            }
            finally
            {
                // If this finished with success, inform other tasks to interrupt
                if (res[num] == mosek.rescode.ok)

```

(continues on next page)

```

        {
            if (!cb.stop) cb.firstStop = num;
            cb.stop = true;
        }
    }
    });
    jobs[i].start();
}

// Join all threads
try {
    for (Thread j: jobs)
        j.join();
}
catch (InterruptedException e) {}

// For debugging, print res and trm codes for all optimizers
for (int i = 0; i < n; ++i)
    System.out.println("Optimizer " + i + " res " + res[i] + " trm " + trm[i]);

return cb.firstStop;
}

```

11.3.2 Linear optimization

We use the multithreaded setup to run the interior-point and simplex optimizers simultaneously on a linear problem. The next methods simply clones the given task and sets a different optimizer for each. The result is the clone which finished first.

Listing 11.14: Concurrent optimization with different optimizers.

```

public static int optimizeconcurrent(mosek.Task task,
                                     mosek.optimizertype[] optimizers,
                                     mosek.Task[] winTask,
                                     mosek.rescode[] winTrm,
                                     mosek.rescode[] winRes)
{
    int n = optimizers.length;
    mosek.Task[] tasks = new mosek.Task[n];
    mosek.rescode[] res = new mosek.rescode[n];
    mosek.rescode[] trm = new mosek.rescode[n];

    // Clone tasks and choose various optimizers
    for (int i = 0; i < n; ++i)
    {
        tasks[i] = new mosek.Task(task);
        tasks[i].putintparam(mosek.iparam.optimizer, optimizers[i].value);
    }

    // Solve tasks in parallel
    int firstOK = optimize(tasks, res, trm);

    if (firstOK >= 0)
    {
        winTask[0] = tasks[firstOK];
        winTrm[0] = trm[firstOK];
    }
}

```

(continues on next page)

(continued from previous page)

```
System.out.println(i + " " + tasks[i].getprimalobj(mosek.soltype.itg));

for (int i = 0; i < n; ++i)
    if ((res[i] == mosek.rescode.ok) &&
        (tasks[i].getsolsta(mosek.soltype.itg) == mosek.solsta.prim_feas ||
         tasks[i].getsolsta(mosek.soltype.itg) == mosek.solsta.integer_optimal) &&
        ((sense == mosek.objsense.minimize) ?
         (tasks[i].getprimalobj(mosek.soltype.itg) < bestObj) :
         (tasks[i].getprimalobj(mosek.soltype.itg) > bestObj) ) )
    {
        bestObj = tasks[i].getprimalobj(mosek.soltype.itg);
        bestPos = i;
    }

if (bestPos != -1)
{
    winTask[0] = tasks[bestPos];
    winTrm[0] = trm[bestPos];
    winRes[0] = res[bestPos];
    return bestPos;
}

return -1;
}
```

It remains to call the method with a choice of seeds, for example:

Listing 11.17: Calling concurrent integer optimization.

```
int[] seeds = { 42, 13, 71749373 };

idx = optimizeconcurrentMIO(task, seeds, t, trm, res);
```



```

        " rightprice = " + rightpricej[i] +
        " leftrange = " + leftrangej[i] +
        " rightrange = " + rightrangej[i] + "\n");

double[] leftprice = new double[2];
double[] rightprice = new double[2];
double[] leftrange = new double[2];
double[] rightrange = new double[2];
int subc[] = {2, 5};

task.dualsensitivity( subc,
                      leftprice,
                      rightprice,
                      leftrange,
                      rightrange
                      );

System.out.println(
    "Results from sensitivity analysis on objective coefficients:"
);

for (int i = 0; i < 2; ++i)
    System.out.print("leftprice = " + leftprice[i] +
                     " rightprice = " + rightprice[i] +
                     " leftrange = " + leftrange[i] +
                     " rightrange = " + rightrange[i] + "\n");

} catch (mosek.Exception e)
    /* Catch both mosek.Error and mosek.Warning */
{
    System.out.println ("An error or warning was encountered");
    System.out.println (e.getMessage ());
    throw e;
}
}
}

```


The type of sensitivity analysis to perform (basis or optimal partition) is controlled by the parameter *iparam.sensitivity_type*.

For an example, please see Section *Example: Sensitivity Analysis*.

Parameters

- **subj** (int[]) – Indexes of objective coefficients to analyze. (input)
- **leftpricej** (double[]) – **leftpricej[j]** is the left shadow price for the coefficient with index **subj[j]**. (output)
- **rightpricej** (double[]) – **rightpricej[j]** is the right shadow price for the coefficient with index **subj[j]**. (output)
- **leftrangej** (double[]) – **leftrangej[j]** is the left range β_1 for the coefficient with index **subj[j]**. (output)
- **rightrangej** (double[]) – **rightrangej[j]** is the right range β_2 for the coefficient with index **subj[j]**. (output)

Groups *Sensitivity analysis*

Task.generateconenames

```
void generateconenames
(int[] subk,
String fmt,
int[] dims,
long[] sp)
```

Generates systematic names for cone.

Parameters

- **subk** (int[]) – Indexes of the cone. (input)
- **fmt** (String) – The cone name formatting string. (input)
- **dims** (int[]) – Dimensions in the shape. (input)
- **sp** (long[]) – Items that should be named. (input)

Groups *Names, Problem data - cones*

Task.generateconnames

```
void generateconnames
(int[] subi,
String fmt,
int[] dims,
long[] sp)
```

Generates systematic names for constraints.

Parameters

- **subi** (int[]) – Indexes of the constraints. (input)
- **fmt** (String) – The constraint name formatting string. (input)
- **dims** (int[]) – Dimensions in the shape. (input)
- **sp** (long[]) – Items that should be named. (input)

Groups *Names, Problem data - constraints, Problem data - linear part*

Task.generatevarnames

```
void generatevarnames
(int[] subj,
String fmt,
int[] dims,
long[] sp)
```

Generates systematic names for variables.

Parameters

- subj (int[]) – Indexes of the variables. (input)
- fmt (String) – The variable name formatting string. (input)
- dims (int[]) – Dimensions in the shape. (input)
- sp (long[]) – Items that should be named. (input)

Groups *Names, Problem data - variables, Problem data - linear part*

Task.getacol

```
void getacol
(int j,
 int[] nzj,
 int[] subj,
 double[] valj)
```

Obtains one column of A in a sparse format.

Parameters

- j (int) – Index of the column. (input)
- nzj (int *by reference*) – Number of non-zeros in the column obtained. (output)
- subj (int[]) – Row indices of the non-zeros in the column obtained. (output)
- valj (double[]) – Numerical values in the column obtained. (output)

Groups *Problem data - linear part, Inspecting the task*

Task.getacolnumnz

```
void getacolnumnz
(int i,
 int[] nzj)
```

```
int getacolnumnz (int i)
```

Obtains the number of non-zero elements in one column of A .

Parameters

- i (int) – Index of the column. (input)
- nzj (int *by reference*) – Number of non-zeros in the j -th column of A . (output)

Return (int) – Number of non-zeros in the j -th column of A .

Groups *Problem data - linear part, Inspecting the task*

Task.getacolslice

```
void getacolslice
(int first,
 int last,
 long[] ptrb,
 long[] ptre,
 int[] sub,
 double[] val)
```

Obtains a sequence of columns from A in sparse format.

Parameters

- first (int) – Index of the first column in the sequence. (input)


```
void getbarcblocktriplet
(long[] num,
 int[] subj,
 int[] subk,
 int[] subl,
 double[] valjkl)
```

```
long getbarcblocktriplet
(int[] subj,
 int[] subk,
 int[] subl,
 double[] valjkl)
```

Obtains \overline{C} in block triplet form.

Parameters

- num (long *by reference*) – Number of elements in the block triplet form. (output)
- subj (int[]) – Symmetric matrix variable index. (output)
- subk (int[]) – Block row index. (output)
- subl (int[]) – Block column index. (output)
- valjkl (double[]) – The numerical value associated with each block triplet. (output)

Return (long) – Number of elements in the block triplet form.

Groups *Problem data - semidefinite, Inspecting the task*

Task.getbarcidx

```
void getbarcidx
(long idx,
 int[] j,
 long[] num,
 long[] sub,
 double[] weights)
```

Obtains information about an element in \overline{C} .

Parameters

- idx (long) – Index of the element for which information should be obtained. (input)
- j (int *by reference*) – Row index in \overline{C} . (output)
- num (long *by reference*) – Number of terms in the weighted sum. (output)
- sub (long[]) – Elements appearing the weighted sum. (output)
- weights (double[]) – Weights of terms in the weighted sum. (output)

Groups *Problem data - semidefinite, Inspecting the task*

Task.getbarcidxinfo

```
void getbarcidxinfo
(long idx,
 long[] num)
```

```
long getbarcidxinfo (long idx)
```

Obtains the number of terms in the weighted sum that forms a particular element in \overline{C} .

Parameters

Obtains bound information for one constraint.

Parameters

- **i** (**int**) – Index of the constraint for which the bound information should be obtained. (input)
- **bk** (*mosek.boundkey by reference*) – Bound keys. (output)
- **bl** (*double by reference*) – Values for lower bounds. (output)
- **bu** (*double by reference*) – Values for upper bounds. (output)

Groups *Problem data - linear part, Inspecting the task, Problem data - bounds, Problem data - constraints*

`Task.getconboundslice`

```
void getconboundslice
(int first,
 int last,
 mosek.boundkey[] bk,
 double[] bl,
 double[] bu)
```

Obtains bounds information for a slice of the constraints.

Parameters

- **first** (**int**) – First index in the sequence. (input)
- **last** (**int**) – Last index plus 1 in the sequence. (input)
- **bk** (*mosek.boundkey []*) – Bound keys. (output)
- **bl** (*double []*) – Values for lower bounds. (output)
- **bu** (*double []*) – Values for upper bounds. (output)

Groups *Problem data - linear part, Inspecting the task, Problem data - bounds, Problem data - constraints*

`Task.getcone`

```
void getcone
(int k,
 mosek.conetype[] ct,
 double[] coneapar,
 int[] nummem,
 int[] submem)
```

Obtains a cone.

Parameters

- **k** (**int**) – Index of the cone. (input)
- **ct** (*mosek.conetype by reference*) – Specifies the type of the cone. (output)
- **coneapar** (*double by reference*) – For the power cone it denotes the exponent alpha. For other cone types it is unused and can be set to 0. (output)
- **nummem** (*int by reference*) – Number of member variables in the cone. (output)
- **submem** (*int []*) – Variable subscripts of the members in the cone. (output)

Groups *Inspecting the task, Problem data - cones*

`Task.getconeinfo`

Parameters

- `k` (int) – Which constraint. (input)
- `numqcnz` (long *by reference*) – Number of quadratic terms. (output)
- `qcsubi` (int[]) – Row subscripts for quadratic constraint matrix. (output)
- `qcsobj` (int[]) – Column subscripts for quadratic constraint matrix. (output)
- `qcval` (double[]) – Quadratic constraint coefficient values. (output)

Return (long) – Number of quadratic terms.

Groups *Inspecting the task, Problem data - quadratic part, Problem data - constraints*

Task.getqobj

```
void getqobj
(long[] numqonz,
 int[] qosubi,
 int[] qosobj,
 double[] qoval)
```

Obtains the quadratic terms in the objective. The required quadratic terms are stored sequentially in `qosubi`, `qosobj`, and `qoval`.

Parameters

- `numqonz` (long *by reference*) – Number of non-zero elements in the quadratic objective terms. (output)
- `qosubi` (int[]) – Row subscripts for quadratic objective coefficients. (output)
- `qosobj` (int[]) – Column subscripts for quadratic objective coefficients. (output)
- `qoval` (double[]) – Quadratic objective coefficient values. (output)

Groups *Inspecting the task, Problem data - quadratic part*

Task.getqobjij

```
void getqobjij
(int i,
 int j,
 double[] qoij)
```

Obtains one coefficient q_{ij}^o in the quadratic term of the objective.

Parameters

- `i` (int) – Row index of the coefficient. (input)
- `j` (int) – Column index of coefficient. (input)
- `qoij` (double *by reference*) – The required coefficient. (output)

Groups *Inspecting the task, Problem data - quadratic part*

Task.getreducedcosts

```
void getreducedcosts
(mosek.soltype whichsol,
 int first,
 int last,
 double[] redcosts)
```

Computes the reduced costs for a slice of variables and returns them in the array `redcosts` i.e.

$$\text{redcosts}[j - \text{first}] = (s_l^x)_j - (s_u^x)_j, \quad j = \text{first}, \dots, \text{last} - 1 \quad (15.2)$$

Task.getslx

```
void getslx
(mosek.soltype whichsol,
double[] slx)
```

Obtains the s_l^x vector for a solution.

Parameters

- `whichsol` (*mosek.soltype*) – Selects a solution. (input)
- `slx` (`double[]`) – Dual variables corresponding to the lower bounds on the variables. (output)

Groups *Solution - dual*

Task.getslxslice

```
void getslxslice
(mosek.soltype whichsol,
int first,
int last,
double[] slx)
```

Obtains a slice of the s_l^x vector for a solution.

Parameters

- `whichsol` (*mosek.soltype*) – Selects a solution. (input)
- `first` (`int`) – First index in the sequence. (input)
- `last` (`int`) – Last index plus 1 in the sequence. (input)
- `slx` (`double[]`) – Dual variables corresponding to the lower bounds on the variables. (output)

Groups *Solution - dual*

Task.getsnx

```
void getsnx
(mosek.soltype whichsol,
double[] snx)
```

Obtains the s_n^x vector for a solution.

Parameters

- `whichsol` (*mosek.soltype*) – Selects a solution. (input)
- `snx` (`double[]`) – Dual variables corresponding to the conic constraints on the variables. (output)

Groups *Solution - dual*

Task.getsnxslice

```
void getsnxslice
(mosek.soltype whichsol,
int first,
int last,
double[] snx)
```

Obtains a slice of the s_n^x vector for a solution.

Parameters

- `snx (double[])` – Dual variables corresponding to the conic constraints on the variables. (output)

Groups *Solution information, Solution - primal, Solution - dual*

`Task.getsolutioninfo`

```
void getsolutioninfo
(mosek.soltype whichsol,
 double[] pobj,
 double[] pviolcon,
 double[] pviolvar,
 double[] pviolbarvar,
 double[] pviolcone,
 double[] pviolitg,
 double[] dobj,
 double[] dviolcon,
 double[] dviolvar,
 double[] dviolbarvar,
 double[] dviolcone)
```

Obtains information about a solution.

Parameters

- `whichsol (mosek.soltype)` – Selects a solution. (input)
- `pobj (double by reference)` – The primal objective value as computed by `Task.getprimalobj`. (output)
- `pviolcon (double by reference)` – Maximal primal violation of the solution associated with the x^c variables where the violations are computed by `Task.getpviolcon`. (output)
- `pviolvar (double by reference)` – Maximal primal violation of the solution for the x variables where the violations are computed by `Task.getpviolvar`. (output)
- `pviolbarvar (double by reference)` – Maximal primal violation of solution for the \bar{X} variables where the violations are computed by `Task.getpviolbarvar`. (output)
- `pviolcone (double by reference)` – Maximal primal violation of solution for the conic constraints where the violations are computed by `Task.getpviolcones`. (output)
- `pviolitg (double by reference)` – Maximal violation in the integer constraints. The violation for an integer variable x_j is given by $\min(x_j - \lfloor x_j \rfloor, \lceil x_j \rceil - x_j)$. This number is always zero for the interior-point and basic solutions. (output)
- `dobj (double by reference)` – Dual objective value as computed by `Task.getdualobj`. (output)
- `dviolcon (double by reference)` – Maximal violation of the dual solution associated with the x^c variable as computed by `Task.getdviolcon`. (output)
- `dviolvar (double by reference)` – Maximal violation of the dual solution associated with the x variable as computed by `Task.getdviolvar`. (output)
- `dviolbarvar (double by reference)` – Maximal violation of the dual solution associated with the \bar{S} variable as computed by `Task.getdviolbarvar`. (output)
- `dviolcone (double by reference)` – Maximal violation of the dual solution associated with the dual conic constraints as computed by `Task.getdviolcones`. (output)

Groups *Solution information*

`Task.getsolutionslice`

Task.putcone

```
void putcone
(int k,
 mosek.conetype ct,
 double coneapar,
 int[] submem)
```

Replaces a conic constraint.

Parameters

- **k** (**int**) – Index of the cone. (input)
- **ct** (*mosek.conetype*) – Specifies the type of the cone. (input)
- **coneapar** (**double**) – For the power cone it denotes the exponent alpha. For other cone types it is unused and can be set to 0. (input)
- **submem** (**int[]**) – Variable subscripts of the members in the cone. (input)

Groups *Problem data - cones*

Task.putconename

```
void putconename
(int j,
 String name)
```

Sets the name of a cone.

Parameters

- **j** (**int**) – Index of the cone. (input)
- **name** (**String**) – The name of the cone. (input)

Groups *Names, Problem data - cones*

Task.putconname

```
void putconname
(int i,
 String name)
```

Sets the name of a constraint.

Parameters

- **i** (**int**) – Index of the constraint. (input)
- **name** (**String**) – The name of the constraint. (input)

Groups *Names, Problem data - constraints, Problem data - linear part*

Task.putconsolutioni

```
void putconsolutioni
(int i,
 mosek.soltype whichsol,
 mosek.stakey sk,
 double x,
 double sl,
 double su)
```

Sets the primal and dual solution information for a single constraint.

Parameters

- `i` (`int`) – Index of the constraint. (input)
- `whichsol` (`mosek.soltype`) – Selects a solution. (input)
- `sk` (`mosek.stakey`) – Status key of the constraint. (input)
- `x` (`double`) – Primal solution value of the constraint. (input)
- `sl` (`double`) – Solution value of the dual variable associated with the lower bound. (input)
- `su` (`double`) – Solution value of the dual variable associated with the upper bound. (input)

Groups *Solution information, Solution - primal, Solution - dual*

`Task.putcslice`

```
void putcslice
(int first,
 int last,
 double[] slice)
```

Modifies a slice in the linear term c in the objective using the principle

$$c_j = \text{slice}[j - \text{first}], \quad j = \text{first}, \dots, \text{last} - 1$$

Data checks are performed as in *Task.putcj*.

Parameters

- `first` (`int`) – First element in the slice of c . (input)
- `last` (`int`) – Last element plus 1 of the slice in c to be changed. (input)
- `slice` (`double[]`) – New numerical values for coefficients in c that should be modified. (input)

Groups *Problem data - linear part, Problem data - objective*

`Task.putdouparam`

```
void putdouparam
(mosek.dparam param,
 double parvalue)
```

Sets the value of a double parameter.

Parameters

- `param` (`mosek.dparam`) – Which parameter. (input)
- `parvalue` (`double`) – Parameter value. (input)

Groups *Parameters*

`Task.putintparam`

```
void putintparam
(mosek.iparam param,
 int parvalue)
```

Sets the value of an integer parameter.

Please notice that some parameters take values that are defined in Enum classes. This function accepts only integer values, so to use e.g. the value *onoffkey.on*, is necessary to use the member *.value*. For example:

```
task.putintparam(mosek.iparam.opf_write_problem, mosek.onoffkey.on.value)
```

Parameters

- `param` (*mosek.iparam*) – Which parameter. (input)
- `parvalue` (`int`) – Parameter value. (input)

Groups *Parameters*

`Task.putmaxnumanz`

```
void putmaxnumanz (long maxnumanz)
```

Sets the number of preallocated non-zero entries in A .

MOSEK stores only the non-zero elements in the linear coefficient matrix A and it cannot predict how much storage is required to store A . Using this function it is possible to specify the number of non-zeros to preallocate for storing A .

If the number of non-zeros in the problem is known, it is a good idea to set `maxnumanz` slightly larger than this number, otherwise a rough estimate can be used. In general, if A is inputted in many small chunks, setting this value may speed up the data input phase.

It is not mandatory to call this function, since **MOSEK** will reallocate internal structures whenever it is necessary.

The function call has no effect if both `maxnumcon` and `maxnumvar` are zero.

Parameters `maxnumanz` (`long`) – Number of preallocated non-zeros in A . (input)

Groups *Environment and task management, Problem data - semidefinite*

`Task.putmaxnumberbarvar`

```
void putmaxnumberbarvar (int maxnumberbarvar)
```

Sets the number of preallocated symmetric matrix variables in the optimization task. When this number of variables is reached **MOSEK** will automatically allocate more space for variables.

It is not mandatory to call this function. It only gives a hint about the amount of data to preallocate for efficiency reasons.

Please note that `maxnumberbarvar` must be larger than the current number of symmetric matrix variables in the task.

Parameters `maxnumberbarvar` (`int`) – Number of preallocated symmetric matrix variables. (input)

Groups *Environment and task management, Problem data - semidefinite*

`Task.putmaxnumcon`

```
void putmaxnumcon (int maxnumcon)
```

Sets the number of preallocated constraints in the optimization task. When this number of constraints is reached **MOSEK** will automatically allocate more space for constraints.

It is never mandatory to call this function, since **MOSEK** will reallocate any internal structures whenever it is required.

Please note that `maxnumcon` must be larger than the current number of constraints in the task.

Parameters `maxnumcon` (`int`) – Number of preallocated constraints in the optimization task. (input)

Groups *Environment and task management, Problem data - constraints*

`Task.putmaxnumcone`

```
void putmaxnumcone (int maxnumcone)
```

Sets the number of preallocated conic constraints in the optimization task. When this number of conic constraints is reached **MOSEK** will automatically allocate more space for conic constraints.

It is not mandatory to call this function, since **MOSEK** will reallocate any internal structures whenever it is required.

Please note that `maxnumcon` must be larger than the current number of conic constraints in the task.

Parameters `maxnumcone (int)` – Number of preallocated conic constraints in the optimization task. (input)

Groups *Environment and task management, Problem data - cones*

`Task.putmaxnumqnz`

```
void putmaxnumqnz (long maxnumqnz)
```

Sets the number of preallocated non-zero entries in quadratic terms.

MOSEK stores only the non-zero elements in Q . Therefore, **MOSEK** cannot predict how much storage is required to store Q . Using this function it is possible to specify the number non-zeros to preallocate for storing Q (both objective and constraints).

It may be advantageous to reserve more non-zeros for Q than actually needed since it may improve the internal efficiency of **MOSEK**, however, it is never worthwhile to specify more than the double of the anticipated number of non-zeros in Q .

It is not mandatory to call this function, since **MOSEK** will reallocate internal structures whenever it is necessary.

Parameters `maxnumqnz (long)` – Number of non-zero elements preallocated in quadratic coefficient matrices. (input)

Groups *Environment and task management, Problem data - quadratic part*

`Task.putmaxnumvar`

```
void putmaxnumvar (int maxnumvar)
```

Sets the number of preallocated variables in the optimization task. When this number of variables is reached **MOSEK** will automatically allocate more space for variables.

It is not mandatory to call this function. It only gives a hint about the amount of data to preallocate for efficiency reasons.

Please note that `maxnumvar` must be larger than the current number of variables in the task.

Parameters `maxnumvar (int)` – Number of preallocated variables in the optimization task. (input)

Groups *Environment and task management, Problem data - variables*

`Task.putnadoupam`

```
void putnadoupam  
(String paramname,  
 double parvalue)
```

Sets the value of a named double parameter.

Parameters

- `paramname (String)` – Name of a parameter. (input)

- `parvalue (double)` – Parameter value. (input)

Groups *Parameters*

`Task.putnaintparam`

```
void putnaintparam
(String paramname,
 int parvalue)
```

Sets the value of a named integer parameter.

Parameters

- `paramname (String)` – Name of a parameter. (input)
- `parvalue (int)` – Parameter value. (input)

Groups *Parameters*

`Task.putnastrparam`

```
void putnastrparam
(String paramname,
 String parvalue)
```

Sets the value of a named string parameter.

Parameters

- `paramname (String)` – Name of a parameter. (input)
- `parvalue (String)` – Parameter value. (input)

Groups *Parameters*

`Task.putobjname`

```
void putobjname (String objname)
```

Assigns a new name to the objective.

Parameters `objname (String)` – Name of the objective. (input)

Groups *Problem data - linear part, Names, Problem data - objective*

`Task.putobjsense`

```
void putobjsense (mosek.objsense sense)
```

Sets the objective sense of the task.

Parameters `sense (mosek.objsense)` – The objective sense of the task. The values *objsense.maximize* and *objsense.minimize* mean that the problem is maximized or minimized respectively. (input)

Groups *Problem data - linear part, Problem data - objective*

`Task.putoptserverhost`

```
void putoptserverhost (String host)
```

Specify an OptServer URL for remote calls. The URL should contain protocol, host and port in the form `http://server:port`. If the URL is set using this function, all subsequent calls to any **MOSEK** function that involves synchronous optimization will be sent to the specified OptServer instead of being executed locally. Passing NULL deactivates this redirection.

Parameters `host` (`String`) – A URL specifying the optimization server to be used. (input)

Groups *Remote optimization*

`Task.putparam`

```
void putparam
(String parname,
String parvalue)
```

Checks if `parname` is valid parameter name. If it is, the parameter is assigned the value specified by `parvalue`.

Parameters

- `parname` (`String`) – Parameter name. (input)
- `parvalue` (`String`) – Parameter value. (input)

Groups *Parameters*

`Task.putqcon`

```
void putqcon
(int[] qcsubk,
int[] qcsubi,
int[] qcsubj,
double[] qcval)
```

Replace all quadratic entries in the constraints. The list of constraints has the form

$$l_k^c \leq \frac{1}{2} \sum_{i=0}^{numvar-1} \sum_{j=0}^{numvar-1} q_{ij}^k x_i x_j + \sum_{j=0}^{numvar-1} a_{kj} x_j \leq u_k^c, \quad k = 0, \dots, m-1.$$

This function sets all the quadratic terms to zero and then performs the update:

$$q_{qcsubi[t], qcsubj[t]}^{qcsubk[t]} = q_{qcsubj[t], qcsubi[t]}^{qcsubk[t]} = q_{qcsubj[t], qcsubi[t]}^{qcsubk[t]} + qcval[t],$$

for $t = 0, \dots, numqcnz - 1$.

Please note that:

- For large problems it is essential for the efficiency that the function `Task.putmaxnumqnz` is employed to pre-allocate space.
- Only the lower triangular parts should be specified because the Q matrices are symmetric. Specifying entries where $i < j$ will result in an error.
- Only non-zero elements should be specified.
- The order in which the non-zero elements are specified is insignificant.
- Duplicate elements are added together as shown above. Hence, it is usually not recommended to specify the same entry multiple times.

For a code example see Section *Quadratic Optimization*

Parameters

- `qcsubk` (`int[]`) – Constraint subscripts for quadratic coefficients. (input)
- `qcsubi` (`int[]`) – Row subscripts for quadratic constraint matrix. (input)
- `qcsubj` (`int[]`) – Column subscripts for quadratic constraint matrix. (input)
- `qcval` (`double[]`) – Quadratic constraint coefficient values. (input)

Groups *Problem data - quadratic part*

`Task.putqconk`

```

void putqconk
(int k,
 int[] qcsubi,
 int[] qcsubj,
 double[] qcval)

```

Replaces all the quadratic entries in one constraint. This function performs the same operations as *Task.putqcon* but only with respect to constraint number *k* and it does not modify the other constraints. See the description of *Task.putqcon* for definitions and important remarks.

Parameters

- *k* (int) – The constraint in which the new *Q* elements are inserted. (input)
- *qcsubi* (int[]) – Row subscripts for quadratic constraint matrix. (input)
- *qcsubj* (int[]) – Column subscripts for quadratic constraint matrix. (input)
- *qcval* (double[]) – Quadratic constraint coefficient values. (input)

Groups *Problem data - quadratic part*

Task.putqobj

```

void putqobj
(int[] qosubi,
 int[] qosubj,
 double[] qoval)

```

Replace all quadratic terms in the objective. If the objective has the form

$$\frac{1}{2} \sum_{i=0}^{numvar-1} \sum_{j=0}^{numvar-1} q_{ij}^o x_i x_j + \sum_{j=0}^{numvar-1} c_j x_j + c^f$$

then this function sets all the quadratic terms to zero and then performs the update:

$$q_{qosubi[t],qosubj[t]}^o = q_{qosubj[t],qosubi[t]}^o = q_{qosubj[t],qosubi[t]}^o + qoval[t],$$

for $t = 0, \dots, numqonz - 1$.

See the description of *Task.putqcon* for important remarks and example.

Parameters

- *qosubi* (int[]) – Row subscripts for quadratic objective coefficients. (input)
- *qosubj* (int[]) – Column subscripts for quadratic objective coefficients. (input)
- *qoval* (double[]) – Quadratic objective coefficient values. (input)

Groups *Problem data - quadratic part, Problem data - objective*

Task.putqobjij

```

void putqobjij
(int i,
 int j,
 double qoij)

```

Replaces one coefficient in the quadratic term in the objective. The function performs the assignment

$$q_{ij}^o = q_{ji}^o = qoij.$$

Only the elements in the lower triangular part are accepted. Setting q_{ij} with $j > i$ will cause an error.

Please note that replacing all quadratic elements one by one is more computationally expensive than replacing them all at once. Use *Task.putqobj* instead whenever possible.

Parameters

- `i` (int) – Row index for the coefficient to be replaced. (input)
- `j` (int) – Column index for the coefficient to be replaced. (input)
- `qoij` (double) – The new value for q_{ij}^o . (input)

Groups *Problem data - quadratic part, Problem data - objective*

`Task.putskc`

```
void putskc
(mosek.soltype whichsol,
 mosek.stakey[] skc)
```

Sets the status keys for the constraints.

Parameters

- `whichsol` (*mosek.soltype*) – Selects a solution. (input)
- `skc` (*mosek.stakey*[]) – Status keys for the constraints. (input)

Groups *Solution information*

`Task.putskcslice`

```
void putskcslice
(mosek.soltype whichsol,
 int first,
 int last,
 mosek.stakey[] skc)
```

Sets the status keys for a slice of the constraints.

Parameters

- `whichsol` (*mosek.soltype*) – Selects a solution. (input)
- `first` (int) – First index in the sequence. (input)
- `last` (int) – Last index plus 1 in the sequence. (input)
- `skc` (*mosek.stakey*[]) – Status keys for the constraints. (input)

Groups *Solution information*

`Task.putskx`

```
void putskx
(mosek.soltype whichsol,
 mosek.stakey[] skx)
```

Sets the status keys for the scalar variables.

Parameters

- `whichsol` (*mosek.soltype*) – Selects a solution. (input)
- `skx` (*mosek.stakey*[]) – Status keys for the variables. (input)

Groups *Solution information*

`Task.putskxslice`

```
void putskxslice
(mosek.soltype whichsol,
 int first,
 int last,
 mosek.stakey[] skx)
```

Sets the status keys for a slice of the variables.

Parameters

- `whichsol` (*mosek.soltype*) – Selects a solution. (input)
- `first` (int) – First index in the sequence. (input)
- `last` (int) – Last index plus 1 in the sequence. (input)
- `skx` (*mosek.stakey*[]) – Status keys for the variables. (input)

Groups *Solution information*

`Task.putslc`

```
void putslc
(mosek.soltype whichsol,
 double[] slc)
```

Sets the s_l^c vector for a solution.

Parameters

- `whichsol` (*mosek.soltype*) – Selects a solution. (input)
- `slc` (double[]) – Dual variables corresponding to the lower bounds on the constraints. (input)

Groups *Solution - dual*

`Task.putslcslice`

```
void putslcslice
(mosek.soltype whichsol,
 int first,
 int last,
 double[] slc)
```

Sets a slice of the s_l^c vector for a solution.

Parameters

- `whichsol` (*mosek.soltype*) – Selects a solution. (input)
- `first` (int) – First index in the sequence. (input)
- `last` (int) – Last index plus 1 in the sequence. (input)
- `slc` (double[]) – Dual variables corresponding to the lower bounds on the constraints. (input)

Groups *Solution - dual*

`Task.putslx`

```
void putslx
(mosek.soltype whichsol,
 double[] slx)
```

Sets the s_l^x vector for a solution.

Parameters

- `whichsol` (*mosek.soltype*) – Selects a solution. (input)
- `slx` (double[]) – Dual variables corresponding to the lower bounds on the variables. (input)

Groups *Solution - dual*

`Task.putslxslice`

```
void putslxslice
(mosek.soltype whichsol,
 int first,
 int last,
 double[] slx)
```

Sets a slice of the s_l^x vector for a solution.

Parameters

- `whichsol` (*mosek.soltype*) – Selects a solution. (input)
- `first` (int) – First index in the sequence. (input)
- `last` (int) – Last index plus 1 in the sequence. (input)
- `slx` (double[]) – Dual variables corresponding to the lower bounds on the variables. (input)

Groups *Solution - dual*

Task.putsnx

```
void putsnx
(mosek.soltype whichsol,
 double[] sux)
```

Sets the s_n^x vector for a solution.

Parameters

- `whichsol` (*mosek.soltype*) – Selects a solution. (input)
- `sux` (double[]) – Dual variables corresponding to the upper bounds on the variables. (input)

Groups *Solution - dual*

Task.putsnxslice

```
void putsnxslice
(mosek.soltype whichsol,
 int first,
 int last,
 double[] snx)
```

Sets a slice of the s_n^x vector for a solution.

Parameters

- `whichsol` (*mosek.soltype*) – Selects a solution. (input)
- `first` (int) – First index in the sequence. (input)
- `last` (int) – Last index plus 1 in the sequence. (input)
- `snx` (double[]) – Dual variables corresponding to the conic constraints on the variables. (input)

Groups *Solution - dual*

Task.putsolution

```
void putsolution
(mosek.soltype whichsol,
 mosek.stakey[] skc,
 mosek.stakey[] skx,
 mosek.stakey[] skn,
```

(continues on next page)

```
double[] xc,
double[] xx,
double[] y,
double[] slc,
double[] suc,
double[] slx,
double[] sux,
double[] snx)
```

Inserts a solution into the task.

Parameters

- `whichsol` (*mosek.soltype*) – Selects a solution. (input)
- `skc` (*mosek.stakey* []) – Status keys for the constraints. (input)
- `skx` (*mosek.stakey* []) – Status keys for the variables. (input)
- `skn` (*mosek.stakey* []) – Status keys for the conic constraints. (input)
- `xc` (double []) – Primal constraint solution. (input)
- `xx` (double []) – Primal variable solution. (input)
- `y` (double []) – Vector of dual variables corresponding to the constraints. (input)
- `slc` (double []) – Dual variables corresponding to the lower bounds on the constraints. (input)
- `suc` (double []) – Dual variables corresponding to the upper bounds on the constraints. (input)
- `slx` (double []) – Dual variables corresponding to the lower bounds on the variables. (input)
- `sux` (double []) – Dual variables corresponding to the upper bounds on the variables. (input)
- `snx` (double []) – Dual variables corresponding to the conic constraints on the variables. (input)

Groups *Solution information, Solution - primal, Solution - dual*

`Task.putsolutionyi`

```
void putsolutionyi
(int i,
 mosek.soltype whichsol,
 double y)
```

Inputs the dual variable of a solution.

Parameters

- `i` (int) – Index of the dual variable. (input)
- `whichsol` (*mosek.soltype*) – Selects a solution. (input)
- `y` (double) – Solution value of the dual variable. (input)

Groups *Solution information, Solution - dual*

`Task.putstrparam`

```
void putstrparam
(mosek.sparam param,
 String parvalue)
```

Sets the value of a string parameter.

Parameters

- param (*mosek.sparam*) – Which parameter. (input)
- parvalue (String) – Parameter value. (input)

Groups *Parameters*

Task.putsuc

```
void putsuc
(mosek.soltype whichsol,
 double[] suc)
```

Sets the s_u^c vector for a solution.

Parameters

- whichsol (*mosek.soltype*) – Selects a solution. (input)
- suc (double[]) – Dual variables corresponding to the upper bounds on the constraints. (input)

Groups *Solution - dual*

Task.putsucslice

```
void putsucslice
(mosek.soltype whichsol,
 int first,
 int last,
 double[] suc)
```

Sets a slice of the s_u^c vector for a solution.

Parameters

- whichsol (*mosek.soltype*) – Selects a solution. (input)
- first (int) – First index in the sequence. (input)
- last (int) – Last index plus 1 in the sequence. (input)
- suc (double[]) – Dual variables corresponding to the upper bounds on the constraints. (input)

Groups *Solution - dual*

Task.putsux

```
void putsux
(mosek.soltype whichsol,
 double[] sux)
```

Sets the s_u^x vector for a solution.

Parameters

- whichsol (*mosek.soltype*) – Selects a solution. (input)
- sux (double[]) – Dual variables corresponding to the upper bounds on the variables. (input)

Groups *Solution - dual*

Task.putsuxslice

```
void putsuxslice
(mosek.soltype whichsol,
 int first,
 int last,
 double[] sux)
```

Sets a slice of the s_u^x vector for a solution.

Parameters

- `whichsol` (*mosek.soltype*) – Selects a solution. (input)
- `first` (int) – First index in the sequence. (input)
- `last` (int) – Last index plus 1 in the sequence. (input)
- `sux` (double[]) – Dual variables corresponding to the upper bounds on the variables. (input)

Groups *Solution - dual*

`Task.puttaskname`

```
void puttaskname (String taskname)
```

Assigns a new name to the task.

Parameters `taskname` (String) – Name assigned to the task. (input)

Groups *Names, Environment and task management*

`Task.putvarbound`

```
void putvarbound  
(int j,  
 mosek.boundkey bkey,  
 double blx,  
 double bux)
```

Changes the bounds for one variable.

If the bound value specified is numerically larger than *dparam.data_tol_bound_inf* it is considered infinite and the bound key is changed accordingly. If a bound value is numerically larger than *dparam.data_tol_bound_wrn*, a warning will be displayed, but the bound is inputted as specified.

Parameters

- `j` (int) – Index of the variable. (input)
- `bkey` (*mosek.boundkey*) – New bound key. (input)
- `blx` (double) – New lower bound. (input)
- `bux` (double) – New upper bound. (input)

Groups *Problem data - linear part, Problem data - variables, Problem data - bounds*

`Task.putvarboundlist`

```
void putvarboundlist  
(int[] sub,  
 mosek.boundkey[] bkey,  
 double[] blx,  
 double[] bux)
```

Changes the bounds for one or more variables. If multiple bound changes are specified for a variable, then only the last change takes effect. Data checks are performed as in *Task.putvarbound*.

Parameters

- `sub` (int[]) – List of variable indexes. (input)
- `bkey` (*mosek.boundkey*[]) – Bound keys for the variables. (input)
- `blx` (double[]) – Lower bounds for the variables. (input)
- `bux` (double[]) – Upper bounds for the variables. (input)

Groups *Problem data - linear part, Problem data - variables, Problem data - bounds*

Task.putvarboundlistconst

```
void putvarboundlistconst
(int[] sub,
 mosek.boundkey bkg,
 double blx,
 double bux)
```

Changes the bounds for one or more variables. Data checks are performed as in *Task.putvarbound*.

Parameters

- `sub (int[])` – List of variable indexes. (input)
- `bkg (mosek.boundkey)` – New bound key for all variables in the list. (input)
- `blx (double)` – New lower bound for all variables in the list. (input)
- `bux (double)` – New upper bound for all variables in the list. (input)

Groups *Problem data - linear part, Problem data - variables, Problem data - bounds*

Task.putvarboundslice

```
void putvarboundslice
(int first,
 int last,
 mosek.boundkey[] bkg,
 double[] blx,
 double[] bux)
```

Changes the bounds for a slice of the variables. Data checks are performed as in *Task.putvarbound*.

Parameters

- `first (int)` – First index in the sequence. (input)
- `last (int)` – Last index plus 1 in the sequence. (input)
- `bkg (mosek.boundkey[])` – Bound keys for the variables. (input)
- `blx (double[])` – Lower bounds for the variables. (input)
- `bux (double[])` – Upper bounds for the variables. (input)

Groups *Problem data - linear part, Problem data - variables, Problem data - bounds*

Task.putvarboundsliceconst

```
void putvarboundsliceconst
(int first,
 int last,
 mosek.boundkey bkg,
 double blx,
 double bux)
```

Changes the bounds for a slice of the variables. Data checks are performed as in *Task.putvarbound*.

Parameters

- `first (int)` – First index in the sequence. (input)
- `last (int)` – Last index plus 1 in the sequence. (input)
- `bkg (mosek.boundkey)` – New bound key for all variables in the slice. (input)
- `blx (double)` – New lower bound for all variables in the slice. (input)
- `bux (double)` – New upper bound for all variables in the slice. (input)

Groups *Problem data - linear part, Problem data - variables, Problem data - bounds*

`Task.putvarname`

```
void putvarname
(int j,
 String name)
```

Sets the name of a variable.

Parameters

- `j` (`int`) – Index of the variable. (input)
- `name` (`String`) – The variable name. (input)

Groups *Names, Problem data - variables, Problem data - linear part*

`Task.putvarsolutionj`

```
void putvarsolutionj
(int j,
 mosek.soltype whichsol,
 mosek.stakey sk,
 double x,
 double sl,
 double su,
 double sn)
```

Sets the primal and dual solution information for a single variable.

Parameters

- `j` (`int`) – Index of the variable. (input)
- `whichsol` (*`mosek.soltype`*) – Selects a solution. (input)
- `sk` (*`mosek.stakey`*) – Status key of the variable. (input)
- `x` (`double`) – Primal solution value of the variable. (input)
- `sl` (`double`) – Solution value of the dual variable associated with the lower bound. (input)
- `su` (`double`) – Solution value of the dual variable associated with the upper bound. (input)
- `sn` (`double`) – Solution value of the dual variable associated with the conic constraint. (input)

Groups *Solution information, Solution - primal, Solution - dual*

`Task.putvartype`

```
void putvartype
(int j,
 mosek.variabletype vartype)
```

Sets the variable type of one variable.

Parameters

- `j` (`int`) – Index of the variable. (input)
- `vartype` (*`mosek.variabletype`*) – The new variable type. (input)

Groups *Problem data - variables*

`Task.putvartypelist`

```
void putvartypelist
(int[] subj,
 mosek.variabletype[] vartype)
```

Sets the variable type for one or more variables. If the same index is specified multiple times in `subj` only the last entry takes effect.

Parameters

- `subj` (`int[]`) – A list of variable indexes for which the variable type should be changed. (input)
- `vartype` (`mosek.variabletype[]`) – A list of variable types that should be assigned to the variables specified by `subj`. (input)

Groups *Problem data - variables*

`Task.putxc`

```
void putxc
(mosek.soltype whichsol,
 double[] xc)
```

Sets the x^c vector for a solution.

Parameters

- `whichsol` (`mosek.soltype`) – Selects a solution. (input)
- `xc` (`double[]`) – Primal constraint solution. (output)

Groups *Solution - primal*

`Task.putxcslice`

```
void putxcslice
(mosek.soltype whichsol,
 int first,
 int last,
 double[] xc)
```

Sets a slice of the x^c vector for a solution.

Parameters

- `whichsol` (`mosek.soltype`) – Selects a solution. (input)
- `first` (`int`) – First index in the sequence. (input)
- `last` (`int`) – Last index plus 1 in the sequence. (input)
- `xc` (`double[]`) – Primal constraint solution. (input)

Groups *Solution - primal*

`Task.putxx`

```
void putxx
(mosek.soltype whichsol,
 double[] xx)
```

Sets the x^x vector for a solution.

Parameters

- `whichsol` (`mosek.soltype`) – Selects a solution. (input)
- `xx` (`double[]`) – Primal variable solution. (input)

Groups *Solution - primal*

Task.putxxslice

```
void putxxslice
(mosek.soltype whichsol,
 int first,
 int last,
 double[] xx)
```

Sets a slice of the x^x vector for a solution.

Parameters

- `whichsol` (*mosek.soltype*) – Selects a solution. (input)
- `first` (int) – First index in the sequence. (input)
- `last` (int) – Last index plus 1 in the sequence. (input)
- `xx` (double[]) – Primal variable solution. (input)

Groups *Solution - primal*

Task.puty

```
void puty
(mosek.soltype whichsol,
 double[] y)
```

Sets the y vector for a solution.

Parameters

- `whichsol` (*mosek.soltype*) – Selects a solution. (input)
- `y` (double[]) – Vector of dual variables corresponding to the constraints. (input)

Groups *Solution - primal*

Task.putyslice

```
void putyslice
(mosek.soltype whichsol,
 int first,
 int last,
 double[] y)
```

Sets a slice of the y vector for a solution.

Parameters

- `whichsol` (*mosek.soltype*) – Selects a solution. (input)
- `first` (int) – First index in the sequence. (input)
- `last` (int) – Last index plus 1 in the sequence. (input)
- `y` (double[]) – Vector of dual variables corresponding to the constraints. (input)

Groups *Solution - dual*

Task.readdata

```
void readdata (String filename)
```

Reads an optimization problem and associated data from a file.

Parameters `filename` (String) – A valid file name. (input)

Groups *Input/Output*

Task.readdataformat

```
void readdataformat
(String filename,
 mosek.dataformat format,
 mosek.compresstype compress)
```

Reads an optimization problem and associated data from a file.

Parameters

- **filename** (String) – A valid file name. (input)
- **format** (*mosek.dataformat*) – File data format. (input)
- **compress** (*mosek.compresstype*) – File compression type. (input)

Groups *Input/Output*

Task.readjsonstring

```
void readjsonstring (String data)
```

Load task data from a JSON string, replacing any data that already exists in the task object. All problem data, parameters and other settings are resorted, but if the string contains solutions, the solution status after loading a file is set to unknown, even if it is optimal or otherwise well-defined.

Parameters **data** (String) – Problem data in text format. (input)

Groups *Input/Output*

Task.readlpstring

```
void readlpstring (String data)
```

Load task data from a string in LP format, replacing any data that already exists in the task object.

Parameters **data** (String) – Problem data in text format. (input)

Groups *Input/Output*

Task.readopfstring

```
void readopfstring (String data)
```

Load task data from a string in OPF format, replacing any data that already exists in the task object.

Parameters **data** (String) – Problem data in text format. (input)

Groups *Input/Output*

Task.readparamfile

```
void readparamfile (String filename)
```

Reads **MOSEK** parameters from a file. Data is read from the file **filename** if it is a nonempty string. Otherwise data is read from the file specified by *sparam.param_read_file_name*.

Parameters **filename** (String) – A valid file name. (input)

Groups *Input/Output, Parameters*

Task.readptfstring

```
void readptfstring (String data)
```

Load task data from a PTF string, replacing any data that already exists in the task object. All problem data, parameters and other settings are resorted, but if the string contains solutions, the solution status after loading a file is set to unknown, even if it is optimal or otherwise well-defined.

Parameters `data (String)` – Problem data in text format. (input)

Groups *Input/Output*

`Task.readsolution`

```
void readsolution  
(mosek.soltype whichsol,  
String filename)
```

Reads a solution file and inserts it as a specified solution in the task. Data is read from the file `filename` if it is a nonempty string. Otherwise data is read from one of the files specified by *sparam.bas_sol_file_name*, *sparam.itr_sol_file_name* or *sparam.int_sol_file_name* depending on which solution is chosen.

Parameters

- `whichsol (mosek.soltype)` – Selects a solution. (input)
- `filename (String)` – A valid file name. (input)

Groups *Input/Output*

`Task.readsummary`

```
void readsummary (mosek.streamtype whichstream)
```

Prints a short summary of last file that was read.

Parameters `whichstream (mosek.streamtype)` – Index of the stream. (input)

Groups *Input/Output, Inspecting the task*

`Task.readtask`

```
void readtask (String filename)
```

Load task data from a file, replacing any data that already exists in the task object. All problem data, parameters and other settings are resorted, but if the file contains solutions, the solution status after loading a file is set to unknown, even if it was optimal or otherwise well-defined when the file was dumped.

See section *The Task Format* for a description of the Task format.

Parameters `filename (String)` – A valid file name. (input)

Groups *Input/Output*

`Task.removebarvars`

```
void removebarvars (int[] subset)
```

The function removes a subset of the symmetric matrices from the optimization task. This implies that the remaining symmetric matrices are renumbered.

Parameters `subset (int[])` – Indexes of symmetric matrices which should be removed. (input)

Groups *Problem data - semidefinite*

Task.removecones

```
void removecones (int[] subset)
```

Removes a number of conic constraints from the problem. This implies that the remaining conic constraints are renumbered. In general, it is much more efficient to remove a cone with a high index than a low index.

Parameters `subset (int[])` – Indexes of cones which should be removed. (input)

Groups *Problem data - cones*

Task.removecons

```
void removecons (int[] subset)
```

The function removes a subset of the constraints from the optimization task. This implies that the remaining constraints are renumbered.

Parameters `subset (int[])` – Indexes of constraints which should be removed. (input)

Groups *Problem data - constraints, Problem data - linear part*

Task.removevars

```
void removevars (int[] subset)
```

The function removes a subset of the variables from the optimization task. This implies that the remaining variables are renumbered.

Parameters `subset (int[])` – Indexes of variables which should be removed. (input)

Groups *Problem data - variables, Problem data - linear part*

Task.resizetask

```
void resizetask  
(int maxnumcon,  
 int maxnumvar,  
 int maxnumcone,  
 long maxnumanz,  
 long maxnumqnz)
```

Sets the amount of preallocated space assigned for each type of data in an optimization task.

It is never mandatory to call this function, since it only gives a hint about the amount of data to preallocate for efficiency reasons.

Please note that the procedure is **destructive** in the sense that all existing data stored in the task is destroyed.

Parameters

- `maxnumcon (int)` – New maximum number of constraints. (input)
- `maxnumvar (int)` – New maximum number of variables. (input)
- `maxnumcone (int)` – New maximum number of cones. (input)
- `maxnumanz (long)` – New maximum number of non-zeros in A . (input)
- `maxnumqnz (long)` – New maximum number of non-zeros in all Q matrices. (input)

Groups *Environment and task management*

Task.sensitivityreport

```
void sensitivityreport (mosek.streamtype whichstream)
```

Reads a sensitivity format file from a location given by `sparam.sensitivity_file_name` and writes the result to the stream `whichstream`. If `sparam.sensitivity_res_file_name` is set to a non-empty string, then the sensitivity report is also written to a file of this name.

Parameters `whichstream` (`mosek.streamtype`) – Index of the stream. (input)

Groups `Sensitivity analysis`

`Task.set_InfoCallback`

```
void set_InfoCallback (mosek.DataCallback callback)
```

Receive callbacks with solver status and information during optimization.

For example:

```
task.set_InfoCallback(
    new mosek.InfoCallback() {
        int callback(mosek.callbackcode code, double[] dinf, int[] iinf, long[] liinf) {
            System.println("Callback "+code+", intpnt time : "+dinf[mosek.dinfitem.
            ↳intpnt_time.getValue()]);
            return 0;
        } } );
```

Parameters `callback` (`DataCallback`) – The callback object. (input)

`Task.set_ItgSolutionCallback`

```
void set_ItgSolutionCallback (mosek.ItgSolutionCallback callback)
```

Receive callbacks with solution updates from the mixed-integer optimizer.

For example:

```
task.set_ItgSolutionCallback(
    new mosek.ItgSolutionCallback() {
        void callback(double[] xx) {
            System.out.print("New integer solution: ");
            for (double v : xx) System.out.print(" " + v + " ");
            System.out.println("");
        } } );
```

Parameters `callback` (`ItgSolutionCallback`) – The callback object. (input)

`Task.set_Progress`

```
void set_Progress (mosek.Progress callback)
```

Receive callbacks about current status of the solver during optimization.

For example:

```
task.set_Progress(new mosek.Progress() { int progress(mosek.callbackcode code) {
    ↳System.println("Callback "+code); return 0; } } );
```

Parameters `callback` (`Progress`) – The callback object. (input)

Task.set_Stream

```
void set_Stream(  
    mosek.streamtype whichstream,  
    mosek.Stream callback)
```

Directs all output from a task stream to a callback object.

Can for example be called as:

```
task.set_Stream(mosek.streamtype.log, new Stream() { public void stream(String_  
→s) { System.out.print(s); } } );
```

Parameters

- whichstream (*streamtype*) – Index of the stream. (input)
- callback (*Stream*) – The callback object. (input)

Task.setdefault

```
void setdefaults ()
```

Resets all the parameters to their default values.

Groups *Parameters*

Task.solutiondef

```
void solutiondef  
(mosek.soltype whichsol,  
    boolean[] isdef)
```

```
boolean solutiondef (mosek.soltype whichsol)
```

Checks whether a solution is defined.

Parameters

- whichsol (*mosek.soltype*) – Selects a solution. (input)
- isdef (boolean *by reference*) – Is non-zero if the requested solution is defined. (output)

Return (boolean) – Is non-zero if the requested solution is defined.

Groups *Solution information*

Task.solutionsummary

```
void solutionsummary (mosek.streamtype whichstream)
```

Prints a short summary of the current solutions.

Parameters whichstream (*mosek.streamtype*) – Index of the stream. (input)

Groups *Logging, Solution information*

Task.solvewithbasis

```
void solvewithbasis  
(int transp,  
    int[] numnz,  
    int[] sub,  
    double[] val)
```

```

int solvewithbasis
(int transp,
 int numnz,
 int[] sub,
 double[] val)

```

If a basic solution is available, then exactly *numcon* basis variables are defined. These *numcon* basis variables are denoted the basis. Associated with the basis is a basis matrix denoted B . This function solves either the linear equation system

$$B\overline{X} = b \quad (15.3)$$

or the system

$$B^T\overline{X} = b \quad (15.4)$$

for the unknowns \overline{X} , with b being a user-defined vector. In order to make sense of the solution \overline{X} it is important to know the ordering of the variables in the basis because the ordering specifies how B is constructed. When calling `Task.initbasissolve` an ordering of the basis variables is obtained, which can be used to deduce how **MOSEK** has constructed B . Indeed if the k -th basis variable is variable x_j it implies that

$$B_{i,k} = A_{i,j}, \quad i = 0, \dots, \text{numcon} - 1.$$

Otherwise if the k -th basis variable is variable x_j^c it implies that

$$B_{i,k} = \begin{cases} -1, & i = j, \\ 0, & i \neq j. \end{cases}$$

The function `Task.initbasissolve` must be called before a call to this function. Please note that this function exploits the sparsity in the vector b to speed up the computations.

Parameters

- **transp** (int) – If this argument is zero, then (15.3) is solved, if non-zero then (15.4) is solved. (input)
- **numnz** (int *by reference*) – As input it is the number of non-zeros in b . As output it is the number of non-zeros in \overline{X} . (input/output)
- **numnz** (int) – As input it is the number of non-zeros in b . As output it is the number of non-zeros in \overline{X} . (input/output)
- **sub** (int[]) – As input it contains the positions of non-zeros in b . As output it contains the positions of the non-zeros in \overline{X} . It must have room for *numcon* elements. (input/output)
- **val** (double[]) – As input it is the vector b as a dense vector (although the positions of non-zeros are specified in **sub** it is required that $\text{val}[i] = 0$ when $b[i] = 0$). As output **val** is the vector \overline{X} as a dense vector. It must have length *numcon*. (input/output)

Return (int) – As input it is the number of non-zeros in b . As output it is the number of non-zeros in \overline{X} .

Groups *Solving systems with basis matrix*

`Task.strtoconetype`

```

void strtoconetype
(String str,
 mosek.conetype[] conetype)

```

Obtains cone type code corresponding to a cone type string.

Parameters

- **str** (**String**) – String corresponding to the cone type code **conetype**. (input)
- **conetype** (*mosek.conetype by reference*) – The cone type corresponding to the string **str**. (output)

Groups *Names*

Task.strtosk

```
void strtosk
(String str,
 mosek.stakey[] sk)
```

Obtains the status key corresponding to an abbreviation string.

Parameters

- **str** (**String**) – A status key abbreviation string. (input)
- **sk** (*mosek.stakey by reference*) – Status key corresponding to the string. (output)

Groups *Names*

Task.toconic

```
void toconic ()
```

This function tries to reformulate a given Quadratically Constrained Quadratic Optimization problem (QCQP) as a Conic Quadratic Optimization problem (CQO). The first step of the reformulation is to convert the quadratic term of the objective function, if any, into a constraint. Then the following steps are repeated for each quadratic constraint:

- a conic constraint is added along with a suitable number of auxiliary variables and constraints;
- the original quadratic constraint is not removed, but all its coefficients are zeroed out.

Note that the reformulation preserves all the original variables.

The conversion is performed in-place, i.e. the task passed as argument is modified on exit. That also means that if the reformulation fails, i.e. the given QCQP is not representable as a CQO, then the task has an undefined state. In some cases, users may want to clone the task to ensure a clean copy is preserved.

Groups *Problem data - quadratic part*

Task.unset_Progress

```
void unset_Progress ()
```

Deactivates all user callback functions.

Task.updatesolutioninfo

```
void updatesolutioninfo (mosek.soltype whichsol)
```

Update the information items related to the solution.

Parameters **whichsol** (*mosek.soltype*) – Selects a solution. (input)

Groups *Information items and statistics*

Task.writedata

```
void writedata (String filename)
```

Writes problem data associated with the optimization task to a file in one of the supported formats. See Section *Supported File Formats* for the complete list.

The data file format is determined by the file name extension. To write in compressed format append the extension `.gz`. E.g to write a gzip compressed MPS file use the extension `mps.gz`.

Please note that MPS, LP and OPF files require all variables to have unique names. If a task contains no names, it is possible to write the file with automatically generated anonymous names by setting the `iparam.write_generic_names` parameter to `onoffkey.on`.

Data is written to the file `filename` if it is a nonempty string. Otherwise data is written to the file specified by `sparam.data_file_name`.

Parameters `filename` (String) – A valid file name. (input)

Groups *Input/Output*

Task.writejsonsol

```
void writejsonsol (String filename)
```

Saves the current solutions and solver information items in a JSON file.

Parameters `filename` (String) – A valid file name. (input)

Groups *Input/Output*

Task.writeparamfile

```
void writeparamfile (String filename)
```

Writes all the parameters to a parameter file.

Parameters `filename` (String) – A valid file name. (input)

Groups *Input/Output, Parameters*

Task.writesolution

```
void writesolution  
(mosek.soltype whichsol,  
String filename)
```

Saves the current basic, interior-point, or integer solution to a file.

Parameters

- `whichsol` (*mosek.soltype*) – Selects a solution. (input)
- `filename` (String) – A valid file name. (input)

Groups *Input/Output*

Task.writetask

```
void writetask (String filename)
```

Write a binary dump of the task data. This format saves all problem data, coefficients and parameter settings. See section *The Task Format* for a description of the Task format.

Parameters `filename` (String) – A valid file name. (input)

Groups *Input/Output*

15.5 Exceptions

MosekException

The base class for all exceptions in **MOSEK**.

Exception

Base class for exceptions that correspond to **MOSEK** response codes.

Implements *MosekException*

Error

Exception class used for all error response codes from **MOSEK**.

Implements *Exception*

Warning

Exception class used for all warning response codes from **MOSEK**.

Implements *Exception*

NullArrayException

Exception thrown when null was passed to a method that expected non-null array argument.

Implements *MosekException*

ArrayLengthException

Exception thrown the length of an array was smaller than required. This will happen, for example, if requesting a list of N values, but the array passed to the method is less than N elements long.

Implements *MosekException*

15.6 Parameters grouped by topic

Analysis

- *dparam.ana_sol_infeas_tol*
- *iparam.ana_sol_basis*
- *iparam.ana_sol_print_violated*
- *iparam.log_ana_pro*

Basis identification

- *dparam.sim_lu_tol_rel_piv*
- *iparam.bi_clean_optimizer*
- *iparam.bi_ignore_max_iter*
- *iparam.bi_ignore_num_error*
- *iparam.bi_max_iterations*
- *iparam.intpnt_basis*
- *iparam.log_bi*
- *iparam.log_bi_freq*

Conic interior-point method

- *dparam.intpnt_co_tol_dfeas*
- *dparam.intpnt_co_tol_infeas*
- *dparam.intpnt_co_tol_mu_red*
- *dparam.intpnt_co_tol_near_rel*
- *dparam.intpnt_co_tol_pfeas*
- *dparam.intpnt_co_tol_rel_gap*

Data check

- *dparam.data_sym_mat_tol*
- *dparam.data_sym_mat_tol_huge*
- *dparam.data_sym_mat_tol_large*
- *dparam.data_tol_aij_huge*
- *dparam.data_tol_aij_large*
- *dparam.data_tol_bound_inf*
- *dparam.data_tol_bound_wrn*
- *dparam.data_tol_c_huge*
- *dparam.data_tol_cj_large*
- *dparam.data_tol_qij*
- *dparam.data_tol_x*
- *dparam.semidefinite_tol_approx*
- *iparam.check_convexity*
- *iparam.log_check_convexity*

Data input/output

- *iparam.infeas_report_auto*
- *iparam.log_file*
- *iparam.opf_write_header*
- *iparam.opf_write_hints*
- *iparam.opf_write_line_length*
- *iparam.opf_write_parameters*
- *iparam.opf_write_problem*
- *iparam.opf_write_sol_bas*
- *iparam.opf_write_sol_itg*
- *iparam.opf_write_sol_itr*
- *iparam.opf_write_solutions*

- *iparam.param_read_case_name*
- *iparam.param_read_ign_error*
- *iparam.ptf_write_transform*
- *iparam.read_debug*
- *iparam.read_keep_free_con*
- *iparam.read_lp_drop_new_vars_in_bou*
- *iparam.read_lp_quoted_names*
- *iparam.read_mps_format*
- *iparam.read_mps_width*
- *iparam.read_task_ignore_param*
- *iparam.sol_read_name_width*
- *iparam.sol_read_width*
- *iparam.write_bas_constraints*
- *iparam.write_bas_head*
- *iparam.write_bas_variables*
- *iparam.write_compression*
- *iparam.write_data_param*
- *iparam.write_free_con*
- *iparam.write_generic_names*
- *iparam.write_generic_names_io*
- *iparam.write_ignore_incompatible_items*
- *iparam.write_int_constraints*
- *iparam.write_int_head*
- *iparam.write_int_variables*
- *iparam.write_lp_full_obj*
- *iparam.write_lp_line_width*
- *iparam.write_lp_quoted_names*
- *iparam.write_lp_strict_format*
- *iparam.write_lp_terms_per_line*
- *iparam.write_mps_format*
- *iparam.write_mps_int*
- *iparam.write_precision*
- *iparam.write_sol_barvariables*
- *iparam.write_sol_constraints*
- *iparam.write_sol_head*
- *iparam.write_sol_ignore_invalid_names*

- *iparam.write_sol_variables*
- *iparam.write_task_inc_sol*
- *iparam.write_xml_mode*
- *sparam.bas_sol_file_name*
- *sparam.data_file_name*
- *sparam.debug_file_name*
- *sparam.int_sol_file_name*
- *sparam.itr_sol_file_name*
- *sparam.mio_debug_string*
- *sparam.param_comment_sign*
- *sparam.param_read_file_name*
- *sparam.param_write_file_name*
- *sparam.read_mps_bou_name*
- *sparam.read_mps_obj_name*
- *sparam.read_mps_ran_name*
- *sparam.read_mps_rhs_name*
- *sparam.sensitivity_file_name*
- *sparam.sensitivity_res_file_name*
- *sparam.sol_filter_xc_low*
- *sparam.sol_filter_xc_upr*
- *sparam.sol_filter_xx_low*
- *sparam.sol_filter_xx_upr*
- *sparam.stat_file_name*
- *sparam.stat_key*
- *sparam.stat_name*
- *sparam.write_lp_gen_var_name*

Debugging

- *iparam.auto_sort_a_before_opt*

Dual simplex

- *iparam.sim_dual_crash*
- *iparam.sim_dual_restrict_selection*
- *iparam.sim_dual_selection*

Infeasibility report

- *iparam.infeas_generic_names*
- *iparam.infeas_report_level*
- *iparam.log_infeas_ana*

Interior-point method

- *dparam.check_convexity_rel_tol*
- *dparam.intpnt_co_tol_dfeas*
- *dparam.intpnt_co_tol_infeas*
- *dparam.intpnt_co_tol_mu_red*
- *dparam.intpnt_co_tol_near_rel*
- *dparam.intpnt_co_tol_pfeas*
- *dparam.intpnt_co_tol_rel_gap*
- *dparam.intpnt_qo_tol_dfeas*
- *dparam.intpnt_qo_tol_infeas*
- *dparam.intpnt_qo_tol_mu_red*
- *dparam.intpnt_qo_tol_near_rel*
- *dparam.intpnt_qo_tol_pfeas*
- *dparam.intpnt_qo_tol_rel_gap*
- *dparam.intpnt_tol_dfeas*
- *dparam.intpnt_tol_dsafe*
- *dparam.intpnt_tol_infeas*
- *dparam.intpnt_tol_mu_red*
- *dparam.intpnt_tol_path*
- *dparam.intpnt_tol_pfeas*
- *dparam.intpnt_tol_psafe*
- *dparam.intpnt_tol_rel_gap*
- *dparam.intpnt_tol_rel_step*
- *dparam.intpnt_tol_step_size*
- *dparam.qcgo_reformulate_rel_drop_tol*
- *iparam.bi_ignore_max_iter*
- *iparam.bi_ignore_num_error*
- *iparam.intpnt_basis*
- *iparam.intpnt_diff_step*
- *iparam.intpnt_hotstart*
- *iparam.intpnt_max_iterations*

- *iparam.intpnt_max_num_cor*
- *iparam.intpnt_max_num_refinement_steps*
- *iparam.intpnt_off_col_trh*
- *iparam.intpnt_order_gp_num_seeds*
- *iparam.intpnt_order_method*
- *iparam.intpnt_purify*
- *iparam.intpnt_regularization_use*
- *iparam.intpnt_scaling*
- *iparam.intpnt_solve_form*
- *iparam.intpnt_starting_point*
- *iparam.log_intpnt*

License manager

- *iparam.cache_license*
- *iparam.license_debug*
- *iparam.license_pause_time*
- *iparam.license_suppress_expire_wrns*
- *iparam.license_trh_expiry_wrn*
- *iparam.license_wait*

Logging

- *iparam.log*
- *iparam.log_ana_pro*
- *iparam.log_bi*
- *iparam.log_bi_freq*
- *iparam.log_cut_second_opt*
- *iparam.log_expand*
- *iparam.log_feas_repair*
- *iparam.log_file*
- *iparam.log_include_summary*
- *iparam.log_infeas_ana*
- *iparam.log_intpnt*
- *iparam.log_local_info*
- *iparam.log_mio*
- *iparam.log_mio_freq*
- *iparam.log_order*

- *iparam.log_presolve*
- *iparam.log_response*
- *iparam.log_sensitivity*
- *iparam.log_sensitivity_opt*
- *iparam.log_sim*
- *iparam.log_sim_freq*
- *iparam.log_storage*

Mixed-integer optimization

- *dparam.mio_max_time*
- *dparam.mio_rel_gap_const*
- *dparam.mio_tol_abs_gap*
- *dparam.mio_tol_abs_relax_int*
- *dparam.mio_tol_feas*
- *dparam.mio_tol_rel_dual_bound_improvement*
- *dparam.mio_tol_rel_gap*
- *iparam.log_mio*
- *iparam.log_mio_freq*
- *iparam.mio_branch_dir*
- *iparam.mio_conic_outer_approximation*
- *iparam.mio_cut_clique*
- *iparam.mio_cut_cmir*
- *iparam.mio_cut_gmi*
- *iparam.mio_cut_implied_bound*
- *iparam.mio_cut_knapsack_cover*
- *iparam.mio_cut_selection_level*
- *iparam.mio_feaspump_level*
- *iparam.mio_heuristic_level*
- *iparam.mio_max_num_branches*
- *iparam.mio_max_num_relaxs*
- *iparam.mio_max_num_root_cut_rounds*
- *iparam.mio_max_num_solutions*
- *iparam.mio_node_optimizer*
- *iparam.mio_node_selection*
- *iparam.mio_perspective_reformulate*
- *iparam.mio_probing_level*

- *iparam.mio_propagate_objective_constraint*
- *iparam.mio_rins_max_nodes*
- *iparam.mio_root_optimizer*
- *iparam.mio_root_repeat_presolve_level*
- *iparam.mio_seed*
- *iparam.mio_vb_detection_level*

Output information

- *iparam.infeas_report_level*
- *iparam.license_suppress_expire_wrns*
- *iparam.license_trh_expiry_wrn*
- *iparam.log*
- *iparam.log_bi*
- *iparam.log_bi_freq*
- *iparam.log_cut_second_opt*
- *iparam.log_expand*
- *iparam.log_feas_repair*
- *iparam.log_file*
- *iparam.log_include_summary*
- *iparam.log_infeas_ana*
- *iparam.log_intpnt*
- *iparam.log_local_info*
- *iparam.log_mio*
- *iparam.log_mio_freq*
- *iparam.log_order*
- *iparam.log_response*
- *iparam.log_sensitivity*
- *iparam.log_sensitivity_opt*
- *iparam.log_sim*
- *iparam.log_sim_freq*
- *iparam.log_sim_minor*
- *iparam.log_storage*
- *iparam.max_num_warnings*

Overall solver

- *iparam.bi_clean_optimizer*
- *iparam.infeas_prefer_primal*
- *iparam.license_wait*
- *iparam.mio_mode*
- *iparam.optimizer*
- *iparam.presolve_level*
- *iparam.presolve_max_num_reductions*
- *iparam.presolve_use*
- *iparam.primal_repair_optimizer*
- *iparam.sensitivity_all*
- *iparam.sensitivity_optimizer*
- *iparam.sensitivity_type*
- *iparam.solution_callback*

Overall system

- *iparam.auto_update_sol_info*
- *iparam.intpnt_multi_thread*
- *iparam.license_wait*
- *iparam.log_storage*
- *iparam.mt_spincount*
- *iparam.num_threads*
- *iparam.remove_unused_solutions*
- *iparam.timing_level*
- *sparam.remote_access_token*

Presolve

- *dparam.presolve_tol_abs_lindp*
- *dparam.presolve_tol_aij*
- *dparam.presolve_tol_rel_lindp*
- *dparam.presolve_tol_s*
- *dparam.presolve_tol_x*
- *iparam.presolve_eliminator_max_fill*
- *iparam.presolve_eliminator_max_num_tries*
- *iparam.presolve_level*
- *iparam.presolve_lindp_abs_work_trh*

- *iparam.presolve_lindep_rel_work_trh*
- *iparam.presolve_lindep_use*
- *iparam.presolve_max_num_pass*
- *iparam.presolve_max_num_reductions*
- *iparam.presolve_use*

Primal simplex

- *iparam.sim_primal_crash*
- *iparam.sim_primal_restrict_selection*
- *iparam.sim_primal_selection*

Progress callback

- *iparam.solution_callback*

Simplex optimizer

- *dparam.basis_rel_tol_s*
- *dparam.basis_tol_s*
- *dparam.basis_tol_x*
- *dparam.sim_lu_tol_rel_piv*
- *dparam.simplex_abs_tol_piv*
- *iparam.basis_solve_use_plus_one*
- *iparam.log_sim*
- *iparam.log_sim_freq*
- *iparam.log_sim_minor*
- *iparam.sensitivity_optimizer*
- *iparam.sim_basis_factor_use*
- *iparam.sim_degen*
- *iparam.sim_dual_phaseone_method*
- *iparam.sim_exploit_dupvec*
- *iparam.sim_hotstart*
- *iparam.sim_hotstart_lu*
- *iparam.sim_max_iterations*
- *iparam.sim_max_num_setbacks*
- *iparam.sim_non_singular*
- *iparam.sim_primal_phaseone_method*
- *iparam.sim_refactor_freq*

- *iparam.sim_reformulation*
- *iparam.sim_save_lu*
- *iparam.sim_scaling*
- *iparam.sim_scaling_method*
- *iparam.sim_seed*
- *iparam.sim_solve_form*
- *iparam.sim_stability_priority*
- *iparam.sim_switch_optimizer*

Solution input/output

- *iparam.infeas_report_auto*
- *iparam.sol_filter_keep_basic*
- *iparam.sol_filter_keep_ranged*
- *iparam.sol_read_name_width*
- *iparam.sol_read_width*
- *iparam.write_bas_constraints*
- *iparam.write_bas_head*
- *iparam.write_bas_variables*
- *iparam.write_int_constraints*
- *iparam.write_int_head*
- *iparam.write_int_variables*
- *iparam.write_sol_barvariables*
- *iparam.write_sol_constraints*
- *iparam.write_sol_head*
- *iparam.write_sol_ignore_invalid_names*
- *iparam.write_sol_variables*
- *sparam.bas_sol_file_name*
- *sparam.int_sol_file_name*
- *sparam.itr_sol_file_name*
- *sparam.sol_filter_xc_low*
- *sparam.sol_filter_xc_upr*
- *sparam.sol_filter_xx_low*
- *sparam.sol_filter_xx_upr*

Termination criteria

- *dparam.basis_rel_tol_s*
- *dparam.basis_tol_s*
- *dparam.basis_tol_x*
- *dparam.intpnt_co_tol_dfeas*
- *dparam.intpnt_co_tol_infeas*
- *dparam.intpnt_co_tol_mu_red*
- *dparam.intpnt_co_tol_near_rel*
- *dparam.intpnt_co_tol_pfeas*
- *dparam.intpnt_co_tol_rel_gap*
- *dparam.intpnt_qo_tol_dfeas*
- *dparam.intpnt_qo_tol_infeas*
- *dparam.intpnt_qo_tol_mu_red*
- *dparam.intpnt_qo_tol_near_rel*
- *dparam.intpnt_qo_tol_pfeas*
- *dparam.intpnt_qo_tol_rel_gap*
- *dparam.intpnt_tol_dfeas*
- *dparam.intpnt_tol_infeas*
- *dparam.intpnt_tol_mu_red*
- *dparam.intpnt_tol_pfeas*
- *dparam.intpnt_tol_rel_gap*
- *dparam.lower_obj_cut*
- *dparam.lower_obj_cut_finite_trh*
- *dparam.mio_max_time*
- *dparam.mio_rel_gap_const*
- *dparam.mio_tol_rel_gap*
- *dparam.optimizer_max_time*
- *dparam.upper_obj_cut*
- *dparam.upper_obj_cut_finite_trh*
- *iparam.bi_max_iterations*
- *iparam.intpnt_max_iterations*
- *iparam.mio_max_num_branches*
- *iparam.mio_max_num_root_cut_rounds*
- *iparam.mio_max_num_solutions*
- *iparam.sim_max_iterations*

Other

- *iparam.compress_statfile*

15.7 Parameters (alphabetical list sorted by type)

- *Double parameters*
- *Integer parameters*
- *String parameters*

15.7.1 Double parameters

dparam

The enumeration type containing all double parameters.

dparam.ana_sol_infeas_tol

If a constraint violates its bound with an amount larger than this value, the constraint name, index and violation will be printed by the solution analyzer.

Default 1e-6

Accepted [0.0; +inf]

Example `task.putdouparam(dparam.ana_sol_infeas_tol, 1e-6)`

Generic name MSK_DPAR_ANA_SOL_INFEAS_TOL

Groups *Analysis*

dparam.basis_rel_tol_s

Maximum relative dual bound violation allowed in an optimal basic solution.

Default 1.0e-12

Accepted [0.0; +inf]

Example `task.putdouparam(dparam.basis_rel_tol_s, 1.0e-12)`

Generic name MSK_DPAR_BASIS_REL_TOL_S

Groups *Simplex optimizer, Termination criteria*

dparam.basis_tol_s

Maximum absolute dual bound violation in an optimal basic solution.

Default 1.0e-6

Accepted [1.0e-9; +inf]

Example `task.putdouparam(dparam.basis_tol_s, 1.0e-6)`

Generic name MSK_DPAR_BASIS_TOL_S

Groups *Simplex optimizer, Termination criteria*

dparam.basis_tol_x

Maximum absolute primal bound violation allowed in an optimal basic solution.

Default 1.0e-6

Accepted [1.0e-9; +inf]

Example `task.putdouparam(dparam.basis_tol_x, 1.0e-6)`

Generic name MSK_DPAR_BASIS_TOL_X

Groups *Simplex optimizer, Termination criteria*

dparam.check_convexity_rel_tol

This parameter controls when the full convexity check declares a problem to be non-convex. Increasing this tolerance relaxes the criteria for declaring the problem non-convex.

A problem is declared non-convex if negative (positive) pivot elements are detected in the Cholesky factor of a matrix which is required to be PSD (NSD). This parameter controls how much this non-negativity requirement may be violated.

If d_i is the pivot element for column i , then the matrix Q is considered to not be PSD if:

$$d_i \leq -|Q_{ii}| \text{check_convexity_rel_tol}$$

Default 1e-10

Accepted [0; +inf]

Example task.putdoupparam(dparam.check_convexity_rel_tol, 1e-10)

Generic name MSK_DPAR_CHECK_CONVEXITY_REL_TOL

Groups *Interior-point method*

dparam.data_sym_mat_tol

Absolute zero tolerance for elements in symmetric matrices. If any value in a symmetric matrix is smaller than this parameter in absolute terms **MOSEK** will treat the values as zero and generate a warning.

Default 1.0e-12

Accepted [1.0e-16; 1.0e-6]

Example task.putdoupparam(dparam.data_sym_mat_tol, 1.0e-12)

Generic name MSK_DPAR_DATA_SYM_MAT_TOL

Groups *Data check*

dparam.data_sym_mat_tol_huge

An element in a symmetric matrix which is larger than this value in absolute size causes an error.

Default 1.0e20

Accepted [0.0; +inf]

Example task.putdoupparam(dparam.data_sym_mat_tol_huge, 1.0e20)

Generic name MSK_DPAR_DATA_SYM_MAT_TOL_HUGE

Groups *Data check*

dparam.data_sym_mat_tol_large

An element in a symmetric matrix which is larger than this value in absolute size causes a warning message to be printed.

Default 1.0e10

Accepted [0.0; +inf]

Example task.putdoupparam(dparam.data_sym_mat_tol_large, 1.0e10)

Generic name MSK_DPAR_DATA_SYM_MAT_TOL_LARGE

Groups *Data check*

dparam.data_tol_aij_huge

An element in A which is larger than this value in absolute size causes an error.

Default 1.0e20

Accepted [0.0; +inf]

Example task.putdoupparam(dparam.data_tol_aij_huge, 1.0e20)

Generic name MSK_DPAR_DATA_TOL_AIJ_HUGE

Groups *Data check*

dparam.data_tol_aij_large

An element in A which is larger than this value in absolute size causes a warning message to be printed.

Default 1.0e10

Accepted [0.0; +inf]

Example task.putdoupparam(dparam.data_tol_aij_large, 1.0e10)

Generic name MSK_DPAR_DATA_TOL_AIJ_LARGE

Groups *Data check*

`dparam.data_tol_bound_inf`

Any bound which in absolute value is greater than this parameter is considered infinite.

Default 1.0e16

Accepted [0.0; +inf]

Example `task.putdoupparam(dparam.data_tol_bound_inf, 1.0e16)`

Generic name MSK_DPAR_DATA_TOL_BOUND_INF

Groups *Data check*

`dparam.data_tol_bound_wrn`

If a bound value is larger than this value in absolute size, then a warning message is issued.

Default 1.0e8

Accepted [0.0; +inf]

Example `task.putdoupparam(dparam.data_tol_bound_wrn, 1.0e8)`

Generic name MSK_DPAR_DATA_TOL_BOUND_WRN

Groups *Data check*

`dparam.data_tol_c_huge`

An element in c which is larger than the value of this parameter in absolute terms is considered to be huge and generates an error.

Default 1.0e16

Accepted [0.0; +inf]

Example `task.putdoupparam(dparam.data_tol_c_huge, 1.0e16)`

Generic name MSK_DPAR_DATA_TOL_C_HUGE

Groups *Data check*

`dparam.data_tol_cj_large`

An element in c which is larger than this value in absolute terms causes a warning message to be printed.

Default 1.0e8

Accepted [0.0; +inf]

Example `task.putdoupparam(dparam.data_tol_cj_large, 1.0e8)`

Generic name MSK_DPAR_DATA_TOL_CJ_LARGE

Groups *Data check*

`dparam.data_tol_qij`

Absolute zero tolerance for elements in Q matrices.

Default 1.0e-16

Accepted [0.0; +inf]

Example `task.putdoupparam(dparam.data_tol_qij, 1.0e-16)`

Generic name MSK_DPAR_DATA_TOL_QIJ

Groups *Data check*

`dparam.data_tol_x`

Zero tolerance for constraints and variables i.e. if the distance between the lower and upper bound is less than this value, then the lower and upper bound is considered identical.

Default 1.0e-8

Accepted [0.0; +inf]

Example `task.putdoupparam(dparam.data_tol_x, 1.0e-8)`

Generic name MSK_DPAR_DATA_TOL_X

Groups *Data check*

`dparam.intpnt_co_tol_dfeas`

Dual feasibility tolerance used by the interior-point optimizer for conic problems.

Default 1.0e-8
Accepted [0.0; 1.0]
Example `task.putdoupparam(dparam.intpnt_co_tol_dfeas, 1.0e-8)`
See also [dparam.intpnt_co_tol_near_rel](#)
Generic name MSK_DPAR_INTPNT_CO_TOL_DFEAS
Groups *Interior-point method, Termination criteria, Conic interior-point method*

`dparam.intpnt_co_tol_infeas`

Infeasibility tolerance used by the interior-point optimizer for conic problems. Controls when the interior-point optimizer declares the model primal or dual infeasible. A small number means the optimizer gets more conservative about declaring the model infeasible.

Default 1.0e-12
Accepted [0.0; 1.0]
Example `task.putdoupparam(dparam.intpnt_co_tol_infeas, 1.0e-12)`
Generic name MSK_DPAR_INTPNT_CO_TOL_INFEAS
Groups *Interior-point method, Termination criteria, Conic interior-point method*

`dparam.intpnt_co_tol_mu_red`

Relative complementarity gap tolerance used by the interior-point optimizer for conic problems.

Default 1.0e-8
Accepted [0.0; 1.0]
Example `task.putdoupparam(dparam.intpnt_co_tol_mu_red, 1.0e-8)`
Generic name MSK_DPAR_INTPNT_CO_TOL_MU_RED
Groups *Interior-point method, Termination criteria, Conic interior-point method*

`dparam.intpnt_co_tol_near_rel`

Optimality tolerance used by the interior-point optimizer for conic problems. If **MOSEK** cannot compute a solution that has the prescribed accuracy then it will check if the solution found satisfies the termination criteria with all tolerances multiplied by the value of this parameter. If yes, then the solution is also declared optimal.

Default 1000
Accepted [1.0; +inf]
Example `task.putdoupparam(dparam.intpnt_co_tol_near_rel, 1000)`
Generic name MSK_DPAR_INTPNT_CO_TOL_NEAR_REL
Groups *Interior-point method, Termination criteria, Conic interior-point method*

`dparam.intpnt_co_tol_pfeas`

Primal feasibility tolerance used by the interior-point optimizer for conic problems.

Default 1.0e-8
Accepted [0.0; 1.0]
Example `task.putdoupparam(dparam.intpnt_co_tol_pfeas, 1.0e-8)`
See also [dparam.intpnt_co_tol_near_rel](#)
Generic name MSK_DPAR_INTPNT_CO_TOL_PFEAS
Groups *Interior-point method, Termination criteria, Conic interior-point method*

`dparam.intpnt_co_tol_rel_gap`

Relative gap termination tolerance used by the interior-point optimizer for conic problems.

Default 1.0e-8
Accepted [0.0; 1.0]
Example `task.putdoupparam(dparam.intpnt_co_tol_rel_gap, 1.0e-8)`
See also [dparam.intpnt_co_tol_near_rel](#)
Generic name MSK_DPAR_INTPNT_CO_TOL_REL_GAP
Groups *Interior-point method, Termination criteria, Conic interior-point method*

`dparam.intpnt_qo_tol_dfeas`

Dual feasibility tolerance used by the interior-point optimizer for quadratic problems.

Default 1.0e-8

Accepted [0.0; 1.0]

Example `task.putdoupam(dparam.intpnt_qo_tol_dfeas, 1.0e-8)`

See also [*dparam.intpnt_qo_tol_near_rel*](#)

Generic name MSK_DPAR_INTPNT_QO_TOL_DFEAS

Groups [*Interior-point method*](#), [*Termination criteria*](#)

`dparam.intpnt_qo_tol_infeas`

Infeasibility tolerance used by the interior-point optimizer for quadratic problems. Controls when the interior-point optimizer declares the model primal or dual infeasible. A small number means the optimizer gets more conservative about declaring the model infeasible.

Default 1.0e-12

Accepted [0.0; 1.0]

Example `task.putdoupam(dparam.intpnt_qo_tol_infeas, 1.0e-12)`

Generic name MSK_DPAR_INTPNT_QO_TOL_INFEAS

Groups [*Interior-point method*](#), [*Termination criteria*](#)

`dparam.intpnt_qo_tol_mu_red`

Relative complementarity gap tolerance used by the interior-point optimizer for quadratic problems.

Default 1.0e-8

Accepted [0.0; 1.0]

Example `task.putdoupam(dparam.intpnt_qo_tol_mu_red, 1.0e-8)`

Generic name MSK_DPAR_INTPNT_QO_TOL_MU_RED

Groups [*Interior-point method*](#), [*Termination criteria*](#)

`dparam.intpnt_qo_tol_near_rel`

Optimality tolerance used by the interior-point optimizer for quadratic problems. If **MOSEK** cannot compute a solution that has the prescribed accuracy then it will check if the solution found satisfies the termination criteria with all tolerances multiplied by the value of this parameter. If yes, then the solution is also declared optimal.

Default 1000

Accepted [1.0; +inf]

Example `task.putdoupam(dparam.intpnt_qo_tol_near_rel, 1000)`

Generic name MSK_DPAR_INTPNT_QO_TOL_NEAR_REL

Groups [*Interior-point method*](#), [*Termination criteria*](#)

`dparam.intpnt_qo_tol_pfeas`

Primal feasibility tolerance used by the interior-point optimizer for quadratic problems.

Default 1.0e-8

Accepted [0.0; 1.0]

Example `task.putdoupam(dparam.intpnt_qo_tol_pfeas, 1.0e-8)`

See also [*dparam.intpnt_qo_tol_near_rel*](#)

Generic name MSK_DPAR_INTPNT_QO_TOL_PFEAS

Groups [*Interior-point method*](#), [*Termination criteria*](#)

`dparam.intpnt_qo_tol_rel_gap`

Relative gap termination tolerance used by the interior-point optimizer for quadratic problems.

Default 1.0e-8

Accepted [0.0; 1.0]

Example `task.putdoupam(dparam.intpnt_qo_tol_rel_gap, 1.0e-8)`

See also `dparam.intpnt_qo_tol_near_rel`

Generic name MSK_DPAR_INTPNT_QO_TOL_REL_GAP

Groups *Interior-point method, Termination criteria*

`dparam.intpnt_tol_dfeas`

Dual feasibility tolerance used by the interior-point optimizer for linear problems.

Default 1.0e-8

Accepted [0.0; 1.0]

Example `task.putdouparam(dparam.intpnt_tol_dfeas, 1.0e-8)`

Generic name MSK_DPAR_INTPNT_TOL_DFEAS

Groups *Interior-point method, Termination criteria*

`dparam.intpnt_tol_dsafe`

Controls the initial dual starting point used by the interior-point optimizer. If the interior-point optimizer converges slowly and/or the constraint or variable bounds are very large, then it might be worthwhile to increase this value.

Default 1.0

Accepted [1.0e-4; +inf]

Example `task.putdouparam(dparam.intpnt_tol_dsafe, 1.0)`

Generic name MSK_DPAR_INTPNT_TOL_DSAFE

Groups *Interior-point method*

`dparam.intpnt_tol_infeas`

Infeasibility tolerance used by the interior-point optimizer for linear problems. Controls when the interior-point optimizer declares the model primal or dual infeasible. A small number means the optimizer gets more conservative about declaring the model infeasible.

Default 1.0e-10

Accepted [0.0; 1.0]

Example `task.putdouparam(dparam.intpnt_tol_infeas, 1.0e-10)`

Generic name MSK_DPAR_INTPNT_TOL_INFEAS

Groups *Interior-point method, Termination criteria*

`dparam.intpnt_tol_mu_red`

Relative complementarity gap tolerance used by the interior-point optimizer for linear problems.

Default 1.0e-16

Accepted [0.0; 1.0]

Example `task.putdouparam(dparam.intpnt_tol_mu_red, 1.0e-16)`

Generic name MSK_DPAR_INTPNT_TOL_MU_RED

Groups *Interior-point method, Termination criteria*

`dparam.intpnt_tol_path`

Controls how close the interior-point optimizer follows the central path. A large value of this parameter means the central path is followed very closely. On numerically unstable problems it may be worthwhile to increase this parameter.

Default 1.0e-8

Accepted [0.0; 0.9999]

Example `task.putdouparam(dparam.intpnt_tol_path, 1.0e-8)`

Generic name MSK_DPAR_INTPNT_TOL_PATH

Groups *Interior-point method*

`dparam.intpnt_tol_pfeas`

Primal feasibility tolerance used by the interior-point optimizer for linear problems.

Default 1.0e-8

Accepted [0.0; 1.0]

Example `task.putdoupparam(dparam.intpnt_tol_pfeas, 1.0e-8)`

Generic name MSK_DPAR_INTPNT_TOL_PFEAS

Groups *Interior-point method, Termination criteria*

`dparam.intpnt_tol_psafe`

Controls the initial primal starting point used by the interior-point optimizer. If the interior-point optimizer converges slowly and/or the constraint or variable bounds are very large, then it may be worthwhile to increase this value.

Default 1.0

Accepted [1.0e-4; +inf]

Example `task.putdoupparam(dparam.intpnt_tol_psafe, 1.0)`

Generic name MSK_DPAR_INTPNT_TOL_PSAFE

Groups *Interior-point method*

`dparam.intpnt_tol_rel_gap`

Relative gap termination tolerance used by the interior-point optimizer for linear problems.

Default 1.0e-8

Accepted [1.0e-14; +inf]

Example `task.putdoupparam(dparam.intpnt_tol_rel_gap, 1.0e-8)`

Generic name MSK_DPAR_INTPNT_TOL_REL_GAP

Groups *Termination criteria, Interior-point method*

`dparam.intpnt_tol_rel_step`

Relative step size to the boundary for linear and quadratic optimization problems.

Default 0.9999

Accepted [1.0e-4; 0.999999]

Example `task.putdoupparam(dparam.intpnt_tol_rel_step, 0.9999)`

Generic name MSK_DPAR_INTPNT_TOL_REL_STEP

Groups *Interior-point method*

`dparam.intpnt_tol_step_size`

Minimal step size tolerance. If the step size falls below the value of this parameter, then the interior-point optimizer assumes that it is stalled. In other words the interior-point optimizer does not make any progress and therefore it is better to stop.

Default 1.0e-6

Accepted [0.0; 1.0]

Example `task.putdoupparam(dparam.intpnt_tol_step_size, 1.0e-6)`

Generic name MSK_DPAR_INTPNT_TOL_STEP_SIZE

Groups *Interior-point method*

`dparam.lower_obj_cut`

If either a primal or dual feasible solution is found proving that the optimal objective value is outside the interval [*dparam.lower_obj_cut*, *dparam.upper_obj_cut*], then **MOSEK** is terminated.

Default -1.0e30

Accepted [-inf; +inf]

Example `task.putdoupparam(dparam.lower_obj_cut, -1.0e30)`

See also *dparam.lower_obj_cut_finite_trh*

Generic name MSK_DPAR_LOWER_OBJ_CUT

Groups *Termination criteria*

`dparam.lower_obj_cut_finite_trh`

If the lower objective cut is less than the value of this parameter value, then the lower objective cut i.e. *dparam.lower_obj_cut* is treated as $-\infty$.

Default -0.5e30
Accepted [-inf; +inf]
Example `task.putdoupparam(dparam.lower_obj_cut_finite_trh, -0.5e30)`
Generic name MSK_DPAR_LOWER_OBJ_CUT_FINITE_TRH
Groups *Termination criteria*

`dparam.mio_max_time`

This parameter limits the maximum time spent by the mixed-integer optimizer. A negative number means infinity.

Default -1.0
Accepted [-inf; +inf]
Example `task.putdoupparam(dparam.mio_max_time, -1.0)`
Generic name MSK_DPAR_MIO_MAX_TIME
Groups *Mixed-integer optimization, Termination criteria*

`dparam.mio_rel_gap_const`

This value is used to compute the relative gap for the solution to an integer optimization problem.

Default 1.0e-10
Accepted [1.0e-15; +inf]
Example `task.putdoupparam(dparam.mio_rel_gap_const, 1.0e-10)`
Generic name MSK_DPAR_MIO_REL_GAP_CONST
Groups *Mixed-integer optimization, Termination criteria*

`dparam.mio_tol_abs_gap`

Absolute optimality tolerance employed by the mixed-integer optimizer.

Default 0.0
Accepted [0.0; +inf]
Example `task.putdoupparam(dparam.mio_tol_abs_gap, 0.0)`
Generic name MSK_DPAR_MIO_TOL_ABS_GAP
Groups *Mixed-integer optimization*

`dparam.mio_tol_abs_relax_int`

Absolute integer feasibility tolerance. If the distance to the nearest integer is less than this tolerance then an integer constraint is assumed to be satisfied.

Default 1.0e-5
Accepted [1e-9; +inf]
Example `task.putdoupparam(dparam.mio_tol_abs_relax_int, 1.0e-5)`
Generic name MSK_DPAR_MIO_TOL_ABS_RELAX_INT
Groups *Mixed-integer optimization*

`dparam.mio_tol_feas`

Feasibility tolerance for mixed integer solver.

Default 1.0e-6
Accepted [1e-9; 1e-3]
Example `task.putdoupparam(dparam.mio_tol_feas, 1.0e-6)`
Generic name MSK_DPAR_MIO_TOL_FEAS
Groups *Mixed-integer optimization*

`dparam.mio_tol_rel_dual_bound_improvement`

If the relative improvement of the dual bound is smaller than this value, the solver will terminate the root cut generation. A value of 0.0 means that the value is selected automatically.

Default 0.0
Accepted [0.0; 1.0]

Example `task.putdouparam(dparam.mio_tol_rel_dual_bound_improvement, 0.0)`
Generic name `MSK_DPAR_MIO_TOL_REL_DUAL_BOUND_IMPROVEMENT`
Groups *Mixed-integer optimization*

`dparam.mio_tol_rel_gap`

Relative optimality tolerance employed by the mixed-integer optimizer.

Default `1.0e-4`
Accepted `[0.0; +inf]`
Example `task.putdouparam(dparam.mio_tol_rel_gap, 1.0e-4)`
Generic name `MSK_DPAR_MIO_TOL_REL_GAP`
Groups *Mixed-integer optimization, Termination criteria*

`dparam.optimizer_max_time`

Maximum amount of time the optimizer is allowed to spent on the optimization. A negative number means infinity.

Default `-1.0`
Accepted `[-inf; +inf]`
Example `task.putdouparam(dparam.optimizer_max_time, -1.0)`
Generic name `MSK_DPAR_OPTIMIZER_MAX_TIME`
Groups *Termination criteria*

`dparam.presolve_tol_abs_lindep`

Absolute tolerance employed by the linear dependency checker.

Default `1.0e-6`
Accepted `[0.0; +inf]`
Example `task.putdouparam(dparam.presolve_tol_abs_lindep, 1.0e-6)`
Generic name `MSK_DPAR_PREOLVE_TOL_ABS_LINDEP`
Groups *Presolve*

`dparam.presolve_tol_aij`

Absolute zero tolerance employed for a_{ij} in the presolve.

Default `1.0e-12`
Accepted `[1.0e-15; +inf]`
Example `task.putdouparam(dparam.presolve_tol_aij, 1.0e-12)`
Generic name `MSK_DPAR_PREOLVE_TOL_AIJ`
Groups *Presolve*

`dparam.presolve_tol_rel_lindep`

Relative tolerance employed by the linear dependency checker.

Default `1.0e-10`
Accepted `[0.0; +inf]`
Example `task.putdouparam(dparam.presolve_tol_rel_lindep, 1.0e-10)`
Generic name `MSK_DPAR_PREOLVE_TOL_REL_LINDEP`
Groups *Presolve*

`dparam.presolve_tol_s`

Absolute zero tolerance employed for s_i in the presolve.

Default `1.0e-8`
Accepted `[0.0; +inf]`
Example `task.putdouparam(dparam.presolve_tol_s, 1.0e-8)`
Generic name `MSK_DPAR_PREOLVE_TOL_S`

Groups *Presolve*

`dparam.presolve_tol_x`

Absolute zero tolerance employed for x_j in the presolve.

Default 1.0e-8

Accepted [0.0; +inf]

Example `task.putdouparam(dparam.presolve_tol_x, 1.0e-8)`

Generic name MSK_DPAR_PREOLVE_TOL_X

Groups *Presolve*

`dparam.qcqp_reformulate_rel_drop_tol`

This parameter determines when columns are dropped in incomplete Cholesky factorization during reformulation of quadratic problems.

Default 1e-15

Accepted [0; +inf]

Example `task.putdouparam(dparam.qcqp_reformulate_rel_drop_tol, 1e-15)`

Generic name MSK_DPAR_QCQP_REFORMULATE_REL_DROP_TOL

Groups *Interior-point method*

`dparam.semidefinite_tol_approx`

Tolerance to define a matrix to be positive semidefinite.

Default 1.0e-10

Accepted [1.0e-15; +inf]

Example `task.putdouparam(dparam.semidefinite_tol_approx, 1.0e-10)`

Generic name MSK_DPAR_SEMIDEFINITE_TOL_APPROX

Groups *Data check*

`dparam.sim_lu_tol_rel_piv`

Relative pivot tolerance employed when computing the LU factorization of the basis in the simplex optimizers and in the basis identification procedure. A value closer to 1.0 generally improves numerical stability but typically also implies an increase in the computational work.

Default 0.01

Accepted [1.0e-6; 0.999999]

Example `task.putdouparam(dparam.sim_lu_tol_rel_piv, 0.01)`

Generic name MSK_DPAR_SIM_LU_TOL_REL_PIV

Groups *Basis identification, Simplex optimizer*

`dparam.simplex_abs_tol_piv`

Absolute pivot tolerance employed by the simplex optimizers.

Default 1.0e-7

Accepted [1.0e-12; +inf]

Example `task.putdouparam(dparam.simplex_abs_tol_piv, 1.0e-7)`

Generic name MSK_DPAR_SIMPLEX_ABS_TOL_PIV

Groups *Simplex optimizer*

`dparam.upper_obj_cut`

If either a primal or dual feasible solution is found proving that the optimal objective value is outside the interval [*dparam.lower_obj_cut*, *dparam.upper_obj_cut*], then **MOSEK** is terminated.

Default 1.0e30

Accepted [-inf; +inf]

Example `task.putdouparam(dparam.upper_obj_cut, 1.0e30)`

See also *dparam.upper_obj_cut_finite_trh*

Generic name MSK_DPAR_UPPER_OBJ_CUT

Groups *Termination criteria*

`dparam.upper_obj_cut_finite_trh`

If the upper objective cut is greater than the value of this parameter, then the upper objective cut `dparam.upper_obj_cut` is treated as ∞ .

Default 0.5e30

Accepted [-inf; +inf]

Example `task.putdparam(dparam.upper_obj_cut_finite_trh, 0.5e30)`

Generic name MSK_DPAR_UPPER_OBJ_CUT_FINITE_TRH

Groups *Termination criteria*

15.7.2 Integer parameters

`iparam`

The enumeration type containing all integer parameters.

`iparam.ana_sol_basis`

Controls whether the basis matrix is analyzed in solution analyzer.

Default *on*

Accepted *on, off* (see *onoffkey*)

Example `task.putintparam(iparam.ana_sol_basis, onoffkey.on.value)`

Generic name MSK_IPAR_ANA_SOL_BASIS

Groups *Analysis*

`iparam.ana_sol_print_violated`

A parameter of the problem analyzer. Controls whether a list of violated constraints is printed. All constraints violated by more than the value set by the parameter `dparam.ana_sol_infeas_tol` will be printed.

Default *off*

Accepted *on, off* (see *onoffkey*)

Example `task.putintparam(iparam.ana_sol_print_violated, onoffkey.off.value)`

Generic name MSK_IPAR_ANA_SOL_PRINT_VIOLATED

Groups *Analysis*

`iparam.auto_sort_a_before_opt`

Controls whether the elements in each column of *A* are sorted before an optimization is performed. This is not required but makes the optimization more deterministic.

Default *off*

Accepted *on, off* (see *onoffkey*)

Example `task.putintparam(iparam.auto_sort_a_before_opt, onoffkey.off.value)`

Generic name MSK_IPAR_AUTO_SORT_A_BEFORE_OPT

Groups *Debugging*

`iparam.auto_update_sol_info`

Controls whether the solution information items are automatically updated after an optimization is performed.

Default *off*

Accepted *on, off* (see *onoffkey*)

Example `task.putintparam(iparam.auto_update_sol_info, onoffkey.off.value)`

Generic name MSK_IPAR_AUTO_UPDATE_SOL_INFO

Groups *Overall system*

`iparam.basis_solve_use_plus_one`

If a slack variable is in the basis, then the corresponding column in the basis is a unit vector with -1 in the right position. However, if this parameter is set to *onoffkey.on*, -1 is replaced by 1.

This has significance for the results returned by the *Task.solvewithbasis* function.

Default *off*

Accepted *on, off* (see *onoffkey*)

Example `task.putintparam(iparam.basis_solve_use_plus_one, onoffkey.off.value)`

Generic name MSK_IPAR_BASIS_SOLVE_USE_PLUS_ONE

Groups *Simplex optimizer*

`iparam.bi_clean_optimizer`

Controls which simplex optimizer is used in the clean-up phase. Anything else than *optimizertype.primal_simplex* or *optimizertype.dual_simplex* is equivalent to *optimizertype.free_simplex*.

Default *free*

Accepted *free, intpnt, conic, primal_simplex, dual_simplex, free_simplex, mixed_int* (see *optimizertype*)

Example `task.putintparam(iparam.bi_clean_optimizer, optimizertype.free.value)`

Generic name MSK_IPAR_BI_CLEAN_OPTIMIZER

Groups *Basis identification, Overall solver*

`iparam.bi_ignore_max_iter`

If the parameter *iparam.intpnt_basis* has the value *basindtype.no_error* and the interior-point optimizer has terminated due to maximum number of iterations, then basis identification is performed if this parameter has the value *onoffkey.on*.

Default *off*

Accepted *on, off* (see *onoffkey*)

Example `task.putintparam(iparam.bi_ignore_max_iter, onoffkey.off.value)`

Generic name MSK_IPAR_BI_IGNORE_MAX_ITER

Groups *Interior-point method, Basis identification*

`iparam.bi_ignore_num_error`

If the parameter *iparam.intpnt_basis* has the value *basindtype.no_error* and the interior-point optimizer has terminated due to a numerical problem, then basis identification is performed if this parameter has the value *onoffkey.on*.

Default *off*

Accepted *on, off* (see *onoffkey*)

Example `task.putintparam(iparam.bi_ignore_num_error, onoffkey.off.value)`

Generic name MSK_IPAR_BI_IGNORE_NUM_ERROR

Groups *Interior-point method, Basis identification*

`iparam.bi_max_iterations`

Controls the maximum number of simplex iterations allowed to optimize a basis after the basis identification.

Default 1000000

Accepted [0; +inf]

Example `task.putintparam(iparam.bi_max_iterations, 1000000)`

Generic name MSK_IPAR_BI_MAX_ITERATIONS

Groups *Basis identification, Termination criteria*

`iparam.cache_license`

Specifies if the license is kept checked out for the lifetime of the **MOSEK** environment/model/process (*onoffkey.on*) or returned to the server immediately after the optimization (*onoffkey.off*).

By default the license is checked out for the lifetime of the **MOSEK** environment by the first call to *Task.optimize*.

Check-in and check-out of licenses have an overhead. Frequent communication with the license server should be avoided.

Default *on*

Accepted *on, off* (see *onoffkey*)

Example `task.putintparam(iparam.cache_license, onoffkey.on.value)`

Generic name `MSK_IPAR_CACHE_LICENSE`

Groups *License manager*

`iparam.check_convexity`

Specify the level of convexity check on quadratic problems.

Default *full*

Accepted *none, simple, full* (see *checkconvexitytype*)

Example `task.putintparam(iparam.check_convexity, checkconvexitytype.full.value)`

Generic name `MSK_IPAR_CHECK_CONVEXITY`

Groups *Data check*

`iparam.compress_statfile`

Control compression of stat files.

Default *on*

Accepted *on, off* (see *onoffkey*)

Example `task.putintparam(iparam.compress_statfile, onoffkey.on.value)`

Generic name `MSK_IPAR_COMPRESS_STATFILE`

`iparam.infeas_generic_names`

Controls whether generic names are used when an infeasible subproblem is created.

Default *off*

Accepted *on, off* (see *onoffkey*)

Example `task.putintparam(iparam.infeas_generic_names, onoffkey.off.value)`

Generic name `MSK_IPAR_INFEAS_GENERIC_NAMES`

Groups *Infeasibility report*

`iparam.infeas_prefer_primal`

If both certificates of primal and dual infeasibility are supplied then only the primal is used when this option is turned on.

Default *on*

Accepted *on, off* (see *onoffkey*)

Example `task.putintparam(iparam.infeas_prefer_primal, onoffkey.on.value)`

Generic name `MSK_IPAR_INFEAS_PREFER_PRIMAL`

Groups *Overall solver*

`iparam.infeas_report_auto`

Controls whether an infeasibility report is automatically produced after the optimization if the problem is primal or dual infeasible.

Default *off*

Accepted *on, off* (see *onoffkey*)

Example `task.putintparam(iparam.infeas_report_auto, onoffkey.off.value)`

Generic name MSK_IPAR_INFEAS_REPORT_AUTO

Groups *Data input/output, Solution input/output*

`iparam.infeas_report_level`

Controls the amount of information presented in an infeasibility report. Higher values imply more information.

Default 1

Accepted [0; +inf]

Example `task.putintparam(iparam.infeas_report_level, 1)`

Generic name MSK_IPAR_INFEAS_REPORT_LEVEL

Groups *Infeasibility report, Output information*

`iparam.intpnt_basis`

Controls whether the interior-point optimizer also computes an optimal basis.

Default *always*

Accepted *never, always, no_error, if_feasible, reserved* (see *basindtype*)

Example `task.putintparam(iparam.intpnt_basis, basindtype.always.value)`

See also *iparam.bi_ignore_max_iter, iparam.bi_ignore_num_error, iparam.bi_max_iterations, iparam.bi_clean_optimizer*

Generic name MSK_IPAR_INTPNT_BASIS

Groups *Interior-point method, Basis identification*

`iparam.intpnt_diff_step`

Controls whether different step sizes are allowed in the primal and dual space.

Default *on*

Accepted

- *on*: Different step sizes are allowed.
- *off*: Different step sizes are not allowed.

Example `task.putintparam(iparam.intpnt_diff_step, onoffkey.on.value)`

Generic name MSK_IPAR_INTPNT_DIFF_STEP

Groups *Interior-point method*

`iparam.intpnt_hotstart`

Currently not in use.

Default *none*

Accepted *none, primal, dual, primal_dual* (see *intpnthotstart*)

Example `task.putintparam(iparam.intpnt_hotstart, intpnthotstart.none.value)`

Generic name MSK_IPAR_INTPNT_HOTSTART

Groups *Interior-point method*

`iparam.intpnt_max_iterations`

Controls the maximum number of iterations allowed in the interior-point optimizer.

Default 400

Accepted [0; +inf]

Example `task.putintparam(iparam.intpnt_max_iterations, 400)`

Generic name MSK_IPAR_INTPNT_MAX_ITERATIONS

Groups *Interior-point method, Termination criteria*

`iparam.intpnt_max_num_cor`

Controls the maximum number of correctors allowed by the multiple corrector procedure. A negative value means that **MOSEK** is making the choice.

Default -1
Accepted [-1; +inf]
Example `task.putintparam(iparam.intpnt_max_num_cor, -1)`
Generic name MSK_IPAR_INTPNT_MAX_NUM_COR
Groups *Interior-point method*

`iparam.intpnt_max_num_refinement_steps`

Maximum number of steps to be used by the iterative refinement of the search direction. A negative value implies that the optimizer chooses the maximum number of iterative refinement steps.

Default -1
Accepted [-inf; +inf]
Example `task.putintparam(iparam.intpnt_max_num_refinement_steps, -1)`
Generic name MSK_IPAR_INTPNT_MAX_NUM_REFINEMENT_STEPS
Groups *Interior-point method*

`iparam.intpnt_multi_thread`

Controls whether the interior-point optimizers are allowed to employ multiple threads if more threads is available.

Default *on*
Accepted *on, off* (see *onoffkey*)
Example `task.putintparam(iparam.intpnt_multi_thread, onoffkey.on.value)`
Generic name MSK_IPAR_INTPNT_MULTI_THREAD
Groups *Overall system*

`iparam.intpnt_off_col_trh`

Controls how many offending columns are detected in the Jacobian of the constraint matrix.

0	no detection
1	aggressive detection
> 1	higher values mean less aggressive detection

Default 40
Accepted [0; +inf]
Example `task.putintparam(iparam.intpnt_off_col_trh, 40)`
Generic name MSK_IPAR_INTPNT_OFF_COL_TRH
Groups *Interior-point method*

`iparam.intpnt_order_gp_num_seeds`

The GP ordering is dependent on a random seed. Therefore, trying several random seeds may lead to a better ordering. This parameter controls the number of random seeds tried.

A value of 0 means that MOSEK makes the choice.

Default 0
Accepted [0; +inf]
Example `task.putintparam(iparam.intpnt_order_gp_num_seeds, 0)`
Generic name MSK_IPAR_INTPNT_ORDER_GP_NUM_SEEDS
Groups *Interior-point method*

`iparam.intpnt_order_method`

Controls the ordering strategy used by the interior-point optimizer when factorizing the Newton equation system.

Default *free*
Accepted *free, appminloc, experimental, try_graphpar, force_graphpar, none*
 (see *orderingtype*)

Example `task.putintparam(iparam.intpnt_order_method, orderingtype.free.value)`

Generic name `MSK_IPAR_INTPNT_ORDER_METHOD`

Groups *Interior-point method*

`iparam.intpnt_purify`

Currently not in use.

Default *none*

Accepted *none, primal, dual, primal_dual, auto* (see *purify*)

Example `task.putintparam(iparam.intpnt_purify, purify.none.value)`

Generic name `MSK_IPAR_INTPNT_PURIFY`

Groups *Interior-point method*

`iparam.intpnt_regularization_use`

Controls whether regularization is allowed.

Default *on*

Accepted *on, off* (see *onoffkey*)

Example `task.putintparam(iparam.intpnt_regularization_use, onoffkey.on.value)`

Generic name `MSK_IPAR_INTPNT_REGULARIZATION_USE`

Groups *Interior-point method*

`iparam.intpnt_scaling`

Controls how the problem is scaled before the interior-point optimizer is used.

Default *free*

Accepted *free, none, moderate, aggressive* (see *scalingtype*)

Example `task.putintparam(iparam.intpnt_scaling, scalingtype.free.value)`

Generic name `MSK_IPAR_INTPNT_SCALING`

Groups *Interior-point method*

`iparam.intpnt_solve_form`

Controls whether the primal or the dual problem is solved.

Default *free*

Accepted *free, primal, dual* (see *solveform*)

Example `task.putintparam(iparam.intpnt_solve_form, solveform.free.value)`

Generic name `MSK_IPAR_INTPNT_SOLVE_FORM`

Groups *Interior-point method*

`iparam.intpnt_starting_point`

Starting point used by the interior-point optimizer.

Default *free*

Accepted *free, guess, constant, satisfy_bounds* (see *startpointtype*)

Example `task.putintparam(iparam.intpnt_starting_point, startpointtype.free.value)`

Generic name `MSK_IPAR_INTPNT_STARTING_POINT`

Groups *Interior-point method*

`iparam.license_debug`

This option is used to turn on debugging of the license manager.

Default *off*

Accepted *on, off* (see *onoffkey*)

Example `task.putintparam(iparam.license_debug, onoffkey.off.value)`

Generic name MSK_IPAR_LICENSE_DEBUG

Groups *License manager*

`iparam.license_pause_time`

If *iparam.license_wait* is *onoffkey.on* and no license is available, then **MOSEK** sleeps a number of milliseconds between each check of whether a license has become free.

Default 100

Accepted [0; 1000000]

Example `task.putintparam(iparam.license_pause_time, 100)`

Generic name MSK_IPAR_LICENSE_PAUSE_TIME

Groups *License manager*

`iparam.license_suppress_expire_wrns`

Controls whether license features expire warnings are suppressed.

Default *off*

Accepted *on, off* (see *onoffkey*)

Example `task.putintparam(iparam.license_suppress_expire_wrns, onoffkey.off.value)`

Generic name MSK_IPAR_LICENSE_SUPPRESS_EXPIRE_WRNS

Groups *License manager, Output information*

`iparam.license_trh_expiry_wrn`

If a license feature expires in a numbers of days less than the value of this parameter then a warning will be issued.

Default 7

Accepted [0; +inf]

Example `task.putintparam(iparam.license_trh_expiry_wrn, 7)`

Generic name MSK_IPAR_LICENSE_TRH_EXPIRY_WRN

Groups *License manager, Output information*

`iparam.license_wait`

If all licenses are in use **MOSEK** returns with an error code. However, by turning on this parameter **MOSEK** will wait for an available license.

Default *off*

Accepted *on, off* (see *onoffkey*)

Example `task.putintparam(iparam.license_wait, onoffkey.off.value)`

Generic name MSK_IPAR_LICENSE_WAIT

Groups *Overall solver, Overall system, License manager*

`iparam.log`

Controls the amount of log information. The value 0 implies that all log information is suppressed. A higher level implies that more information is logged.

Please note that if a task is employed to solve a sequence of optimization problems the value of this parameter is reduced by the value of *iparam.log_cut_second_opt* for the second and any subsequent optimizations.

Default 10

Accepted [0; +inf]

Example `task.putintparam(iparam.log, 10)`

See also *iparam.log_cut_second_opt*

Generic name MSK_IPAR_LOG

Groups *Output information, Logging*

`iparam.log_ana_pro`

Controls amount of output from the problem analyzer.

Default 1

Accepted [0; +inf]

Example `task.putintparam(iparam.log_ana_pro, 1)`

Generic name MSK_IPAR_LOG_ANA_PRO

Groups *Analysis, Logging*

`iparam.log_bi`

Controls the amount of output printed by the basis identification procedure. A higher level implies that more information is logged.

Default 1

Accepted [0; +inf]

Example `task.putintparam(iparam.log_bi, 1)`

Generic name MSK_IPAR_LOG_BI

Groups *Basis identification, Output information, Logging*

`iparam.log_bi_freq`

Controls how frequently the optimizer outputs information about the basis identification and how frequent the user-defined callback function is called.

Default 2500

Accepted [0; +inf]

Example `task.putintparam(iparam.log_bi_freq, 2500)`

Generic name MSK_IPAR_LOG_BI_FREQ

Groups *Basis identification, Output information, Logging*

`iparam.log_check_convexity`

Controls logging in convexity check on quadratic problems. Set to a positive value to turn logging on. If a quadratic coefficient matrix is found to violate the requirement of PSD (NSD) then a list of negative (positive) pivot elements is printed. The absolute value of the pivot elements is also shown.

Default 0

Accepted [0; +inf]

Example `task.putintparam(iparam.log_check_convexity, 0)`

Generic name MSK_IPAR_LOG_CHECK_CONVEXITY

Groups *Data check*

`iparam.log_cut_second_opt`

If a task is employed to solve a sequence of optimization problems, then the value of the log levels is reduced by the value of this parameter. E.g *iparam.log* and *iparam.log_sim* are reduced by the value of this parameter for the second and any subsequent optimizations.

Default 1

Accepted [0; +inf]

Example `task.putintparam(iparam.log_cut_second_opt, 1)`

See also *iparam.log*, *iparam.log_intpnt*, *iparam.log_mio*, *iparam.log_sim*

Generic name MSK_IPAR_LOG_CUT_SECOND_OPT

Groups *Output information, Logging*

`iparam.log_expand`

Controls the amount of logging when a data item such as the maximum number constraints is expanded.

Default 0

Accepted [0; +inf]

Example `task.putintparam(iparam.log_expand, 0)`

Generic name MSK_IPAR_LOG_EXPAND

Groups *Output information, Logging*

`iparam.log_feas_repair`

Controls the amount of output printed when performing feasibility repair. A value higher than one means extensive logging.

Default 1

Accepted [0; +inf]

Example `task.putintparam(iparam.log_feas_repair, 1)`

Generic name MSK_IPAR_LOG_FEAS_REPAIR

Groups *Output information, Logging*

`iparam.log_file`

If turned on, then some log info is printed when a file is written or read.

Default 1

Accepted [0; +inf]

Example `task.putintparam(iparam.log_file, 1)`

Generic name MSK_IPAR_LOG_FILE

Groups *Data input/output, Output information, Logging*

`iparam.log_include_summary`

If on, then the solution summary will be printed by *Task.optimize*, so a separate call to *Task.solutionssummary* is not necessary.

Default *off*

Accepted *on, off* (see *onoffkey*)

Example `task.putintparam(iparam.log_include_summary, onoffkey.off.value)`

Generic name MSK_IPAR_LOG_INCLUDE_SUMMARY

Groups *Output information, Logging*

`iparam.log_infeas_ana`

Controls amount of output printed by the infeasibility analyzer procedures. A higher level implies that more information is logged.

Default 1

Accepted [0; +inf]

Example `task.putintparam(iparam.log_infeas_ana, 1)`

Generic name MSK_IPAR_LOG_INFEAS_ANA

Groups *Infeasibility report, Output information, Logging*

`iparam.log_intpnt`

Controls amount of output printed by the interior-point optimizer. A higher level implies that more information is logged.

Default 1

Accepted [0; +inf]

Example `task.putintparam(iparam.log_intpnt, 1)`

Generic name MSK_IPAR_LOG_INTPNT

Groups *Interior-point method, Output information, Logging*

`iparam.log_local_info`

Controls whether local identifying information like environment variables, filenames, IP addresses etc. are printed to the log.

Note that this will only affect some functions. Some functions that specifically emit system information will not be affected.

Default *on*

Accepted *on, off* (see *onoffkey*)

Example `task.putintparam(iparam.log_local_info, onoffkey.on.value)`

Generic name MSK_IPAR_LOG_LOCAL_INFO

Groups *Output information, Logging*

`iparam.log_mio`

Controls the log level for the mixed-integer optimizer. A higher level implies that more information is logged.

Default 4

Accepted [0; +inf]

Example `task.putintparam(iparam.log_mio, 4)`

Generic name MSK_IPAR_LOG_MIO

Groups *Mixed-integer optimization, Output information, Logging*

`iparam.log_mio_freq`

Controls how frequent the mixed-integer optimizer prints the log line. It will print line every time *iparam.log_mio_freq* relaxations have been solved.

Default 10

Accepted [-inf; +inf]

Example `task.putintparam(iparam.log_mio_freq, 10)`

Generic name MSK_IPAR_LOG_MIO_FREQ

Groups *Mixed-integer optimization, Output information, Logging*

`iparam.log_order`

If turned on, then factor lines are added to the log.

Default 1

Accepted [0; +inf]

Example `task.putintparam(iparam.log_order, 1)`

Generic name MSK_IPAR_LOG_ORDER

Groups *Output information, Logging*

`iparam.log_presolve`

Controls amount of output printed by the presolve procedure. A higher level implies that more information is logged.

Default 1

Accepted [0; +inf]

Example `task.putintparam(iparam.log_presolve, 1)`

Generic name MSK_IPAR_LOG_PREOLVE

Groups *Logging*

`iparam.log_response`

Controls amount of output printed when response codes are reported. A higher level implies that more information is logged.

Default 0

Accepted [0; +inf]

Example `task.putintparam(iparam.log_response, 0)`

Generic name MSK_IPAR_LOG_RESPONSE

Groups *Output information, Logging*

`iparam.log_sensitivity`

Controls the amount of logging during the sensitivity analysis.

- 0. Means no logging information is produced.
- 1. Timing information is printed.

- 2. Sensitivity results are printed.

Default 1

Accepted [0; +inf]

Example `task.putintparam(iparam.log_sensitivity, 1)`

Generic name MSK_IPAR_LOG_SENSITIVITY

Groups *Output information, Logging*

`iparam.log_sensitivity_opt`

Controls the amount of logging from the optimizers employed during the sensitivity analysis. 0 means no logging information is produced.

Default 0

Accepted [0; +inf]

Example `task.putintparam(iparam.log_sensitivity_opt, 0)`

Generic name MSK_IPAR_LOG_SENSITIVITY_OPT

Groups *Output information, Logging*

`iparam.log_sim`

Controls amount of output printed by the simplex optimizer. A higher level implies that more information is logged.

Default 4

Accepted [0; +inf]

Example `task.putintparam(iparam.log_sim, 4)`

Generic name MSK_IPAR_LOG_SIM

Groups *Simplex optimizer, Output information, Logging*

`iparam.log_sim_freq`

Controls how frequent the simplex optimizer outputs information about the optimization and how frequent the user-defined callback function is called.

Default 1000

Accepted [0; +inf]

Example `task.putintparam(iparam.log_sim_freq, 1000)`

Generic name MSK_IPAR_LOG_SIM_FREQ

Groups *Simplex optimizer, Output information, Logging*

`iparam.log_sim_minor`

Currently not in use.

Default 1

Accepted [0; +inf]

Example `task.putintparam(iparam.log_sim_minor, 1)`

Generic name MSK_IPAR_LOG_SIM_MINOR

Groups *Simplex optimizer, Output information*

`iparam.log_storage`

When turned on, **MOSEK** prints messages regarding the storage usage and allocation.

Default 0

Accepted [0; +inf]

Example `task.putintparam(iparam.log_storage, 0)`

Generic name MSK_IPAR_LOG_STORAGE

Groups *Output information, Overall system, Logging*

`iparam.max_num_warnings`

Each warning is shown a limited number of times controlled by this parameter. A negative value is identical to infinite number of times.

Default 10
Accepted [-inf; +inf]
Example task.putintparam(iparam.max_num_warnings, 10)
Generic name MSK_IPAR_MAX_NUM_WARNINGS
Groups *Output information*

iparam.mio_branch_dir

Controls whether the mixed-integer optimizer is branching up or down by default.

Default *free*
Accepted *free, up, down, near, far, root_lp, guided, pseudocost* (see *branchdir*)
Example task.putintparam(iparam.mio_branch_dir, branchdir.free.value)
Generic name MSK_IPAR_MIO_BRANCH_DIR
Groups *Mixed-integer optimization*

iparam.mio_conic_outer_approximation

If this option is turned on outer approximation is used when solving relaxations of conic problems; otherwise interior point is used.

Default *off*
Accepted *on, off* (see *onoffkey*)
Example task.putintparam(iparam.mio_conic_outer_approximation, onoffkey.off.value)
Generic name MSK_IPAR_MIO_CONIC_OUTER_APPROXIMATION
Groups *Mixed-integer optimization*

iparam.mio_cut_clique

Controls whether clique cuts should be generated.

Default *on*
Accepted *on, off* (see *onoffkey*)
Example task.putintparam(iparam.mio_cut_clique, onoffkey.on.value)
Generic name MSK_IPAR_MIO_CUT_CLIQUE
Groups *Mixed-integer optimization*

iparam.mio_cut_cmir

Controls whether mixed integer rounding cuts should be generated.

Default *on*
Accepted *on, off* (see *onoffkey*)
Example task.putintparam(iparam.mio_cut_cmir, onoffkey.on.value)
Generic name MSK_IPAR_MIO_CUT_CMIR
Groups *Mixed-integer optimization*

iparam.mio_cut_gmi

Controls whether GMI cuts should be generated.

Default *on*
Accepted *on, off* (see *onoffkey*)
Example task.putintparam(iparam.mio_cut_gmi, onoffkey.on.value)
Generic name MSK_IPAR_MIO_CUT_GMI
Groups *Mixed-integer optimization*

iparam.mio_cut_implied_bound

Controls whether implied bound cuts should be generated.

Default *off*
Accepted *on, off* (see *onoffkey*)

Example `task.putintparam(iparam.mio_cut_implied_bound, onoffkey.off.value)`

Generic name MSK_IPAR_MIO_CUT_IMPLIED_BOUND

Groups *Mixed-integer optimization*

`iparam.mio_cut_knapsack_cover`

Controls whether knapsack cover cuts should be generated.

Default *off*

Accepted *on, off* (see *onoffkey*)

Example `task.putintparam(iparam.mio_cut_knapsack_cover, onoffkey.off.value)`

Generic name MSK_IPAR_MIO_CUT_KNAPSACK_COVER

Groups *Mixed-integer optimization*

`iparam.mio_cut_selection_level`

Controls how aggressively generated cuts are selected to be included in the relaxation.

- -1. The optimizer chooses the level of cut selection
- 0. Generated cuts less likely to be added to the relaxation
- 1. Cuts are more aggressively selected to be included in the relaxation

Default -1

Accepted [-1; +1]

Example `task.putintparam(iparam.mio_cut_selection_level, -1)`

Generic name MSK_IPAR_MIO_CUT_SELECTION_LEVEL

Groups *Mixed-integer optimization*

`iparam.mio_feaspump_level`

Controls the way the Feasibility Pump heuristic is employed by the mixed-integer optimizer.

- -1. The optimizer chooses how the Feasibility Pump is used
- 0. The Feasibility Pump is disabled
- 1. The Feasibility Pump is enabled with an effort to improve solution quality
- 2. The Feasibility Pump is enabled with an effort to reach feasibility early

Default -1

Accepted [-1; 2]

Example `task.putintparam(iparam.mio_feaspump_level, -1)`

Generic name MSK_IPAR_MIO_FEASPUMP_LEVEL

Groups *Mixed-integer optimization*

`iparam.mio_heuristic_level`

Controls the heuristic employed by the mixed-integer optimizer to locate an initial good integer feasible solution. A value of zero means the heuristic is not used at all. A larger value than 0 means that a gradually more sophisticated heuristic is used which is computationally more expensive. A negative value implies that the optimizer chooses the heuristic. Normally a value around 3 to 5 should be optimal.

Default -1

Accepted [-inf; +inf]

Example `task.putintparam(iparam.mio_heuristic_level, -1)`

Generic name MSK_IPAR_MIO_HEURISTIC_LEVEL

Groups *Mixed-integer optimization*

`iparam.mio_max_num_branches`

Maximum number of branches allowed during the branch and bound search. A negative value means infinite.

Default -1
Accepted [-inf; +inf]
Example `task.putintparam(iparam.mio_max_num_branches, -1)`
Generic name MSK_IPAR_MIO_MAX_NUM_BRANCHES
Groups *Mixed-integer optimization, Termination criteria*

`iparam.mio_max_num_relaxs`
Maximum number of relaxations allowed during the branch and bound search. A negative value means infinite.

Default -1
Accepted [-inf; +inf]
Example `task.putintparam(iparam.mio_max_num_relaxs, -1)`
Generic name MSK_IPAR_MIO_MAX_NUM_RELAXS
Groups *Mixed-integer optimization*

`iparam.mio_max_num_root_cut_rounds`
Maximum number of cut separation rounds at the root node.

Default 100
Accepted [0; +inf]
Example `task.putintparam(iparam.mio_max_num_root_cut_rounds, 100)`
Generic name MSK_IPAR_MIO_MAX_NUM_ROOT_CUT_ROUNDS
Groups *Mixed-integer optimization, Termination criteria*

`iparam.mio_max_num_solutions`
The mixed-integer optimizer can be terminated after a certain number of different feasible solutions has been located. If this parameter has the value $n > 0$, then the mixed-integer optimizer will be terminated when n feasible solutions have been located.

Default -1
Accepted [-inf; +inf]
Example `task.putintparam(iparam.mio_max_num_solutions, -1)`
Generic name MSK_IPAR_MIO_MAX_NUM_SOLUTIONS
Groups *Mixed-integer optimization, Termination criteria*

`iparam.mio_mode`
Controls whether the optimizer includes the integer restrictions when solving a (mixed) integer optimization problem.

Default *satisfied*
Accepted *ignored, satisfied* (see *miomode*)
Example `task.putintparam(iparam.mio_mode, miomode.satisfied.value)`
Generic name MSK_IPAR_MIO_MODE
Groups *Overall solver*

`iparam.mio_node_optimizer`
Controls which optimizer is employed at the non-root nodes in the mixed-integer optimizer.

Default *free*
Accepted *free, intpnt, conic, primal_simplex, dual_simplex, free_simplex, mixed_int* (see *optimizertype*)
Example `task.putintparam(iparam.mio_node_optimizer, optimizertype.free.value)`
Generic name MSK_IPAR_MIO_NODE_OPTIMIZER
Groups *Mixed-integer optimization*

`iparam.mio_node_selection`
Controls the node selection strategy employed by the mixed-integer optimizer.

Default *free*

Accepted *free, first, best, pseudo* (see *mionodeseltype*)

Example `task.putintparam(iparam.mio_node_selection, mionodeseltype.free.value)`

Generic name MSK_IPAR_MIO_NODE_SELECTION

Groups *Mixed-integer optimization*

`iparam.mio_perspective_reformulate`

Enables or disables perspective reformulation in presolve.

Default *on*

Accepted *on, off* (see *onoffkey*)

Example `task.putintparam(iparam.mio_perspective_reformulate, onoffkey.on.value)`

Generic name MSK_IPAR_MIO_PERSPECTIVE_REFORMULATE

Groups *Mixed-integer optimization*

`iparam.mio_probing_level`

Controls the amount of probing employed by the mixed-integer optimizer in presolve.

- -1. The optimizer chooses the level of probing employed
- 0. Probing is disabled
- 1. A low amount of probing is employed
- 2. A medium amount of probing is employed
- 3. A high amount of probing is employed

Default -1

Accepted [-1; 3]

Example `task.putintparam(iparam.mio_probing_level, -1)`

Generic name MSK_IPAR_MIO_PROBING_LEVEL

Groups *Mixed-integer optimization*

`iparam.mio_propagate_objective_constraint`

Use objective domain propagation.

Default *off*

Accepted *on, off* (see *onoffkey*)

Example `task.putintparam(iparam.mio_propagate_objective_constraint, onoffkey.off.value)`

Generic name MSK_IPAR_MIO_PROPAGATE_OBJECTIVE_CONSTRAINT

Groups *Mixed-integer optimization*

`iparam.mio_rins_max_nodes`

Controls the maximum number of nodes allowed in each call to the RINS heuristic. The default value of -1 means that the value is determined automatically. A value of zero turns off the heuristic.

Default -1

Accepted [-1; +inf]

Example `task.putintparam(iparam.mio_rins_max_nodes, -1)`

Generic name MSK_IPAR_MIO_RINS_MAX_NODES

Groups *Mixed-integer optimization*

`iparam.mio_root_optimizer`

Controls which optimizer is employed at the root node in the mixed-integer optimizer.

Default *free*

Accepted *free, intpnt, conic, primal_simplex, dual_simplex, free_simplex, mixed_int* (see *optimizertype*)

Example `task.putintparam(iparam.mio_root_optimizer, optimizertype.free.value)`

Generic name MSK_IPAR_MIO_ROOT_OPTIMIZER

Groups *Mixed-integer optimization*

`iparam.mio_root_repeat_presolve_level`

Controls whether presolve can be repeated at root node.

- -1. The optimizer chooses whether presolve is repeated
- 0. Never repeat presolve
- 1. Always repeat presolve

Default -1

Accepted [-1; 1]

Example `task.putintparam(iparam.mio_root_repeat_presolve_level, -1)`

Generic name MSK_IPAR_MIO_ROOT_REPEAT_PREOLVE_LEVEL

Groups *Mixed-integer optimization*

`iparam.mio_seed`

Sets the random seed used for randomization in the mixed integer optimizer. Selecting a different seed can change the path the optimizer takes to the optimal solution.

Default 42

Accepted [0; +inf]

Example `task.putintparam(iparam.mio_seed, 42)`

Generic name MSK_IPAR_MIO_SEED

Groups *Mixed-integer optimization*

`iparam.mio_vb_detection_level`

Controls how much effort is put into detecting variable bounds.

- -1. The optimizer chooses
- 0. No variable bounds are detected
- 1. Only detect variable bounds that are directly represented in the problem
- 2. Detect variable bounds in probing

Default -1

Accepted [-1; +2]

Example `task.putintparam(iparam.mio_vb_detection_level, -1)`

Generic name MSK_IPAR_MIO_VB_DETECTION_LEVEL

Groups *Mixed-integer optimization*

`iparam.mt_spincount`

Set the number of iterations to spin before sleeping.

Default 0

Accepted [0; 1000000000]

Example `task.putintparam(iparam.mt_spincount, 0)`

Generic name MSK_IPAR_MT_SPINCOUNT

Groups *Overall system*

`iparam.num_threads`

Controls the number of threads employed by the optimizer. If set to 0 the number of threads used will be equal to the number of cores detected on the machine.

If using the conic optimizer, the value of this parameter set at first optimization remains constant through the lifetime of the process. **MOSEK** will allocate a thread pool of given size, and changing the parameter value later will have no effect. It will, however, remain possible to demand single-threaded execution by setting `iparam.intpnt_multi_thread`.

For the mixed-integer optimizer and interior-point linear optimizer there is no such restriction. On the `linuxaarch64` platform the settings only applies to the mixed-integer optimizer; the continuous optimizers are single-threaded.

Default 0
Accepted [0; +inf]
Example `task.putintparam(iparam.num_threads, 0)`
Generic name `MSK_IPAR_NUM_THREADS`
Groups *Overall system*

`iparam.opf_write_header`

Write a text header with date and **MOSEK** version in an OPF file.

Default *on*
Accepted *on, off* (see *onoffkey*)
Example `task.putintparam(iparam.opf_write_header, onoffkey.on.value)`
Generic name `MSK_IPAR_OPF_WRITE_HEADER`
Groups *Data input/output*

`iparam.opf_write_hints`

Write a hint section with problem dimensions in the beginning of an OPF file.

Default *on*
Accepted *on, off* (see *onoffkey*)
Example `task.putintparam(iparam.opf_write_hints, onoffkey.on.value)`
Generic name `MSK_IPAR_OPF_WRITE_HINTS`
Groups *Data input/output*

`iparam.opf_write_line_length`

Aim to keep lines in OPF files not much longer than this.

Default 80
Accepted [0; +inf]
Example `task.putintparam(iparam.opf_write_line_length, 80)`
Generic name `MSK_IPAR_OPF_WRITE_LINE_LENGTH`
Groups *Data input/output*

`iparam.opf_write_parameters`

Write a parameter section in an OPF file.

Default *off*
Accepted *on, off* (see *onoffkey*)
Example `task.putintparam(iparam.opf_write_parameters, onoffkey.off.value)`
Generic name `MSK_IPAR_OPF_WRITE_PARAMETERS`
Groups *Data input/output*

`iparam.opf_write_problem`

Write objective, constraints, bounds etc. to an OPF file.

Default *on*
Accepted *on, off* (see *onoffkey*)
Example `task.putintparam(iparam.opf_write_problem, onoffkey.on.value)`
Generic name `MSK_IPAR_OPF_WRITE_PROBLEM`
Groups *Data input/output*

`iparam.opf_write_sol_bas`

If *iparam.opf_write_solutions* is *onoffkey.on* and a basic solution is defined, include the basic solution in OPF files.

Default *on*

Accepted *on, off* (see *onoffkey*)

Example `task.putintparam(iparam.opf_write_sol_bas, onoffkey.on.value)`

Generic name MSK_IPAR_OPF_WRITE_SOL_BAS

Groups *Data input/output*

`iparam.opf_write_sol_itg`

If *iparam.opf_write_solutions* is *onoffkey.on* and an integer solution is defined, write the integer solution in OPF files.

Default *on*

Accepted *on, off* (see *onoffkey*)

Example `task.putintparam(iparam.opf_write_sol_itg, onoffkey.on.value)`

Generic name MSK_IPAR_OPF_WRITE_SOL_ITG

Groups *Data input/output*

`iparam.opf_write_sol_itr`

If *iparam.opf_write_solutions* is *onoffkey.on* and an interior solution is defined, write the interior solution in OPF files.

Default *on*

Accepted *on, off* (see *onoffkey*)

Example `task.putintparam(iparam.opf_write_sol_itr, onoffkey.on.value)`

Generic name MSK_IPAR_OPF_WRITE_SOL_ITR

Groups *Data input/output*

`iparam.opf_write_solutions`

Enable inclusion of solutions in the OPF files.

Default *off*

Accepted *on, off* (see *onoffkey*)

Example `task.putintparam(iparam.opf_write_solutions, onoffkey.off.value)`

Generic name MSK_IPAR_OPF_WRITE_SOLUTIONS

Groups *Data input/output*

`iparam.optimizer`

The parameter controls which optimizer is used to optimize the task.

Default *free*

Accepted *free, intpnt, conic, primal_simplex, dual_simplex, free_simplex, mixed_int* (see *optimizertype*)

Example `task.putintparam(iparam.optimizer, optimizertype.free.value)`

Generic name MSK_IPAR_OPTIMIZER

Groups *Overall solver*

`iparam.param_read_case_name`

If turned on, then names in the parameter file are case sensitive.

Default *on*

Accepted *on, off* (see *onoffkey*)

Example `task.putintparam(iparam.param_read_case_name, onoffkey.on.value)`

Generic name MSK_IPAR_PARAM_READ_CASE_NAME

Groups *Data input/output*

`iparam.param_read_ign_error`

If turned on, then errors in parameter settings is ignored.

Default *off*
Accepted *on, off* (see *onoffkey*)
Example `task.putintparam(iparam.param_read_ign_error, onoffkey.off.value)`
Generic name `MSK_IPAR_PARAM_READ_IGN_ERROR`
Groups *Data input/output*

`iparam.presolve_eliminator_max_fill`
Controls the maximum amount of fill-in that can be created by one pivot in the elimination phase of the presolve. A negative value means the parameter value is selected automatically.

Default `-1`
Accepted `[-inf; +inf]`
Example `task.putintparam(iparam.presolve_eliminator_max_fill, -1)`
Generic name `MSK_IPAR_PREOLVE_ELIMINATOR_MAX_FILL`
Groups *Presolve*

`iparam.presolve_eliminator_max_num_tries`
Control the maximum number of times the eliminator is tried. A negative value implies **MOSEK** decides.

Default `-1`
Accepted `[-inf; +inf]`
Example `task.putintparam(iparam.presolve_eliminator_max_num_tries, -1)`
Generic name `MSK_IPAR_PREOLVE_ELIMINATOR_MAX_NUM_TRIES`
Groups *Presolve*

`iparam.presolve_level`
Currently not used.

Default `-1`
Accepted `[-inf; +inf]`
Example `task.putintparam(iparam.presolve_level, -1)`
Generic name `MSK_IPAR_PREOLVE_LEVEL`
Groups *Overall solver, Presolve*

`iparam.presolve_lindep_abs_work_trh`
Controls linear dependency check in presolve. The linear dependency check is potentially computationally expensive.

Default `100`
Accepted `[-inf; +inf]`
Example `task.putintparam(iparam.presolve_lindep_abs_work_trh, 100)`
Generic name `MSK_IPAR_PREOLVE_LINDEP_ABS_WORK_TRH`
Groups *Presolve*

`iparam.presolve_lindep_rel_work_trh`
Controls linear dependency check in presolve. The linear dependency check is potentially computationally expensive.

Default `100`
Accepted `[-inf; +inf]`
Example `task.putintparam(iparam.presolve_lindep_rel_work_trh, 100)`
Generic name `MSK_IPAR_PREOLVE_LINDEP_REL_WORK_TRH`
Groups *Presolve*

`iparam.presolve_lindep_use`
Controls whether the linear constraints are checked for linear dependencies.

Default *on*

Accepted *on, off* (see *onoffkey*)

Example `task.putintparam(iparam.presolve_lindep_use, onoffkey.on.value)`

Generic name MSK_IPAR_PRESOLVE_LINDEP_USE

Groups *Presolve*

`iparam.presolve_max_num_pass`

Control the maximum number of times presolve passes over the problem. A negative value implies MOSEK decides.

Default -1

Accepted [-inf; +inf]

Example `task.putintparam(iparam.presolve_max_num_pass, -1)`

Generic name MSK_IPAR_PRESOLVE_MAX_NUM_PASS

Groups *Presolve*

`iparam.presolve_max_num_reductions`

Controls the maximum number of reductions performed by the presolve. The value of the parameter is normally only changed in connection with debugging. A negative value implies that an infinite number of reductions are allowed.

Default -1

Accepted [-inf; +inf]

Example `task.putintparam(iparam.presolve_max_num_reductions, -1)`

Generic name MSK_IPAR_PRESOLVE_MAX_NUM_REDUCTIONS

Groups *Overall solver, Presolve*

`iparam.presolve_use`

Controls whether the presolve is applied to a problem before it is optimized.

Default *free*

Accepted *off, on, free* (see *presolvemode*)

Example `task.putintparam(iparam.presolve_use, presolvemode.free.value)`

Generic name MSK_IPAR_PRESOLVE_USE

Groups *Overall solver, Presolve*

`iparam.primal_repair_optimizer`

Controls which optimizer that is used to find the optimal repair.

Default *free*

Accepted *free, intpnt, conic, primal_simplex, dual_simplex, free_simplex, mixed_int* (see *optimizertype*)

Example `task.putintparam(iparam.primal_repair_optimizer, optimizertype.free.value)`

Generic name MSK_IPAR_PRIMAL_REPAIR_OPTIMIZER

Groups *Overall solver*

`iparam.ptf_write_transform`

If *iparam.ptf_write_transform* is *onoffkey.on*, constraint blocks with identifiable conic slacks are transformed into conic constraints and the slacks are eliminated.

Default *on*

Accepted *on, off* (see *onoffkey*)

Example `task.putintparam(iparam.ptf_write_transform, onoffkey.on.value)`

Generic name MSK_IPAR_PTF_WRITE_TRANSFORM

Groups *Data input/output*

`iparam.read_debug`

Turns on additional debugging information when reading files.

Default *off*
Accepted *on, off* (see *onoffkey*)
Example `task.putintparam(iparam.read_debug, onoffkey.off.value)`
Generic name `MSK_IPAR_READ_DEBUG`
Groups *Data input/output*

`iparam.read_keep_free_con`
Controls whether the free constraints are included in the problem.

Default *off*
Accepted

- *on*: The free constraints are kept.
- *off*: The free constraints are discarded.

Example `task.putintparam(iparam.read_keep_free_con, onoffkey.off.value)`
Generic name `MSK_IPAR_READ_KEEP_FREE_CON`
Groups *Data input/output*

`iparam.read_lp_drop_new_vars_in_bou`
If this option is turned on, **MOSEK** will drop variables that are defined for the first time in the bounds section.

Default *off*
Accepted *on, off* (see *onoffkey*)
Example `task.putintparam(iparam.read_lp_drop_new_vars_in_bou, onoffkey.off.value)`
Generic name `MSK_IPAR_READ_LP_DROP_NEW_VARS_IN_BOU`
Groups *Data input/output*

`iparam.read_lp_quoted_names`
If a name is in quotes when reading an LP file, the quotes will be removed.

Default *on*
Accepted *on, off* (see *onoffkey*)
Example `task.putintparam(iparam.read_lp_quoted_names, onoffkey.on.value)`
Generic name `MSK_IPAR_READ_LP_QUOTED_NAMES`
Groups *Data input/output*

`iparam.read_mps_format`
Controls how strictly the MPS file reader interprets the MPS format.

Default *free*
Accepted *strict, relaxed, free, cplex* (see *mpsformat*)
Example `task.putintparam(iparam.read_mps_format, mpsformat.free.value)`
Generic name `MSK_IPAR_READ_MPS_FORMAT`
Groups *Data input/output*

`iparam.read_mps_width`
Controls the maximal number of characters allowed in one line of the MPS file.

Default `1024`
Accepted `[80; +inf]`
Example `task.putintparam(iparam.read_mps_width, 1024)`
Generic name `MSK_IPAR_READ_MPS_WIDTH`
Groups *Data input/output*

`iparam.read_task_ignore_param`
Controls whether **MOSEK** should ignore the parameter setting defined in the task file and use the default parameter setting instead.

Default *off*
Accepted *on, off* (see *onoffkey*)
Example `task.putintparam(iparam.read_task_ignore_param, onoffkey.off.value)`
Generic name `MSK_IPAR_READ_TASK_IGNORE_PARAM`
Groups *Data input/output*

`iparam.remove_unused_solutions`
 Removes unused solutions before the optimization is performed.

Default *off*
Accepted *on, off* (see *onoffkey*)
Example `task.putintparam(iparam.remove_unused_solutions, onoffkey.off.value)`
Generic name `MSK_IPAR_REMOVE_UNUSED_SOLUTIONS`
Groups *Overall system*

`iparam.sensitivity_all`
 If set to *onoffkey.on*, then *Task.sensitivityreport* analyzes all bounds and variables instead of reading a specification from the file.

Default *off*
Accepted *on, off* (see *onoffkey*)
Example `task.putintparam(iparam.sensitivity_all, onoffkey.off.value)`
Generic name `MSK_IPAR_SENSITIVITY_ALL`
Groups *Overall solver*

`iparam.sensitivity_optimizer`
 Controls which optimizer is used for optimal partition sensitivity analysis.

Default *free_simplex*
Accepted *free, intpnt, conic, primal_simplex, dual_simplex, free_simplex, mixed_int* (see *optimizertype*)
Example `task.putintparam(iparam.sensitivity_optimizer, optimizertype.free_simplex.value)`
Generic name `MSK_IPAR_SENSITIVITY_OPTIMIZER`
Groups *Overall solver, Simplex optimizer*

`iparam.sensitivity_type`
 Controls which type of sensitivity analysis is to be performed.

Default *basis*
Accepted *basis* (see *sensitivitytype*)
Example `task.putintparam(iparam.sensitivity_type, sensitivitytype.basis.value)`
Generic name `MSK_IPAR_SENSITIVITY_TYPE`
Groups *Overall solver*

`iparam.sim_basis_factor_use`
 Controls whether an LU factorization of the basis is used in a hot-start. Forcing a refactorization sometimes improves the stability of the simplex optimizers, but in most cases there is a performance penalty.

Default *on*
Accepted *on, off* (see *onoffkey*)
Example `task.putintparam(iparam.sim_basis_factor_use, onoffkey.on.value)`
Generic name `MSK_IPAR_SIM_BASIS_FACTOR_USE`

Groups *Simplex optimizer*

`iparam.sim_degen`

Controls how aggressively degeneration is handled.

Default *free*

Accepted *none, free, aggressive, moderate, minimum* (see *simdegen*)

Example `task.putintparam(iparam.sim_degen, simdegen.free.value)`

Generic name MSK_IPAR_SIM_DEGEN

Groups *Simplex optimizer*

`iparam.sim_dual_crash`

Controls whether crashing is performed in the dual simplex optimizer. If this parameter is set to x , then a crash will be performed if a basis consists of more than $(100 - x) \bmod f_v$ entries, where f_v is the number of fixed variables.

Default 90

Accepted $[0; +\infty]$

Example `task.putintparam(iparam.sim_dual_crash, 90)`

Generic name MSK_IPAR_SIM_DUAL_CRASH

Groups *Dual simplex*

`iparam.sim_dual_phaseone_method`

An experimental feature.

Default 0

Accepted $[0; 10]$

Example `task.putintparam(iparam.sim_dual_phaseone_method, 0)`

Generic name MSK_IPAR_SIM_DUAL_PHASEONE_METHOD

Groups *Simplex optimizer*

`iparam.sim_dual_restrict_selection`

The dual simplex optimizer can use a so-called restricted selection/pricing strategy to choose the outgoing variable. Hence, if restricted selection is applied, then the dual simplex optimizer first choose a subset of all the potential outgoing variables. Next, for some time it will choose the outgoing variable only among the subset. From time to time the subset is redefined. A larger value of this parameter implies that the optimizer will be more aggressive in its restriction strategy, i.e. a value of 0 implies that the restriction strategy is not applied at all.

Default 50

Accepted $[0; 100]$

Example `task.putintparam(iparam.sim_dual_restrict_selection, 50)`

Generic name MSK_IPAR_SIM_DUAL_RESTRICT_SELECTION

Groups *Dual simplex*

`iparam.sim_dual_selection`

Controls the choice of the incoming variable, known as the selection strategy, in the dual simplex optimizer.

Default *free*

Accepted *free, full, ase, devex, se, partial* (see *simseltype*)

Example `task.putintparam(iparam.sim_dual_selection, simseltypes.free.value)`

Generic name MSK_IPAR_SIM_DUAL_SELECTION

Groups *Dual simplex*

`iparam.sim_exploit_dupvec`

Controls if the simplex optimizers are allowed to exploit duplicated columns.

Default *off*
Accepted *on, off, free* (see *simdupvec*)
Example `task.putintparam(iparam.sim_exploit_dupvec, simdupvec.off.value)`
Generic name `MSK_IPAR_SIM_EXPLOIT_DUPVEC`
Groups *Simplex optimizer*

`iparam.sim_hotstart`
Controls the type of hot-start that the simplex optimizer perform.

Default *free*
Accepted *none, free, status_keys* (see *simhotstart*)
Example `task.putintparam(iparam.sim_hotstart, simhotstart.free.value)`
Generic name `MSK_IPAR_SIM_HOTSTART`
Groups *Simplex optimizer*

`iparam.sim_hotstart_lu`
Determines if the simplex optimizer should exploit the initial factorization.

Default *on*
Accepted

- *on*: Factorization is reused if possible.
- *off*: Factorization is recomputed.

Example `task.putintparam(iparam.sim_hotstart_lu, onoffkey.on.value)`
Generic name `MSK_IPAR_SIM_HOTSTART_LU`
Groups *Simplex optimizer*

`iparam.sim_max_iterations`
Maximum number of iterations that can be used by a simplex optimizer.

Default 10000000
Accepted `[0; +inf]`
Example `task.putintparam(iparam.sim_max_iterations, 10000000)`
Generic name `MSK_IPAR_SIM_MAX_ITERATIONS`
Groups *Simplex optimizer, Termination criteria*

`iparam.sim_max_num_setbacks`
Controls how many set-backs are allowed within a simplex optimizer. A set-back is an event where the optimizer moves in the wrong direction. This is impossible in theory but may happen due to numerical problems.

Default 250
Accepted `[0; +inf]`
Example `task.putintparam(iparam.sim_max_num_setbacks, 250)`
Generic name `MSK_IPAR_SIM_MAX_NUM_SETBACKS`
Groups *Simplex optimizer*

`iparam.sim_non_singular`
Controls if the simplex optimizer ensures a non-singular basis, if possible.

Default *on*
Accepted *on, off* (see *onoffkey*)
Example `task.putintparam(iparam.sim_non_singular, onoffkey.on.value)`
Generic name `MSK_IPAR_SIM_NON_SINGULAR`
Groups *Simplex optimizer*

`iparam.sim_primal_crash`
Controls whether crashing is performed in the primal simplex optimizer. In general, if a basis consists of more than (100-this parameter value)% fixed variables, then a crash will be performed.

Default 90
Accepted [0; +inf]
Example `task.putintparam(iparam.sim_primal_crash, 90)`
Generic name MSK_IPAR_SIM_PRIMAL_CRASH
Groups *Primal simplex*

`iparam.sim_primal_phaseone_method`
 An experimental feature.

Default 0
Accepted [0; 10]
Example `task.putintparam(iparam.sim_primal_phaseone_method, 0)`
Generic name MSK_IPAR_SIM_PRIMAL_PHASEONE_METHOD
Groups *Simplex optimizer*

`iparam.sim_primal_restrict_selection`

The primal simplex optimizer can use a so-called restricted selection/pricing strategy to choose the outgoing variable. Hence, if restricted selection is applied, then the primal simplex optimizer first choose a subset of all the potential incoming variables. Next, for some time it will choose the incoming variable only among the subset. From time to time the subset is redefined. A larger value of this parameter implies that the optimizer will be more aggressive in its restriction strategy, i.e. a value of 0 implies that the restriction strategy is not applied at all.

Default 50
Accepted [0; 100]
Example `task.putintparam(iparam.sim_primal_restrict_selection, 50)`
Generic name MSK_IPAR_SIM_PRIMAL_RESTRICT_SELECTION
Groups *Primal simplex*

`iparam.sim_primal_selection`

Controls the choice of the incoming variable, known as the selection strategy, in the primal simplex optimizer.

Default *free*
Accepted *free, full, ase, devex, se, partial* (see *simseltype*)
Example `task.putintparam(iparam.sim_primal_selection, simseltypes.free.value)`
Generic name MSK_IPAR_SIM_PRIMAL_SELECTION
Groups *Primal simplex*

`iparam.sim_refactor_freq`

Controls how frequent the basis is refactorized. The value 0 means that the optimizer determines the best point of refactorization. It is strongly recommended NOT to change this parameter.

Default 0
Accepted [0; +inf]
Example `task.putintparam(iparam.sim_refactor_freq, 0)`
Generic name MSK_IPAR_SIM_REFACTOR_FREQ
Groups *Simplex optimizer*

`iparam.sim_reformulation`

Controls if the simplex optimizers are allowed to reformulate the problem.

Default *off*
Accepted *on, off, free, aggressive* (see *simreform*)
Example `task.putintparam(iparam.sim_reformulation, simreform.off.value)`
Generic name MSK_IPAR_SIM_REFORMULATION
Groups *Simplex optimizer*

iparam.sim_save_lu
 Controls if the LU factorization stored should be replaced with the LU factorization corresponding to the initial basis.

Default *off*
Accepted *on, off* (see *onoffkey*)
Example `task.putintparam(iparam.sim_save_lu, onoffkey.off.value)`
Generic name MSK_IPAR_SIM_SAVE_LU
Groups *Simplex optimizer*

iparam.sim_scaling
 Controls how much effort is used in scaling the problem before a simplex optimizer is used.

Default *free*
Accepted *free, none, moderate, aggressive* (see *scalingtype*)
Example `task.putintparam(iparam.sim_scaling, scalingtype.free.value)`
Generic name MSK_IPAR_SIM_SCALING
Groups *Simplex optimizer*

iparam.sim_scaling_method
 Controls how the problem is scaled before a simplex optimizer is used.

Default *pow2*
Accepted *pow2, free* (see *scalingmethod*)
Example `task.putintparam(iparam.sim_scaling_method, scalingmethod.pow2.value)`
Generic name MSK_IPAR_SIM_SCALING_METHOD
Groups *Simplex optimizer*

iparam.sim_seed
 Sets the random seed used for randomization in the simplex optimizers.

Default 23456
Accepted [0; 32749]
Example `task.putintparam(iparam.sim_seed, 23456)`
Generic name MSK_IPAR_SIM_SEED
Groups *Simplex optimizer*

iparam.sim_solve_form
 Controls whether the primal or the dual problem is solved by the primal-/dual-simplex optimizer.

Default *free*
Accepted *free, primal, dual* (see *solveform*)
Example `task.putintparam(iparam.sim_solve_form, solveform.free.value)`
Generic name MSK_IPAR_SIM_SOLVE_FORM
Groups *Simplex optimizer*

iparam.sim_stability_priority
 Controls how high priority the numerical stability should be given.

Default 50
Accepted [0; 100]
Example `task.putintparam(iparam.sim_stability_priority, 50)`
Generic name MSK_IPAR_SIM_STABILITY_PRIORITY
Groups *Simplex optimizer*

`iparam.sim_switch_optimizer`

The simplex optimizer sometimes chooses to solve the dual problem instead of the primal problem. This implies that if you have chosen to use the dual simplex optimizer and the problem is dualized, then it actually makes sense to use the primal simplex optimizer instead. If this parameter is on and the problem is dualized and furthermore the simplex optimizer is chosen to be the primal (dual) one, then it is switched to the dual (primal).

Default *off*

Accepted *on, off* (see *onoffkey*)

Example `task.putintparam(iparam.sim_switch_optimizer, onoffkey.off.value)`

Generic name `MSK_IPAR_SIM_SWITCH_OPTIMIZER`

Groups *Simplex optimizer*

`iparam.sol_filter_keep_basic`

If turned on, then basic and super basic constraints and variables are written to the solution file independent of the filter setting.

Default *off*

Accepted *on, off* (see *onoffkey*)

Example `task.putintparam(iparam.sol_filter_keep_basic, onoffkey.off.value)`

Generic name `MSK_IPAR_SOL_FILTER_KEEP_BASIC`

Groups *Solution input/output*

`iparam.sol_filter_keep_ranged`

If turned on, then ranged constraints and variables are written to the solution file independent of the filter setting.

Default *off*

Accepted *on, off* (see *onoffkey*)

Example `task.putintparam(iparam.sol_filter_keep_ranged, onoffkey.off.value)`

Generic name `MSK_IPAR_SOL_FILTER_KEEP_RANGED`

Groups *Solution input/output*

`iparam.sol_read_name_width`

When a solution is read by **MOSEK** and some constraint, variable or cone names contain blanks, then a maximum name width must be specified. A negative value implies that no name contain blanks.

Default `-1`

Accepted `[-inf; +inf]`

Example `task.putintparam(iparam.sol_read_name_width, -1)`

Generic name `MSK_IPAR_SOL_READ_NAME_WIDTH`

Groups *Data input/output, Solution input/output*

`iparam.sol_read_width`

Controls the maximal acceptable width of line in the solutions when read by **MOSEK**.

Default `1024`

Accepted `[80; +inf]`

Example `task.putintparam(iparam.sol_read_width, 1024)`

Generic name `MSK_IPAR_SOL_READ_WIDTH`

Groups *Data input/output, Solution input/output*

`iparam.solution_callback`

Indicates whether solution callbacks will be performed during the optimization.

Default *off*
Accepted *on, off* (see *onoffkey*)
Example `task.putintparam(iparam.solution_callback, onoffkey.off.value)`
Generic name MSK_IPAR_SOLUTION_CALLBACK
Groups *Progress callback, Overall solver*

`iparam.timing_level`
 Controls the amount of timing performed inside MOSEK.

Default 1
Accepted [0; +inf]
Example `task.putintparam(iparam.timing_level, 1)`
Generic name MSK_IPAR_TIMING_LEVEL
Groups *Overall system*

`iparam.write_bas_constraints`
 Controls whether the constraint section is written to the basic solution file.

Default *on*
Accepted *on, off* (see *onoffkey*)
Example `task.putintparam(iparam.write_bas_constraints, onoffkey.on.value)`
Generic name MSK_IPAR_WRITE_BAS_CONSTRAINTS
Groups *Data input/output, Solution input/output*

`iparam.write_bas_head`
 Controls whether the header section is written to the basic solution file.

Default *on*
Accepted *on, off* (see *onoffkey*)
Example `task.putintparam(iparam.write_bas_head, onoffkey.on.value)`
Generic name MSK_IPAR_WRITE_BAS_HEAD
Groups *Data input/output, Solution input/output*

`iparam.write_bas_variables`
 Controls whether the variables section is written to the basic solution file.

Default *on*
Accepted *on, off* (see *onoffkey*)
Example `task.putintparam(iparam.write_bas_variables, onoffkey.on.value)`
Generic name MSK_IPAR_WRITE_BAS_VARIABLES
Groups *Data input/output, Solution input/output*

`iparam.write_compression`
 Controls whether the data file is compressed while it is written. 0 means no compression while higher values mean more compression.

Default 9
Accepted [0; +inf]
Example `task.putintparam(iparam.write_compression, 9)`
Generic name MSK_IPAR_WRITE_COMPRESSION
Groups *Data input/output*

`iparam.write_data_param`
 If this option is turned on the parameter settings are written to the data file as parameters.

Default *off*
Accepted *on, off* (see *onoffkey*)

Example `task.putintparam(iparam.write_data_param, onoffkey.off.value)`
Generic name `MSK_IPAR_WRITE_DATA_PARAM`
Groups *Data input/output*

`iparam.write_free_con`

Controls whether the free constraints are written to the data file.

Default *on*

Accepted *on, off* (see *onoffkey*)

Example `task.putintparam(iparam.write_free_con, onoffkey.on.value)`

Generic name `MSK_IPAR_WRITE_FREE_CON`

Groups *Data input/output*

`iparam.write_generic_names`

Controls whether generic names should be used instead of user-defined names when writing to the data file.

Default *off*

Accepted *on, off* (see *onoffkey*)

Example `task.putintparam(iparam.write_generic_names, onoffkey.off.value)`

Generic name `MSK_IPAR_WRITE_GENERIC_NAMES`

Groups *Data input/output*

`iparam.write_generic_names_io`

Index origin used in generic names.

Default *1*

Accepted *[0; +inf]*

Example `task.putintparam(iparam.write_generic_names_io, 1)`

Generic name `MSK_IPAR_WRITE_GENERIC_NAMES_IO`

Groups *Data input/output*

`iparam.write_ignore_incompatible_items`

Controls if the writer ignores incompatible problem items when writing files.

Default *off*

Accepted

- *on*: Ignore items that cannot be written to the current output file format.
- *off*: Produce an error if the problem contains items that cannot be written to the current output file format.

Example `task.putintparam(iparam.write_ignore_incompatible_items, onoffkey.off.value)`

Generic name `MSK_IPAR_WRITE_IGNORE_INCOMPATIBLE_ITEMS`

Groups *Data input/output*

`iparam.write_int_constraints`

Controls whether the constraint section is written to the integer solution file.

Default *on*

Accepted *on, off* (see *onoffkey*)

Example `task.putintparam(iparam.write_int_constraints, onoffkey.on.value)`

Generic name `MSK_IPAR_WRITE_INT_CONSTRAINTS`

Groups *Data input/output, Solution input/output*

`iparam.write_int_head`

Controls whether the header section is written to the integer solution file.

Default *on*
Accepted *on, off* (see *onoffkey*)
Example `task.putintparam(iparam.write_int_head, onoffkey.on.value)`
Generic name `MSK_IPAR_WRITE_INT_HEAD`
Groups *Data input/output, Solution input/output*

`iparam.write_int_variables`

Controls whether the variables section is written to the integer solution file.

Default *on*
Accepted *on, off* (see *onoffkey*)
Example `task.putintparam(iparam.write_int_variables, onoffkey.on.value)`
Generic name `MSK_IPAR_WRITE_INT_VARIABLES`
Groups *Data input/output, Solution input/output*

`iparam.write_lp_full_obj`

Write all variables, including the ones with 0-coefficients, in the objective.

Default *on*
Accepted *on, off* (see *onoffkey*)
Example `task.putintparam(iparam.write_lp_full_obj, onoffkey.on.value)`
Generic name `MSK_IPAR_WRITE_LP_FULL_OBJ`
Groups *Data input/output*

`iparam.write_lp_line_width`

Maximum width of line in an LP file written by **MOSEK**.

Default 80
Accepted [40; +inf]
Example `task.putintparam(iparam.write_lp_line_width, 80)`
Generic name `MSK_IPAR_WRITE_LP_LINE_WIDTH`
Groups *Data input/output*

`iparam.write_lp_quoted_names`

If this option is turned on, then **MOSEK** will quote invalid LP names when writing an LP file.

Default *on*
Accepted *on, off* (see *onoffkey*)
Example `task.putintparam(iparam.write_lp_quoted_names, onoffkey.on.value)`
Generic name `MSK_IPAR_WRITE_LP_QUOTED_NAMES`
Groups *Data input/output*

`iparam.write_lp_strict_format`

Controls whether LP output files satisfy the LP format strictly.

Default *off*
Accepted *on, off* (see *onoffkey*)
Example `task.putintparam(iparam.write_lp_strict_format, onoffkey.off.value)`
Generic name `MSK_IPAR_WRITE_LP_STRICT_FORMAT`
Groups *Data input/output*

`iparam.write_lp_terms_per_line`

Maximum number of terms on a single line in an LP file written by **MOSEK**. 0 means unlimited.

Default 10
Accepted [0; +inf]

Example `task.putintparam(iparam.write_lp_terms_per_line, 10)`

Generic name `MSK_IPAR_WRITE_LP_TERMS_PER_LINE`

Groups *Data input/output*

`iparam.write_mps_format`

Controls in which format the MPS is written.

Default *free*

Accepted *strict, relaxed, free, cplex* (see *mpsformat*)

Example `task.putintparam(iparam.write_mps_format, mpsformat.free.value)`

Generic name `MSK_IPAR_WRITE_MPS_FORMAT`

Groups *Data input/output*

`iparam.write_mps_int`

Controls if marker records are written to the MPS file to indicate whether variables are integer restricted.

Default *on*

Accepted *on, off* (see *onoffkey*)

Example `task.putintparam(iparam.write_mps_int, onoffkey.on.value)`

Generic name `MSK_IPAR_WRITE_MPS_INT`

Groups *Data input/output*

`iparam.write_precision`

Controls the precision with which double numbers are printed in the MPS data file. In general it is not worthwhile to use a value higher than 15.

Default 15

Accepted `[0; +inf]`

Example `task.putintparam(iparam.write_precision, 15)`

Generic name `MSK_IPAR_WRITE_PRECISION`

Groups *Data input/output*

`iparam.write_sol_barvariables`

Controls whether the symmetric matrix variables section is written to the solution file.

Default *on*

Accepted *on, off* (see *onoffkey*)

Example `task.putintparam(iparam.write_sol_barvariables, onoffkey.on.value)`

Generic name `MSK_IPAR_WRITE_SOL_BARVARIABLES`

Groups *Data input/output, Solution input/output*

`iparam.write_sol_constraints`

Controls whether the constraint section is written to the solution file.

Default *on*

Accepted *on, off* (see *onoffkey*)

Example `task.putintparam(iparam.write_sol_constraints, onoffkey.on.value)`

Generic name `MSK_IPAR_WRITE_SOL_CONSTRAINTS`

Groups *Data input/output, Solution input/output*

`iparam.write_sol_head`

Controls whether the header section is written to the solution file.

Default *on*

Accepted *on, off* (see *onoffkey*)

Example `task.putintparam(iparam.write_sol_head, onoffkey.on.value)`

Generic name MSK_IPAR_WRITE_SOL_HEAD

Groups *Data input/output, Solution input/output*

`iparam.write_sol_ignore_invalid_names`

Even if the names are invalid MPS names, then they are employed when writing the solution file.

Default *off*

Accepted *on, off* (see *onoffkey*)

Example `task.putintparam(iparam.write_sol_ignore_invalid_names,
onoffkey.off.value)`

Generic name MSK_IPAR_WRITE_SOL_IGNORE_INVALID_NAMES

Groups *Data input/output, Solution input/output*

`iparam.write_sol_variables`

Controls whether the variables section is written to the solution file.

Default *on*

Accepted *on, off* (see *onoffkey*)

Example `task.putintparam(iparam.write_sol_variables, onoffkey.on.value)`

Generic name MSK_IPAR_WRITE_SOL_VARIABLES

Groups *Data input/output, Solution input/output*

`iparam.write_task_inc_sol`

Controls whether the solutions are stored in the task file too.

Default *on*

Accepted *on, off* (see *onoffkey*)

Example `task.putintparam(iparam.write_task_inc_sol, onoffkey.on.value)`

Generic name MSK_IPAR_WRITE_TASK_INC_SOL

Groups *Data input/output*

`iparam.write_xml_mode`

Controls if linear coefficients should be written by row or column when writing in the XML file format.

Default *row*

Accepted *row, col* (see *xmlwriteroutputtype*)

Example `task.putintparam(iparam.write_xml_mode, xmlwriteroutputtype.
row.value)`

Generic name MSK_IPAR_WRITE_XML_MODE

Groups *Data input/output*

15.7.3 String parameters

`sparam`

The enumeration type containing all string parameters.

`sparam.bas_sol_file_name`

Name of the bas solution file.

Accepted Any valid file name.

Example `task.putstrparam(sparam.bas_sol_file_name, "somevalue")`

Generic name MSK_SPAR_BAS_SOL_FILE_NAME

Groups *Data input/output, Solution input/output*

`sparam.data_file_name`

Data are read and written to this file.

Accepted Any valid file name.

Example `task.putstrparam(sparam.data_file_name, "somevalue")`

Generic name MSK_SPAR_DATA_FILE_NAME

Groups *Data input/output*

sparam.debug_file_name

MOSEK debug file.

Accepted Any valid file name.

Example task.putstrparam(sparam.debug_file_name, "somevalue")

Generic name MSK_SPAR_DEBUG_FILE_NAME

Groups *Data input/output*

sparam.int_sol_file_name

Name of the int solution file.

Accepted Any valid file name.

Example task.putstrparam(sparam.int_sol_file_name, "somevalue")

Generic name MSK_SPAR_INT_SOL_FILE_NAME

Groups *Data input/output, Solution input/output*

sparam.itr_sol_file_name

Name of the itr solution file.

Accepted Any valid file name.

Example task.putstrparam(sparam.itr_sol_file_name, "somevalue")

Generic name MSK_SPAR_ITR_SOL_FILE_NAME

Groups *Data input/output, Solution input/output*

sparam.mio_debug_string

For internal debugging purposes.

Accepted Any valid string.

Example task.putstrparam(sparam.mio_debug_string, "somevalue")

Generic name MSK_SPAR_MIO_DEBUG_STRING

Groups *Data input/output*

sparam.param_comment_sign

Only the first character in this string is used. It is considered as a start of comment sign in the MOSEK parameter file. Spaces are ignored in the string.

Default

%%

Accepted Any valid string.

Example task.putstrparam(sparam.param_comment_sign, "%")

Generic name MSK_SPAR_PARAM_COMMENT_SIGN

Groups *Data input/output*

sparam.param_read_file_name

Modifications to the parameter database is read from this file.

Accepted Any valid file name.

Example task.putstrparam(sparam.param_read_file_name, "somevalue")

Generic name MSK_SPAR_PARAM_READ_FILE_NAME

Groups *Data input/output*

sparam.param_write_file_name

The parameter database is written to this file.

Accepted Any valid file name.

Example task.putstrparam(sparam.param_write_file_name, "somevalue")

Generic name MSK_SPAR_PARAM_WRITE_FILE_NAME

Groups *Data input/output*

`sparam.read_mps_bou_name`

Name of the BOUNDS vector used. An empty name means that the first BOUNDS vector is used.

Accepted Any valid MPS name.

Example `task.putstrparam(sparam.read_mps_bou_name, "somevalue")`

Generic name `MSK_SPAR_READ_MPS_BOU_NAME`

Groups *Data input/output*

`sparam.read_mps_obj_name`

Name of the free constraint used as objective function. An empty name means that the first constraint is used as objective function.

Accepted Any valid MPS name.

Example `task.putstrparam(sparam.read_mps_obj_name, "somevalue")`

Generic name `MSK_SPAR_READ_MPS_OBJ_NAME`

Groups *Data input/output*

`sparam.read_mps_ran_name`

Name of the RANGE vector used. An empty name means that the first RANGE vector is used.

Accepted Any valid MPS name.

Example `task.putstrparam(sparam.read_mps_ran_name, "somevalue")`

Generic name `MSK_SPAR_READ_MPS_RAN_NAME`

Groups *Data input/output*

`sparam.read_mps_rhs_name`

Name of the RHS used. An empty name means that the first RHS vector is used.

Accepted Any valid MPS name.

Example `task.putstrparam(sparam.read_mps_rhs_name, "somevalue")`

Generic name `MSK_SPAR_READ_MPS_RHS_NAME`

Groups *Data input/output*

`sparam.remote_access_token`

An access token used to submit tasks to a remote **MOSEK** server. An access token is a random 32-byte string encoded in base64, i.e. it is a 44 character ASCII string.

Accepted Any valid string.

Example `task.putstrparam(sparam.remote_access_token, "somevalue")`

Generic name `MSK_SPAR_REMOTE_ACCESS_TOKEN`

Groups *Overall system*

`sparam.sensitivity_file_name`

If defined *Task.sensitivityreport* reads this file as a sensitivity analysis data file specifying the type of analysis to be done.

Accepted Any valid string.

Example `task.putstrparam(sparam.sensitivity_file_name, "somevalue")`

Generic name `MSK_SPAR_SENSITIVITY_FILE_NAME`

Groups *Data input/output*

`sparam.sensitivity_res_file_name`

If this is a nonempty string, then *Task.sensitivityreport* writes results to this file.

Accepted Any valid string.

Example `task.putstrparam(sparam.sensitivity_res_file_name, "somevalue")`

Generic name `MSK_SPAR_SENSITIVITY_RES_FILE_NAME`

Groups *Data input/output*

`sparam.sol_filter_xc_low`

A filter used to determine which constraints should be listed in the solution file. A value of 0.5 means that all constraints having $xc[i] > 0.5$ should be listed, whereas +0.5 means that all constraints having $xc[i] \geq blc[i] + 0.5$ should be listed. An empty filter means that no filter is applied.

Accepted Any valid filter.

Example `task.putstrparam(sparam.sol_filter_xc_low, "somevalue")`

Generic name MSK_SPAR_SOL_FILTER_XC_LOW

Groups *Data input/output, Solution input/output*

`sparam.sol_filter_xc_upr`

A filter used to determine which constraints should be listed in the solution file. A value of 0.5 means that all constraints having $xc[i] < 0.5$ should be listed, whereas -0.5 means all constraints having $xc[i] \leq buc[i] - 0.5$ should be listed. An empty filter means that no filter is applied.

Accepted Any valid filter.

Example `task.putstrparam(sparam.sol_filter_xc_upr, "somevalue")`

Generic name MSK_SPAR_SOL_FILTER_XC_UPR

Groups *Data input/output, Solution input/output*

`sparam.sol_filter_xx_low`

A filter used to determine which variables should be listed in the solution file. A value of "0.5" means that all constraints having $xx[j] \geq 0.5$ should be listed, whereas "+0.5" means that all constraints having $xx[j] \geq blx[j] + 0.5$ should be listed. An empty filter means no filter is applied.

Accepted Any valid filter.

Example `task.putstrparam(sparam.sol_filter_xx_low, "somevalue")`

Generic name MSK_SPAR_SOL_FILTER_XX_LOW

Groups *Data input/output, Solution input/output*

`sparam.sol_filter_xx_upr`

A filter used to determine which variables should be listed in the solution file. A value of "0.5" means that all constraints having $xx[j] < 0.5$ should be printed, whereas "-0.5" means all constraints having $xx[j] \leq bux[j] - 0.5$ should be listed. An empty filter means no filter is applied.

Accepted Any valid file name.

Example `task.putstrparam(sparam.sol_filter_xx_upr, "somevalue")`

Generic name MSK_SPAR_SOL_FILTER_XX_UPR

Groups *Data input/output, Solution input/output*

`sparam.stat_file_name`

Statistics file name.

Accepted Any valid file name.

Example `task.putstrparam(sparam.stat_file_name, "somevalue")`

Generic name MSK_SPAR_STAT_FILE_NAME

Groups *Data input/output*

`sparam.stat_key`

Key used when writing the summary file.

Accepted Any valid string.

Example `task.putstrparam(sparam.stat_key, "somevalue")`

Generic name MSK_SPAR_STAT_KEY

Groups *Data input/output*

`sparam.stat_name`

Name used when writing the statistics file.

Accepted Any valid XML string.

Example `task.putstrparam(sparam.stat_name, "somevalue")`

Generic name `MSK_SPAR_STAT_NAME`

Groups *Data input/output*

`sparam.write_lp_gen_var_name`

Sometimes when an LP file is written additional variables must be inserted. They will have the prefix denoted by this parameter.

Default `xmskgen`

Accepted Any valid string.

Example `task.putstrparam(sparam.write_lp_gen_var_name, "xmskgen")`

Generic name `MSK_SPAR_WRITE_LP_GEN_VAR_NAME`

Groups *Data input/output*

15.8 Response codes

Response codes include:

- *Termination codes*
- *Warnings*
- *Errors*

The numerical code (in brackets) identifies the response in error messages and in the log output.

`rescode`

The enumeration type containing all response codes.

15.8.1 Termination

`rescode.ok (0)`

No error occurred.

`rescode.trm_max_iterations (10000)`

The optimizer terminated at the maximum number of iterations.

`rescode.trm_max_time (10001)`

The optimizer terminated at the maximum amount of time.

`rescode.trm_objective_range (10002)`

The optimizer terminated with an objective value outside the objective range.

`rescode.trm_mio_num_relaxs (10008)`

The mixed-integer optimizer terminated as the maximum number of relaxations was reached.

`rescode.trm_mio_num_branches (10009)`

The mixed-integer optimizer terminated as the maximum number of branches was reached.

`rescode.trm_num_max_num_int_solutions (10015)`

The mixed-integer optimizer terminated as the maximum number of feasible solutions was reached.

`rescode.trm_stall (10006)`

The optimizer is terminated due to slow progress.

Stalling means that numerical problems prevent the optimizer from making reasonable progress and that it makes no sense to continue. In many cases this happens if the problem is badly scaled or otherwise ill-conditioned. There is no guarantee that the solution will be feasible or optimal. However, often stalling happens near the optimum, and the returned solution may be of good quality. Therefore, it is recommended to check the status of the solution. If the solution status is optimal the solution is most likely good enough for most practical purposes.

Please note that if a linear optimization problem is solved using the interior-point optimizer with basis identification turned on, the returned basic solution likely to have high accuracy, even though the optimizer stalled.

Some common causes of stalling are a) badly scaled models, b) near feasible or near infeasible problems.

`rescode.trm_user_callback` (10007)
 The optimizer terminated due to the return of the user-defined callback function.

`rescode.trm_max_num_setbacks` (10020)
 The optimizer terminated as the maximum number of set-backs was reached. This indicates serious numerical problems and a possibly badly formulated problem.

`rescode.trm_numerical_problem` (10025)
 The optimizer terminated due to numerical problems.

`rescode.trm_internal` (10030)
 The optimizer terminated due to some internal reason. Please contact **MOSEK** support.

`rescode.trm_internal_stop` (10031)
 The optimizer terminated for internal reasons. Please contact **MOSEK** support.

15.8.2 Warnings

`rescode.wrn_open_param_file` (50)
 The parameter file could not be opened.

`rescode.wrn_large_bound` (51)
 A numerically large bound value is specified.

`rescode.wrn_large_lo_bound` (52)
 A numerically large lower bound value is specified.

`rescode.wrn_large_up_bound` (53)
 A numerically large upper bound value is specified.

`rescode.wrn_large_con_fx` (54)
 An equality constraint is fixed to a numerically large value. This can cause numerical problems.

`rescode.wrn_large_cj` (57)
 A numerically large value is specified for one c_j .

`rescode.wrn_large_aij` (62)
 A numerically large value is specified for an $a_{i,j}$ element in A . The parameter `dparam.data_tol_aij_large` controls when an $a_{i,j}$ is considered large.

`rescode.wrn_zero_aij` (63)
 One or more zero elements are specified in A .

`rescode.wrn_name_max_len` (65)
 A name is longer than the buffer that is supposed to hold it.

`rescode.wrn_spar_max_len` (66)
 A value for a string parameter is longer than the buffer that is supposed to hold it.

`rescode.wrn_mps_split_rhs_vector` (70)
 An RHS vector is split into several nonadjacent parts in an MPS file.

`rescode.wrn_mps_split_ran_vector` (71)
 A RANGE vector is split into several nonadjacent parts in an MPS file.

`rescode.wrn_mps_split_bou_vector` (72)
 A BOUNDS vector is split into several nonadjacent parts in an MPS file.

`rescode.wrn_lp_old_quad_format` (80)
 Missing `'/2'` after quadratic expressions in bound or objective.

`rescode.wrn_lp_drop_variable` (85)
 Ignored a variable because the variable was not previously defined. Usually this implies that a variable appears in the bound section but not in the objective or the constraints.

`rescode.wrn_nz_in_upr_tri` (200)
 Non-zero elements specified in the upper triangle of a matrix were ignored.

`rescode.wrn_dropped_nz_qobj` (201)
 One or more non-zero elements were dropped in the Q matrix in the objective.

`rescode.wrn_ignore_integer` (250)
 Ignored integer constraints.

`rescode.wrn_no_global_optimizer` (251)
 No global optimizer is available.

`rescode.wrn_mio_infeasible_final` (270)
 The final mixed-integer problem with all the integer variables fixed at their optimal values is infeasible.

`rescode.wrn_sol_filter (300)`
 Invalid solution filter is specified.

`rescode.wrn_undef_sol_file_name (350)`
 Undefined name occurred in a solution.

`rescode.wrn_sol_file_ignored_con (351)`
 One or more lines in the constraint section were ignored when reading a solution file.

`rescode.wrn_sol_file_ignored_var (352)`
 One or more lines in the variable section were ignored when reading a solution file.

`rescode.wrn_too_few_basis_vars (400)`
 An incomplete basis has been specified. Too few basis variables are specified.

`rescode.wrn_too_many_basis_vars (405)`
 A basis with too many variables has been specified.

`rescode.wrn_license_expire (500)`
 The license expires.

`rescode.wrn_license_server (501)`
 The license server is not responding.

`rescode.wrn_empty_name (502)`
 A variable or constraint name is empty. The output file may be invalid.

`rescode.wrn_using_generic_names (503)`
 Generic names are used because a name is not valid. For instance when writing an LP file the names must not contain blanks or start with a digit.

`rescode.wrn_license_feature_expire (505)`
 The license expires.

`rescode.wrn_param_name_dou (510)`
 The parameter name is not recognized as a double parameter.

`rescode.wrn_param_name_int (511)`
 The parameter name is not recognized as a integer parameter.

`rescode.wrn_param_name_str (512)`
 The parameter name is not recognized as a string parameter.

`rescode.wrn_param_str_value (515)`
 The string is not recognized as a symbolic value for the parameter.

`rescode.wrn_param_ignored_cmio (516)`
 A parameter was ignored by the conic mixed integer optimizer.

`rescode.wrn_zeros_in_sparse_row (705)`
 One or more (near) zero elements are specified in a sparse row of a matrix. Since, it is redundant to specify zero elements then it may indicate an error.

`rescode.wrn_zeros_in_sparse_col (710)`
 One or more (near) zero elements are specified in a sparse column of a matrix. It is redundant to specify zero elements. Hence, it may indicate an error.

`rescode.wrn_incomplete_linear_dependency_check (800)`
 The linear dependency check(s) is incomplete. Normally this is not an important warning unless the optimization problem has been formulated with linear dependencies. Linear dependencies may prevent **MOSEK** from solving the problem.

`rescode.wrn_eliminator_space (801)`
 The eliminator is skipped at least once due to lack of space.

`rescode.wrn_presolve_outofspace (802)`
 The presolve is incomplete due to lack of space.

`rescode.wrn_write_changed_names (803)`
 Some names were changed because they were invalid for the output file format.

`rescode.wrn_write_discarded_cfix (804)`
 The fixed objective term could not be converted to a variable and was discarded in the output file.

`rescode.wrn_duplicate_constraint_names (850)`
 Two constraint names are identical.

`rescode.wrn_duplicate_variable_names (851)`
 Two variable names are identical.

`rescode.wrn_duplicate_barvariable_names (852)`
 Two barvariable names are identical.

rescode.wrn_duplicate_cone_names (853)
Two cone names are identical.

rescode.wrn_ana_large_bounds (900)
This warning is issued by the problem analyzer, if one or more constraint or variable bounds are very large. One should consider omitting these bounds entirely by setting them to $+\text{inf}$ or $-\text{inf}$.

rescode.wrn_ana_c_zero (901)
This warning is issued by the problem analyzer, if the coefficients in the linear part of the objective are all zero.

rescode.wrn_ana_empty_cols (902)
This warning is issued by the problem analyzer, if columns, in which all coefficients are zero, are found.

rescode.wrn_ana_close_bounds (903)
This warning is issued by problem analyzer, if ranged constraints or variables with very close upper and lower bounds are detected. One should consider treating such constraints as equalities and such variables as constants.

rescode.wrn_ana_almost_int_bounds (904)
This warning is issued by the problem analyzer if a constraint is bound nearly integral.

rescode.wrn_quad_cones_with_root_fixed_at_zero (930)
For at least one quadratic cone the root is fixed at (nearly) zero. This may cause problems such as a very large dual solution. Therefore, it is recommended to remove such cones before optimizing the problem, or to fix all the variables in the cone to 0.

rescode.wrn_rquad_cones_with_root_fixed_at_zero (931)
For at least one rotated quadratic cone at least one of the root variables are fixed at (nearly) zero. This may cause problems such as a very large dual solution. Therefore, it is recommended to remove such cones before optimizing the problem, or to fix all the variables in the cone to 0.

rescode.wrn_exp_cones_with_variables_fixed_at_zero (932)
For at least one exponential cone $x \geq y \exp(z/y)$ either the variable x or y is fixed at (nearly) zero. This may cause problems such as a very large dual solution. Therefore, it is recommended to remove such cones before optimizing the problem, or to fix all the variables in the cone to 0.

rescode.wrn_pow_cones_with_root_fixed_at_zero (933)
For at least one power cone at least one of the root variables are fixed at (nearly) zero. This may cause problems such as a very large dual solution. Therefore, it is recommended to remove such cones before optimizing the problem, or to fix all the variables in the cone to 0.

rescode.wrn_no_dualizer (950)
No automatic dualizer is available for the specified problem. The primal problem is solved.

rescode.wrn_sym_mat_large (960)
A numerically large value is specified for an $e_{i,j}$ element in E . The parameter `dparam.data_sym_mat_tol_large` controls when an $e_{i,j}$ is considered large.

15.8.3 Errors

rescode.err_license (1000)
Invalid license.

rescode.err_license_expired (1001)
The license has expired.

rescode.err_license_version (1002)
The license is valid for another version of **MOSEK**.

rescode.err_size_license (1005)
The problem is bigger than the license.

rescode.err_prob_license (1006)
The software is not licensed to solve the problem.

rescode.err_file_license (1007)
Invalid license file.

rescode.err_missing_license_file (1008)
MOSEK cannot find license file or a token server. See the **MOSEK** licensing manual for details.

rescode.err_size_license_con (1010)
The problem has too many constraints to be solved with the available license.

`rescode.err_size_license_var (1011)`
The problem has too many variables to be solved with the available license.

`rescode.err_size_license_intvar (1012)`
The problem contains too many integer variables to be solved with the available license.

`rescode.err_optimizer_license (1013)`
The optimizer required is not licensed.

`rescode.err_flexlm (1014)`
The FLEXlm license manager reported an error.

`rescode.err_license_server (1015)`
The license server is not responding.

`rescode.err_license_max (1016)`
Maximum number of licenses is reached.

`rescode.err_license_moseklm_daemon (1017)`
The MOSEKLM license manager daemon is not up and running.

`rescode.err_license_feature (1018)`
A requested feature is not available in the license file(s). Most likely due to an incorrect license system setup.

`rescode.err_platform_not_licensed (1019)`
A requested license feature is not available for the required platform.

`rescode.err_license_cannot_allocate (1020)`
The license system cannot allocate the memory required.

`rescode.err_license_cannot_connect (1021)`
MOSEK cannot connect to the license server. Most likely the license server is not up and running.

`rescode.err_license_invalid_hostid (1025)`
The host ID specified in the license file does not match the host ID of the computer.

`rescode.err_license_server_version (1026)`
The version specified in the checkout request is greater than the highest version number the daemon supports.

`rescode.err_license_no_server_support (1027)`
The license server does not support the requested feature. Possible reasons for this error include:

- The feature has expired.
- The feature's start date is later than today's date.
- The version requested is higher than feature's the highest supported version.
- A corrupted license file.

Try restarting the license and inspect the license server debug file, usually called `lmgrd.log`.

`rescode.err_license_no_server_line (1028)`
There is no **SERVER** line in the license file. All non-zero license count features need at least one **SERVER** line.

`rescode.err_older_dll (1035)`
The dynamic link library is older than the specified version.

`rescode.err_newer_dll (1036)`
The dynamic link library is newer than the specified version.

`rescode.err_link_file_dll (1040)`
A file cannot be linked to a stream in the DLL version.

`rescode.err_thread_mutex_init (1045)`
Could not initialize a mutex.

`rescode.err_thread_mutex_lock (1046)`
Could not lock a mutex.

`rescode.err_thread_mutex_unlock (1047)`
Could not unlock a mutex.

`rescode.err_thread_create (1048)`
Could not create a thread. This error may occur if a large number of environments are created and not deleted again. In any case it is a good practice to minimize the number of environments created.

`rescode.err_thread_cond_init (1049)`
Could not initialize a condition.

```

rescode.err_unknown (1050)
    Unknown error.
rescode.err_space (1051)
    Out of space.
rescode.err_file_open (1052)
    Error while opening a file.
rescode.err_file_read (1053)
    File read error.
rescode.err_file_write (1054)
    File write error.
rescode.err_data_file_ext (1055)
    The data file format cannot be determined from the file name.
rescode.err_invalid_file_name (1056)
    An invalid file name has been specified.
rescode.err_invalid_sol_file_name (1057)
    An invalid file name has been specified.
rescode.err_end_of_file (1059)
    End of file reached.
rescode.err_null_env (1060)
    env is a NULL pointer.
rescode.err_null_task (1061)
    task is a NULL pointer.
rescode.err_invalid_stream (1062)
    An invalid stream is referenced.
rescode.err_no_init_env (1063)
    env is not initialized.
rescode.err_invalid_task (1064)
    The task is invalid.
rescode.err_null_pointer (1065)
    An argument to a function is unexpectedly a NULL pointer.
rescode.err_living_tasks (1066)
    All tasks associated with an enviroment must be deleted before the environment is deleted. There
    are still some undeleted tasks.
rescode.err_blank_name (1070)
    An all blank name has been specified.
rescode.err_dup_name (1071)
    The same name was used multiple times for the same problem item type.
rescode.err_format_string (1072)
    The name format string is invalid.
rescode.err_invalid_obj_name (1075)
    An invalid objective name is specified.
rescode.err_invalid_con_name (1076)
    An invalid constraint name is used.
rescode.err_invalid_var_name (1077)
    An invalid variable name is used.
rescode.err_invalid_cone_name (1078)
    An invalid cone name is used.
rescode.err_invalid_barvar_name (1079)
    An invalid symmetric matrix variable name is used.
rescode.err_space_leaking (1080)
    MOSEK is leaking memory. This can be due to either an incorrect use of MOSEK or a bug.
rescode.err_space_no_info (1081)
    No available information about the space usage.
rescode.err_read_format (1090)
    The specified format cannot be read.
rescode.err_mps_file (1100)
    An error occurred while reading an MPS file.

```

rescode.err_mps_inv_field (1101)
 A field in the MPS file is invalid. Probably it is too wide.

rescode.err_mps_inv_marker (1102)
 An invalid marker has been specified in the MPS file.

rescode.err_mps_null_con_name (1103)
 An empty constraint name is used in an MPS file.

rescode.err_mps_null_var_name (1104)
 An empty variable name is used in an MPS file.

rescode.err_mps_undef_con_name (1105)
 An undefined constraint name occurred in an MPS file.

rescode.err_mps_undef_var_name (1106)
 An undefined variable name occurred in an MPS file.

rescode.err_mps_inv_con_key (1107)
 An invalid constraint key occurred in an MPS file.

rescode.err_mps_inv_bound_key (1108)
 An invalid bound key occurred in an MPS file.

rescode.err_mps_inv_sec_name (1109)
 An invalid section name occurred in an MPS file.

rescode.err_mps_no_objective (1110)
 No objective is defined in an MPS file.

rescode.err_mps_splitting_var (1111)
 All elements in a column of the A matrix must be specified consecutively. Hence, it is illegal to specify non-zero elements in A for variable 1, then for variable 2 and then variable 1 again.

rescode.err_mps_mul_con_name (1112)
 A constraint name was specified multiple times in the ROWS section.

rescode.err_mps_mul_qsec (1113)
 Multiple QSECTIONs are specified for a constraint in the MPS data file.

rescode.err_mps_mul_qobj (1114)
 The Q term in the objective is specified multiple times in the MPS data file.

rescode.err_mps_inv_sec_order (1115)
 The sections in the MPS data file are not in the correct order.

rescode.err_mps_mul_csec (1116)
 Multiple CSECTIONs are given the same name.

rescode.err_mps_cone_type (1117)
 Invalid cone type specified in a CSECTION.

rescode.err_mps_cone_overlap (1118)
 A variable is specified to be a member of several cones.

rescode.err_mps_cone_repeat (1119)
 A variable is repeated within the CSECTION.

rescode.err_mps_non_symmetric_q (1120)
 A non symmetric matrix has been specified.

rescode.err_mps_duplicate_q_element (1121)
 Duplicate elements is specified in a Q matrix.

rescode.err_mps_invalid_objsense (1122)
 An invalid objective sense is specified.

rescode.err_mps_tab_in_field2 (1125)
 A tab char occurred in field 2.

rescode.err_mps_tab_in_field3 (1126)
 A tab char occurred in field 3.

rescode.err_mps_tab_in_field5 (1127)
 A tab char occurred in field 5.

rescode.err_mps_invalid_obj_name (1128)
 An invalid objective name is specified.

rescode.err_lp_incompatible (1150)
 The problem cannot be written to an LP formatted file.

rescode.err_lp_empty (1151)
 The problem cannot be written to an LP formatted file.

rescode.err_lp_dup_slack_name (1152)
The name of the slack variable added to a ranged constraint already exists.

rescode.err_write_mps_invalid_name (1153)
An invalid name is created while writing an MPS file. Usually this will make the MPS file unreadable.

rescode.err_lp_invalid_var_name (1154)
A variable name is invalid when used in an LP formatted file.

rescode.err_lp_free_constraint (1155)
Free constraints cannot be written in LP file format.

rescode.err_write_opf_invalid_var_name (1156)
Empty variable names cannot be written to OPF files.

rescode.err_lp_file_format (1157)
Syntax error in an LP file.

rescode.err_write_lp_format (1158)
Problem cannot be written as an LP file.

rescode.err_read_lp_missing_end_tag (1159)
Syntax error in LP file. Possibly missing End tag.

rescode.err_lp_format (1160)
Syntax error in an LP file.

rescode.err_write_lp_non_unique_name (1161)
An auto-generated name is not unique.

rescode.err_read_lp_nonexisting_name (1162)
A variable never occurred in objective or constraints.

rescode.err_lp_write_conic_problem (1163)
The problem contains cones that cannot be written to an LP formatted file.

rescode.err_lp_write_geco_problem (1164)
The problem contains general convex terms that cannot be written to an LP formatted file.

rescode.err_writing_file (1166)
An error occurred while writing file

rescode.err_ptf_format (1167)
Syntax error in an PTF file

rescode.err_opf_format (1168)
Syntax error in an OPF file

rescode.err_opf_new_variable (1169)
Introducing new variables is now allowed. When a [variables] section is present, it is not allowed to introduce new variables later in the problem.

rescode.err_invalid_name_in_sol_file (1170)
An invalid name occurred in a solution file.

rescode.err_lp_invalid_con_name (1171)
A constraint name is invalid when used in an LP formatted file.

rescode.err_opf_premature_eof (1172)
Premature end of file in an OPF file.

rescode.err_json_syntax (1175)
Syntax error in an JSON data

rescode.err_json_string (1176)
Error in JSON string.

rescode.err_json_number_overflow (1177)
Invalid number entry - wrong type or value overflow.

rescode.err_json_format (1178)
Error in an JSON Task file

rescode.err_json_data (1179)
Inconsistent data in JSON Task file

rescode.err_json_missing_data (1180)
Missing data section in JSON task file.

rescode.err_argument_lenneq (1197)
Incorrect length of arguments.

rescode.err_argument_type (1198)
Incorrect argument type.

```

rescode.err_num_arguments (1199)
    Incorrect number of function arguments.
rescode.err_in_argument (1200)
    A function argument is incorrect.
rescode.err_argument_dimension (1201)
    A function argument is of incorrect dimension.
rescode.err_shape_is_too_large (1202)
    The size of the n-dimensional shape is too large.
rescode.err_index_is_too_small (1203)
    An index in an argument is too small.
rescode.err_index_is_too_large (1204)
    An index in an argument is too large.
rescode.err_param_name (1205)
    The parameter name is not correct.
rescode.err_param_name_dou (1206)
    The parameter name is not correct for a double parameter.
rescode.err_param_name_int (1207)
    The parameter name is not correct for an integer parameter.
rescode.err_param_name_str (1208)
    The parameter name is not correct for a string parameter.
rescode.err_param_index (1210)
    Parameter index is out of range.
rescode.err_param_is_too_large (1215)
    The parameter value is too large.
rescode.err_param_is_too_small (1216)
    The parameter value is too small.
rescode.err_param_value_str (1217)
    The parameter value string is incorrect.
rescode.err_param_type (1218)
    The parameter type is invalid.
rescode.err_inf_dou_index (1219)
    A double information index is out of range for the specified type.
rescode.err_inf_int_index (1220)
    An integer information index is out of range for the specified type.
rescode.err_index_arr_is_too_small (1221)
    An index in an array argument is too small.
rescode.err_index_arr_is_too_large (1222)
    An index in an array argument is too large.
rescode.err_inf_lint_index (1225)
    A long integer information index is out of range for the specified type.
rescode.err_arg_is_too_small (1226)
    The value of a argument is too small.
rescode.err_arg_is_too_large (1227)
    The value of a argument is too large.
rescode.err_invalid_whichsol (1228)
    whichsol is invalid.
rescode.err_inf_dou_name (1230)
    A double information name is invalid.
rescode.err_inf_int_name (1231)
    An integer information name is invalid.
rescode.err_inf_type (1232)
    The information type is invalid.
rescode.err_inf_lint_name (1234)
    A long integer information name is invalid.
rescode.err_index (1235)
    An index is out of range.
rescode.err_whichsol (1236)
    The solution defined by whichsol does not exists.

```

`rescode.err_solitem (1237)`
The solution item number `solitem` is invalid. Please note that `solitem.snz` is invalid for the basic solution.

`rescode.err_whichitem_not_allowed (1238)`
`whichitem` is unacceptable.

`rescode.err_maxnumcon (1240)`
The maximum number of constraints specified is smaller than the number of constraints in the task.

`rescode.err_maxnumvar (1241)`
The maximum number of variables specified is smaller than the number of variables in the task.

`rescode.err_maxnumbarvar (1242)`
The maximum number of semidefinite variables specified is smaller than the number of semidefinite variables in the task.

`rescode.err_maxnumqnz (1243)`
The maximum number of non-zeros specified for the Q matrices is smaller than the number of non-zeros in the current Q matrices.

`rescode.err_too_small_max_num_nz (1245)`
The maximum number of non-zeros specified is too small.

`rescode.err_invalid_idx (1246)`
A specified index is invalid.

`rescode.err_invalid_max_num (1247)`
A specified index is invalid.

`rescode.err_numconlim (1250)`
Maximum number of constraints limit is exceeded.

`rescode.err_numvarlim (1251)`
Maximum number of variables limit is exceeded.

`rescode.err_too_small_maxnumanz (1252)`
The maximum number of non-zeros specified for A is smaller than the number of non-zeros in the current A .

`rescode.err_inv_aptre (1253)`
`aptrb[j]` is strictly smaller than `aptrb[j]` for some j .

`rescode.err_mul_a_element (1254)`
An element in A is defined multiple times.

`rescode.err_inv_bk (1255)`
Invalid bound key.

`rescode.err_inv_bkc (1256)`
Invalid bound key is specified for a constraint.

`rescode.err_inv_bkx (1257)`
An invalid bound key is specified for a variable.

`rescode.err_inv_var_type (1258)`
An invalid variable type is specified for a variable.

`rescode.err_solver_probtype (1259)`
Problem type does not match the chosen optimizer.

`rescode.err_objective_range (1260)`
Empty objective range.

`rescode.err_undef_solution (1265)`
MOSEK has the following solution types:

- an interior-point solution,
- a basic solution,
- and an integer solution.

Each optimizer may set one or more of these solutions; e.g by default a successful optimization with the interior-point optimizer defines the interior-point solution and, for linear problems, also the basic solution. This error occurs when asking for a solution or for information about a solution that is not defined.

`rescode.err_basis (1266)`
An invalid basis is specified. Either too many or too few basis variables are specified.

`rescode.err_inv_skc (1267)`
 Invalid value in `skc`.
`rescode.err_inv_sxx (1268)`
 Invalid value in `skx`.
`rescode.err_inv_skn (1274)`
 Invalid value in `skn`.
`rescode.err_inv_sk_str (1269)`
 Invalid status key string encountered.
`rescode.err_inv_sk (1270)`
 Invalid status key code.
`rescode.err_inv_cone_type_str (1271)`
 Invalid cone type string encountered.
`rescode.err_inv_cone_type (1272)`
 Invalid cone type code is encountered.
`rescode.err_invalid_surplus (1275)`
 Invalid surplus.
`rescode.err_inv_name_item (1280)`
 An invalid name item code is used.
`rescode.err_pro_item (1281)`
 An invalid problem is used.
`rescode.err_invalid_format_type (1283)`
 Invalid format type.
`rescode.err_firsti (1285)`
 Invalid `firsti`.
`rescode.err_lasti (1286)`
 Invalid `lasti`.
`rescode.err_firstj (1287)`
 Invalid `firstj`.
`rescode.err_lastj (1288)`
 Invalid `lastj`.
`rescode.err_max_len_is_too_small (1289)`
 A maximum length that is too small has been specified.
`rescode.err_nonlinear_equality (1290)`
 The model contains a nonlinear equality which defines a nonconvex set.
`rescode.err_nonconvex (1291)`
 The optimization problem is nonconvex.
`rescode.err_nonlinear_ranged (1292)`
 Nonlinear constraints with finite lower and upper bound always define a nonconvex feasible set.
`rescode.err_con_q_not_psd (1293)`
 The quadratic constraint matrix is not positive semidefinite as expected for a constraint with finite upper bound. This results in a nonconvex problem. The parameter `dparam.check_convexity_rel_tol` can be used to relax the convexity check.
`rescode.err_con_q_not_nsd (1294)`
 The quadratic constraint matrix is not negative semidefinite as expected for a constraint with finite lower bound. This results in a nonconvex problem. The parameter `dparam.check_convexity_rel_tol` can be used to relax the convexity check.
`rescode.err_obj_q_not_psd (1295)`
 The quadratic coefficient matrix in the objective is not positive semidefinite as expected for a minimization problem. The parameter `dparam.check_convexity_rel_tol` can be used to relax the convexity check.
`rescode.err_obj_q_not_nsd (1296)`
 The quadratic coefficient matrix in the objective is not negative semidefinite as expected for a maximization problem. The parameter `dparam.check_convexity_rel_tol` can be used to relax the convexity check.
`rescode.err_argument_perm_array (1299)`
 An invalid permutation array is specified.
`rescode.err_cone_index (1300)`
 An index of a non-existing cone has been specified.

rescode.err_cone_size (1301)
 A cone with incorrect number of members is specified.

rescode.err_cone_overlap (1302)
 One or more of the variables in the cone to be added is already member of another cone. Now assume the variable is x_j then add a new variable say x_k and the constraint

$$x_j = x_k$$

and then let x_k be member of the cone to be appended.

rescode.err_cone_rep_var (1303)
 A variable is included multiple times in the cone.

rescode.err_maxnumcone (1304)
 The value specified for maxnumcone is too small.

rescode.err_cone_type (1305)
 Invalid cone type specified.

rescode.err_cone_type_str (1306)
 Invalid cone type specified.

rescode.err_cone_overlap_append (1307)
 The cone to be appended has one variable which is already member of another cone.

rescode.err_remove_cone_variable (1310)
 A variable cannot be removed because it will make a cone invalid.

rescode.err_appending_too_big_cone (1311)
 Trying to append a too big cone.

rescode.err_cone_parameter (1320)
 An invalid cone parameter.

rescode.err_sol_file_invalid_number (1350)
 An invalid number is specified in a solution file.

rescode.err_huge_c (1375)
 A huge value in absolute size is specified for one c_j .

rescode.err_huge_aij (1380)
 A numerically huge value is specified for an $a_{i,j}$ element in A . The parameter *dparam.data_tol_aij_huge* controls when an $a_{i,j}$ is considered huge.

rescode.err_duplicate_aij (1385)
 An element in the A matrix is specified twice.

rescode.err_lower_bound_is_a_nan (1390)
 The lower bound specified is not a number (nan).

rescode.err_upper_bound_is_a_nan (1391)
 The upper bound specified is not a number (nan).

rescode.err_infinite_bound (1400)
 A numerically huge bound value is specified.

rescode.err_inv_qobj_subi (1401)
 Invalid value in qosubi.

rescode.err_inv_qobj_subj (1402)
 Invalid value in qosubj.

rescode.err_inv_qobj_val (1403)
 Invalid value in qoval.

rescode.err_inv_qcon_subk (1404)
 Invalid value in qcsubk.

rescode.err_inv_qcon_subi (1405)
 Invalid value in qcsubi.

rescode.err_inv_qcon_subj (1406)
 Invalid value in qcsubj.

rescode.err_inv_qcon_val (1407)
 Invalid value in qcval.

rescode.err_qcon_subi_too_small (1408)
 Invalid value in qcsubi.

rescode.err_qcon_subi_too_large (1409)
 Invalid value in qcsubi.

`rescode.err_qobj_upper_triangle` (1415)
 An element in the upper triangle of Q^o is specified. Only elements in the lower triangle should be specified.

`rescode.err_qcon_upper_triangle` (1417)
 An element in the upper triangle of a Q^k is specified. Only elements in the lower triangle should be specified.

`rescode.err_fixed_bound_values` (1420)
 A fixed constraint/variable has been specified using the bound keys but the numerical value of the lower and upper bound is different.

`rescode.err_too_small_a_truncation_value` (1421)
 A too small value for the A truncation value is specified.

`rescode.err_invalid_objective_sense` (1445)
 An invalid objective sense is specified.

`rescode.err_undefined_objective_sense` (1446)
 The objective sense has not been specified before the optimization.

`rescode.err_y_is_undefined` (1449)
 The solution item y is undefined.

`rescode.err_nan_in_double_data` (1450)
 An invalid floating point value was used in some double data.

`rescode.err_nan_in_blc` (1461)
 l^c contains an invalid floating point value, i.e. a NaN.

`rescode.err_nan_in_buc` (1462)
 u^c contains an invalid floating point value, i.e. a NaN.

`rescode.err_nan_in_c` (1470)
 c contains an invalid floating point value, i.e. a NaN.

`rescode.err_nan_in_blx` (1471)
 l^x contains an invalid floating point value, i.e. a NaN.

`rescode.err_nan_in_bux` (1472)
 u^x contains an invalid floating point value, i.e. a NaN.

`rescode.err_invalid_aij` (1473)
 $a_{i,j}$ contains an invalid floating point value, i.e. a NaN or an infinite value.

`rescode.err_invalid_cj` (1474)
 c_j contains an invalid floating point value, i.e. a NaN or an infinite value.

`rescode.err_sym_mat_invalid` (1480)
 A symmetric matrix contains an invalid floating point value, i.e. a NaN or an infinite value.

`rescode.err_sym_mat_huge` (1482)
 A symmetric matrix contains a huge value in absolute size. The parameter `dparam.data_sym_mat_tol_huge` controls when an $e_{i,j}$ is considered huge.

`rescode.err_inv_problem` (1500)
 Invalid problem type. Probably a nonconvex problem has been specified.

`rescode.err_mixed_conic_and_nl` (1501)
 The problem contains nonlinear terms conic constraints. The requested operation cannot be applied to this type of problem.

`rescode.err_global_inv_conic_problem` (1503)
 The global optimizer can only be applied to problems without semidefinite variables.

`rescode.err_inv_optimizer` (1550)
 An invalid optimizer has been chosen for the problem.

`rescode.err_mio_no_optimizer` (1551)
 No optimizer is available for the current class of integer optimization problems.

`rescode.err_no_optimizer_var_type` (1552)
 No optimizer is available for this class of optimization problems.

`rescode.err_final_solution` (1560)
 An error occurred during the solution finalization.

`rescode.err_first` (1570)
 Invalid first.

`rescode.err_last` (1571)
 Invalid index last. A given index was out of expected range.

`rescode.err_slice_size (1572)`
 Invalid slice size specified.

`rescode.err_negative_surplus (1573)`
 Negative surplus.

`rescode.err_negative_append (1578)`
 Cannot append a negative number.

`rescode.err_postsolve (1580)`
 An error occurred during the postsolve. Please contact **MOSEK** support.

`rescode.err_overflow (1590)`
 A computation produced an overflow i.e. a very large number.

`rescode.err_no_basis_sol (1600)`
 No basic solution is defined.

`rescode.err_basis_factor (1610)`
 The factorization of the basis is invalid.

`rescode.err_basis_singular (1615)`
 The basis is singular and hence cannot be factored.

`rescode.err_factor (1650)`
 An error occurred while factorizing a matrix.

`rescode.err_feasrepair_cannot_relax (1700)`
 An optimization problem cannot be relaxed.

`rescode.err_feasrepair_solving_relaxed (1701)`
 The relaxed problem could not be solved to optimality. Please consult the log file for further details.

`rescode.err_feasrepair_inconsistent_bound (1702)`
 The upper bound is less than the lower bound for a variable or a constraint. Please correct this before running the feasibility repair.

`rescode.err_repair_invalid_problem (1710)`
 The feasibility repair does not support the specified problem type.

`rescode.err_repair_optimization_failed (1711)`
 Computation the optimal relaxation failed. The cause may have been numerical problems.

`rescode.err_name_max_len (1750)`
 A name is longer than the buffer that is supposed to hold it.

`rescode.err_name_is_null (1760)`
 The name buffer is a NULL pointer.

`rescode.err_invalid_compression (1800)`
 Invalid compression type.

`rescode.err_invalid_iomode (1801)`
 Invalid io mode.

`rescode.err_no_primal_infeas_cer (2000)`
 A certificate of primal infeasibility is not available.

`rescode.err_no_dual_infeas_cer (2001)`
 A certificate of infeasibility is not available.

`rescode.err_no_solution_in_callback (2500)`
 The required solution is not available.

`rescode.err_inv_marki (2501)`
 Invalid value in marki.

`rescode.err_inv_markj (2502)`
 Invalid value in markj.

`rescode.err_inv_numi (2503)`
 Invalid numi.

`rescode.err_inv_numj (2504)`
 Invalid numj.

`rescode.err_task_incompatible (2560)`
 The Task file is incompatible with this platform. This results from reading a file on a 32 bit platform generated on a 64 bit platform.

`rescode.err_task_invalid (2561)`
 The Task file is invalid.

`rescode.err_task_write (2562)`
 Failed to write the task file.

`rescode.err_lu_max_num_tries (2800)`
 Could not compute the LU factors of the matrix within the maximum number of allowed tries.

`rescode.err_invalid_utf8 (2900)`
 An invalid UTF8 string is encountered.

`rescode.err_invalid_wchar (2901)`
 An invalid wchar string is encountered.

`rescode.err_no_dual_for_itg_sol (2950)`
 No dual information is available for the integer solution.

`rescode.err_no_snx_for_bas_sol (2953)`
 s_n^x is not available for the basis solution.

`rescode.err_internal (3000)`
 An internal error occurred. Please report this problem.

`rescode.err_api_array_too_small (3001)`
 An input array was too short.

`rescode.err_api_cb_connect (3002)`
 Failed to connect a callback object.

`rescode.err_api_fatal_error (3005)`
 An internal error occurred in the API. Please report this problem.

`rescode.err_api_internal (3999)`
 An internal fatal error occurred in an interface function.

`rescode.err_sen_format (3050)`
 Syntax error in sensitivity analysis file.

`rescode.err_sen_undef_name (3051)`
 An undefined name was encountered in the sensitivity analysis file.

`rescode.err_sen_index_range (3052)`
 Index out of range in the sensitivity analysis file.

`rescode.err_sen_bound_invalid_up (3053)`
 Analysis of upper bound requested for an index, where no upper bound exists.

`rescode.err_sen_bound_invalid_lo (3054)`
 Analysis of lower bound requested for an index, where no lower bound exists.

`rescode.err_sen_index_invalid (3055)`
 Invalid range given in the sensitivity file.

`rescode.err_sen_invalid_regexp (3056)`
 Syntax error in regexp or regexp longer than 1024.

`rescode.err_sen_solution_status (3057)`
 No optimal solution found to the original problem given for sensitivity analysis.

`rescode.err_sen_numerical (3058)`
 Numerical difficulties encountered performing the sensitivity analysis.

`rescode.err_sen_unhandled_problem_type (3080)`
 Sensitivity analysis cannot be performed for the specified problem. Sensitivity analysis is only possible for linear problems.

`rescode.err_unb_step_size (3100)`
 A step size in an optimizer was unexpectedly unbounded. For instance, if the step-size becomes unbounded in phase 1 of the simplex algorithm then an error occurs. Normally this will happen only if the problem is badly formulated. Please contact **MOSEK** support if this error occurs.

`rescode.err_identical_tasks (3101)`
 Some tasks related to this function call were identical. Unique tasks were expected.

`rescode.err_ad_invalid_codelist (3102)`
 The code list data was invalid.

`rescode.err_internal_test_failed (3500)`
 An internal unit test function failed.

`rescode.err_xml_invalid_problem_type (3600)`
 The problem type is not supported by the XML format.

`rescode.err_invalid_ampl_stub (3700)`
 Invalid AMPL stub.

`rescode.err_int64_to_int32_cast (3800)`
 A 64 bit integer could not be cast to a 32 bit integer.

rescode.err_size_license_numcores (3900)
The computer contains more cpu cores than the license allows for.

rescode.err_infeas_undefined (3910)
The requested value is not defined for this solution type.

rescode.err_no_barx_for_solution (3915)
There is no \bar{X} available for the solution specified. In particular note there are no \bar{X} defined for the basic and integer solutions.

rescode.err_no_bars_for_solution (3916)
There is no \bar{s} available for the solution specified. In particular note there are no \bar{s} defined for the basic and integer solutions.

rescode.err_bar_var_dim (3920)
The dimension of a symmetric matrix variable has to be greater than 0.

rescode.err_sym_mat_invalid_row_index (3940)
A row index specified for sparse symmetric matrix is invalid.

rescode.err_sym_mat_invalid_col_index (3941)
A column index specified for sparse symmetric matrix is invalid.

rescode.err_sym_mat_not_lower_tringular (3942)
Only the lower triangular part of sparse symmetric matrix should be specified.

rescode.err_sym_mat_invalid_value (3943)
The numerical value specified in a sparse symmetric matrix is not a floating point value.

rescode.err_sym_mat_duplicate (3944)
A value in a symmetric matrix as been specified more than once.

rescode.err_invalid_sym_mat_dim (3950)
A sparse symmetric matrix of invalid dimension is specified.

rescode.err_invalid_file_format_for_sym_mat (4000)
The file format does not support a problem with symmetric matrix variables.

rescode.err_invalid_file_format_for_cfix (4001)
The file format does not support a problem with nonzero fixed term in c.

rescode.err_invalid_file_format_for_ranged_constraints (4002)
The file format does not support a problem with ranged constraints.

rescode.err_invalid_file_format_for_free_constraints (4003)
The file format does not support a problem with free constraints.

rescode.err_invalid_file_format_for_cones (4005)
The file format does not support a problem with conic constraints.

rescode.err_invalid_file_format_for_nonlinear (4010)
The file format does not support a problem with nonlinear terms.

rescode.err_duplicate_constraint_names (4500)
Two constraint names are identical.

rescode.err_duplicate_variable_names (4501)
Two variable names are identical.

rescode.err_duplicate_barvariable_names (4502)
Two barvariable names are identical.

rescode.err_duplicate_cone_names (4503)
Two cone names are identical.

rescode.err_non_unique_array (5000)
An array does not contain unique elements.

rescode.err_argument_is_too_large (5005)
The value of a function argument is too large.

rescode.err_mio_internal (5010)
A fatal error occurred in the mixed integer optimizer. Please contact **MOSEK** support.

rescode.err_invalid_problem_type (6000)
An invalid problem type.

rescode.err_unhandled_solution_status (6010)
Unhandled solution status.

rescode.err_upper_triangle (6020)
An element in the upper triangle of a lower triangular matrix is specified.

rescode.err_lau_singular_matrix (7000)
A matrix is singular.

```

rescode.err_lau_not_positive_definite (7001)
    A matrix is not positive definite.
rescode.err_lau_invalid_lower_triangular_matrix (7002)
    An invalid lower triangular matrix.
rescode.err_lau_unknown (7005)
    An unknown error.
rescode.err_lau_arg_m (7010)
    Invalid argument m.
rescode.err_lau_arg_n (7011)
    Invalid argument n.
rescode.err_lau_arg_k (7012)
    Invalid argument k.
rescode.err_lau_arg_transa (7015)
    Invalid argument transa.
rescode.err_lau_arg_transb (7016)
    Invalid argument transb.
rescode.err_lau_arg_uplo (7017)
    Invalid argument uplo.
rescode.err_lau_arg_trans (7018)
    Invalid argument trans.
rescode.err_lau_invalid_sparse_symmetric_matrix (7019)
    An invalid sparse symmetric matrix is specified. Note only the lower triangular part with no
    duplicates is specified.
rescode.err_cbf_parse (7100)
    An error occurred while parsing an CBF file.
rescode.err_cbf_obj_sense (7101)
    An invalid objective sense is specified.
rescode.err_cbf_no_variables (7102)
    No variables are specified.
rescode.err_cbf_too_many_constraints (7103)
    Too many constraints specified.
rescode.err_cbf_too_many_variables (7104)
    Too many variables specified.
rescode.err_cbf_no_version_specified (7105)
    No version specified.
rescode.err_cbf_syntax (7106)
    Invalid syntax.
rescode.err_cbf_duplicate_obj (7107)
    Duplicate OBJ keyword.
rescode.err_cbf_duplicate_con (7108)
    Duplicate CON keyword.
rescode.err_cbf_duplicate_var (7109)
    Duplicate VAR keyword.
rescode.err_cbf_duplicate_int (7110)
    Duplicate INT keyword.
rescode.err_cbf_invalid_var_type (7111)
    Invalid variable type.
rescode.err_cbf_invalid_con_type (7112)
    Invalid constraint type.
rescode.err_cbf_invalid_domain_dimension (7113)
    Invalid domain dimension.
rescode.err_cbf_duplicate_objcoord (7114)
    Duplicate index in OBJCOORD.
rescode.err_cbf_duplicate_bcoord (7115)
    Duplicate index in BCOORD.
rescode.err_cbf_duplicate_acoord (7116)
    Duplicate index in ACOORD.

```

```

rescode.err_cbf_too_few_variables (7117)
    Too few variables defined.
rescode.err_cbf_too_few_constraints (7118)
    Too few constraints defined.
rescode.err_cbf_too_few_ints (7119)
    Too few ints are specified.
rescode.err_cbf_too_many_ints (7120)
    Too many ints are specified.
rescode.err_cbf_invalid_int_index (7121)
    Invalid INT index.
rescode.err_cbf_unsupported (7122)
    Unsupported feature is present.
rescode.err_cbf_duplicate_psdvar (7123)
    Duplicate PSDVAR keyword.
rescode.err_cbf_invalid_psdvar_dimension (7124)
    Invalid PSDVAR dimension.
rescode.err_cbf_too_few_psdvar (7125)
    Too few variables defined.
rescode.err_cbf_invalid_exp_dimension (7126)
    Invalid dimension of a exponential cone.
rescode.err_cbf_duplicate_pow_cones (7130)
    Multiple POWCONES specified.
rescode.err_cbf_duplicate_pow_star_cones (7131)
    Multiple POW*CONES specified.
rescode.err_cbf_invalid_power (7132)
    Invalid power specified.
rescode.err_cbf_power_cone_is_too_long (7133)
    Power cone is too long.
rescode.err_cbf_invalid_power_cone_index (7134)
    Invalid power cone index.
rescode.err_cbf_invalid_power_star_cone_index (7135)
    Invalid power star cone index.
rescode.err_cbf_unhandled_power_cone_type (7136)
    An unhandled power cone type.
rescode.err_cbf_unhandled_power_star_cone_type (7137)
    An unhandled power star cone type.
rescode.err_cbf_power_cone_mismatch (7138)
    The power cone does not match with it definition.
rescode.err_cbf_power_star_cone_mismatch (7139)
    The power star cone does not match with it definition.
rescode.err_cbf_invalid_number_of_cones (7740)
    Invalid number of cones.
rescode.err_cbf_invalid_dimension_of_cones (7741)
    Invalid dimension of cones.
rescode.err_mio_invalid_root_optimizer (7700)
    An invalid root optimizer was selected for the problem type.
rescode.err_mio_invalid_node_optimizer (7701)
    An invalid node optimizer was selected for the problem type.
rescode.err_toconic_constr_q_not_psd (7800)
    The matrix defining the quadratic part of constraint is not positive semidefinite.
rescode.err_toconic_constraint_fx (7801)
    The quadratic constraint is an equality, thus not convex.
rescode.err_toconic_constraint_ra (7802)
    The quadratic constraint has finite lower and upper bound, and therefore it is not convex.
rescode.err_toconic_constr_not_conic (7803)
    The constraint is not conic representable.
rescode.err_toconic_objective_not_psd (7804)
    The matrix defining the quadratic part of the objective function is not positive semidefinite.

```

rescode.err_server_connect (8000)
 Failed to connect to remote solver server. The server string or the port string were invalid, or the server did not accept connection.

rescode.err_server_protocol (8001)
 Unexpected message or data from solver server.

rescode.err_server_status (8002)
 Server returned non-ok HTTP status code

rescode.err_server_token (8003)
 The job ID specified is incorrect or invalid

rescode.err_server_problem_size (8008)
 The size of the problem exceeds the dimensions permitted by the instance of the OptServer where it was run.

15.9 Enumerations

basindtype
 Basis identification

basindtype.never
 Never do basis identification.

basindtype.always
 Basis identification is always performed even if the interior-point optimizer terminates abnormally.

basindtype.no_error
 Basis identification is performed if the interior-point optimizer terminates without an error.

basindtype.if_feasible
 Basis identification is not performed if the interior-point optimizer terminates with a problem status saying that the problem is primal or dual infeasible.

basindtype.reservered
 Not currently in use.

boundkey
 Bound keys

boundkey.lo
 The constraint or variable has a finite lower bound and an infinite upper bound.

boundkey.up
 The constraint or variable has an infinite lower bound and an finite upper bound.

boundkey.fx
 The constraint or variable is fixed.

boundkey.fr
 The constraint or variable is free.

boundkey.ra
 The constraint or variable is ranged.

mark
 Mark

mark.lo
 The lower bound is selected for sensitivity analysis.

mark.up
 The upper bound is selected for sensitivity analysis.

simdegen
 Degeneracy strategies

simdegen.none
 The simplex optimizer should use no degeneration strategy.

simdegen.free
 The simplex optimizer chooses the degeneration strategy.

`simdegen.aggressive`
The simplex optimizer should use an aggressive degeneration strategy.

`simdegen.moderate`
The simplex optimizer should use a moderate degeneration strategy.

`simdegen.minimum`
The simplex optimizer should use a minimum degeneration strategy.

`transpose`
Transposed matrix.

`transpose.no`
No transpose is applied.

`transpose.yes`
A transpose is applied.

`uplo`
Triangular part of a symmetric matrix.

`uplo.lo`
Lower part.

`uplo.up`
Upper part.

`simreform`
Problem reformulation.

`simreform.on`
Allow the simplex optimizer to reformulate the problem.

`simreform.off`
Disallow the simplex optimizer to reformulate the problem.

`simreform.free`
The simplex optimizer can choose freely.

`simreform.aggressive`
The simplex optimizer should use an aggressive reformulation strategy.

`simdupvec`
Exploit duplicate columns.

`simdupvec.on`
Allow the simplex optimizer to exploit duplicated columns.

`simdupvec.off`
Disallow the simplex optimizer to exploit duplicated columns.

`simdupvec.free`
The simplex optimizer can choose freely.

`simhotstart`
Hot-start type employed by the simplex optimizer

`simhotstart.none`
The simplex optimizer performs a coldstart.

`simhotstart.free`
The simplex optimizer chooses the hot-start type.

`simhotstart.status_keys`
Only the status keys of the constraints and variables are used to choose the type of hot-start.

`intpnthotstart`
Hot-start type employed by the interior-point optimizers.

`intpnthotstart.none`
The interior-point optimizer performs a coldstart.

`intpnthotstart.primal`
The interior-point optimizer exploits the primal solution only.

`intpnthotstart.dual`
The interior-point optimizer exploits the dual solution only.

`intpnthotstart.primal_dual`
The interior-point optimizer exploits both the primal and dual solution.

`purify`
Solution purification employed optimizer.

`purify.none`
The optimizer performs no solution purification.

`purify.primal`
The optimizer purifies the primal solution.

`purify.dual`
The optimizer purifies the dual solution.

`purify.primal_dual`
The optimizer purifies both the primal and dual solution.

`purify.auto`
TBD

`callbackcode`
Progress callback codes

`callbackcode.begin_bi`
The basis identification procedure has been started.

`callbackcode.begin_conic`
The callback function is called when the conic optimizer is started.

`callbackcode.begin_dual_bi`
The callback function is called from within the basis identification procedure when the dual phase is started.

`callbackcode.begin_dual_sensitivity`
Dual sensitivity analysis is started.

`callbackcode.begin_dual_setup_bi`
The callback function is called when the dual BI phase is started.

`callbackcode.begin_dual_simplex`
The callback function is called when the dual simplex optimizer started.

`callbackcode.begin_dual_simplex_bi`
The callback function is called from within the basis identification procedure when the dual simplex clean-up phase is started.

`callbackcode.begin_full_convexity_check`
Begin full convexity check.

`callbackcode.begin_infeas_ana`
The callback function is called when the infeasibility analyzer is started.

`callbackcode.begin_intpnt`
The callback function is called when the interior-point optimizer is started.

`callbackcode.begin_license_wait`
Begin waiting for license.

`callbackcode.begin_mio`
The callback function is called when the mixed-integer optimizer is started.

`callbackcode.begin_optimizer`
The callback function is called when the optimizer is started.

`callbackcode.begin_presolve`
The callback function is called when the presolve is started.

`callbackcode.begin_primal_bi`
The callback function is called from within the basis identification procedure when the primal phase is started.

`callbackcode.begin_primal_repair`
Begin primal feasibility repair.

`callbackcode.begin_primal_sensitivity`
 Primal sensitivity analysis is started.

`callbackcode.begin_primal_setup_bi`
 The callback function is called when the primal BI setup is started.

`callbackcode.begin_primal_simplex`
 The callback function is called when the primal simplex optimizer is started.

`callbackcode.begin_primal_simplex_bi`
 The callback function is called from within the basis identification procedure when the primal simplex clean-up phase is started.

`callbackcode.begin_qcqp_reformulate`
 Begin QCQP reformulation.

`callbackcode.begin_read`
MOSEK has started reading a problem file.

`callbackcode.begin_root_cutgen`
 The callback function is called when root cut generation is started.

`callbackcode.begin_simplex`
 The callback function is called when the simplex optimizer is started.

`callbackcode.begin_simplex_bi`
 The callback function is called from within the basis identification procedure when the simplex clean-up phase is started.

`callbackcode.begin_to_conic`
 Begin conic reformulation.

`callbackcode.begin_write`
MOSEK has started writing a problem file.

`callbackcode.conic`
 The callback function is called from within the conic optimizer after the information database has been updated.

`callbackcode.dual_simplex`
 The callback function is called from within the dual simplex optimizer.

`callbackcode.end_bi`
 The callback function is called when the basis identification procedure is terminated.

`callbackcode.end_conic`
 The callback function is called when the conic optimizer is terminated.

`callbackcode.end_dual_bi`
 The callback function is called from within the basis identification procedure when the dual phase is terminated.

`callbackcode.end_dual_sensitivity`
 Dual sensitivity analysis is terminated.

`callbackcode.end_dual_setup_bi`
 The callback function is called when the dual BI phase is terminated.

`callbackcode.end_dual_simplex`
 The callback function is called when the dual simplex optimizer is terminated.

`callbackcode.end_dual_simplex_bi`
 The callback function is called from within the basis identification procedure when the dual clean-up phase is terminated.

`callbackcode.end_full_convexity_check`
 End full convexity check.

`callbackcode.end_infeas_ana`
 The callback function is called when the infeasibility analyzer is terminated.

`callbackcode.end_intpnt`
 The callback function is called when the interior-point optimizer is terminated.

`callbackcode.end_license_wait`
End waiting for license.

`callbackcode.end_mio`
The callback function is called when the mixed-integer optimizer is terminated.

`callbackcode.end_optimizer`
The callback function is called when the optimizer is terminated.

`callbackcode.end_presolve`
The callback function is called when the presolve is completed.

`callbackcode.end_primal_bi`
The callback function is called from within the basis identification procedure when the primal phase is terminated.

`callbackcode.end_primal_repair`
End primal feasibility repair.

`callbackcode.end_primal_sensitivity`
Primal sensitivity analysis is terminated.

`callbackcode.end_primal_setup_bi`
The callback function is called when the primal BI setup is terminated.

`callbackcode.end_primal_simplex`
The callback function is called when the primal simplex optimizer is terminated.

`callbackcode.end_primal_simplex_bi`
The callback function is called from within the basis identification procedure when the primal clean-up phase is terminated.

`callbackcode.end_qcqp_reformulate`
End QCQP reformulation.

`callbackcode.end_read`
MOSEK has finished reading a problem file.

`callbackcode.end_root_cutgen`
The callback function is called when root cut generation is terminated.

`callbackcode.end_simplex`
The callback function is called when the simplex optimizer is terminated.

`callbackcode.end_simplex_bi`
The callback function is called from within the basis identification procedure when the simplex clean-up phase is terminated.

`callbackcode.end_to_conic`
End conic reformulation.

`callbackcode.end_write`
MOSEK has finished writing a problem file.

`callbackcode.im_bi`
The callback function is called from within the basis identification procedure at an intermediate point.

`callbackcode.im_conic`
The callback function is called at an intermediate stage within the conic optimizer where the information database has not been updated.

`callbackcode.im_dual_bi`
The callback function is called from within the basis identification procedure at an intermediate point in the dual phase.

`callbackcode.im_dual_sensitivity`
The callback function is called at an intermediate stage of the dual sensitivity analysis.

`callbackcode.im_dual_simplex`
The callback function is called at an intermediate point in the dual simplex optimizer.

`callbackcode.im_full_convexity_check`

The callback function is called at an intermediate stage of the full convexity check.

`callbackcode.im_intpnt`

The callback function is called at an intermediate stage within the interior-point optimizer where the information database has not been updated.

`callbackcode.im_license_wait`

MOSEK is waiting for a license.

`callbackcode.im_lu`

The callback function is called from within the LU factorization procedure at an intermediate point.

`callbackcode.im_mio`

The callback function is called at an intermediate point in the mixed-integer optimizer.

`callbackcode.im_mio_dual_simplex`

The callback function is called at an intermediate point in the mixed-integer optimizer while running the dual simplex optimizer.

`callbackcode.im_mio_intpnt`

The callback function is called at an intermediate point in the mixed-integer optimizer while running the interior-point optimizer.

`callbackcode.im_mio_primal_simplex`

The callback function is called at an intermediate point in the mixed-integer optimizer while running the primal simplex optimizer.

`callbackcode.im_order`

The callback function is called from within the matrix ordering procedure at an intermediate point.

`callbackcode.im_presolve`

The callback function is called from within the presolve procedure at an intermediate stage.

`callbackcode.im_primal_bi`

The callback function is called from within the basis identification procedure at an intermediate point in the primal phase.

`callbackcode.im_primal_sensitivity`

The callback function is called at an intermediate stage of the primal sensitivity analysis.

`callbackcode.im_primal_simplex`

The callback function is called at an intermediate point in the primal simplex optimizer.

`callbackcode.im_qo_reformulate`

The callback function is called at an intermediate stage of the conic quadratic reformulation.

`callbackcode.im_read`

Intermediate stage in reading.

`callbackcode.im_root_cutgen`

The callback is called from within root cut generation at an intermediate stage.

`callbackcode.im_simplex`

The callback function is called from within the simplex optimizer at an intermediate point.

`callbackcode.im_simplex_bi`

The callback function is called from within the basis identification procedure at an intermediate point in the simplex clean-up phase. The frequency of the callbacks is controlled by the *iparam.log_sim_freq* parameter.

`callbackcode.intpnt`

The callback function is called from within the interior-point optimizer after the information database has been updated.

`callbackcode.new_int_mio`

The callback function is called after a new integer solution has been located by the mixed-integer optimizer.

`callbackcode.primal_simplex`
The callback function is called from within the primal simplex optimizer.

`callbackcode.read_opf`
The callback function is called from the OPF reader.

`callbackcode.read_opf_section`
A chunk of Q non-zeros has been read from a problem file.

`callbackcode.solving_remote`
The callback function is called while the task is being solved on a remote server.

`callbackcode.update_dual_bi`
The callback function is called from within the basis identification procedure at an intermediate point in the dual phase.

`callbackcode.update_dual_simplex`
The callback function is called in the dual simplex optimizer.

`callbackcode.update_dual_simplex_bi`
The callback function is called from within the basis identification procedure at an intermediate point in the dual simplex clean-up phase. The frequency of the callbacks is controlled by the *iparam.log_sim_freq* parameter.

`callbackcode.update_presolve`
The callback function is called from within the presolve procedure.

`callbackcode.update_primal_bi`
The callback function is called from within the basis identification procedure at an intermediate point in the primal phase.

`callbackcode.update_primal_simplex`
The callback function is called in the primal simplex optimizer.

`callbackcode.update_primal_simplex_bi`
The callback function is called from within the basis identification procedure at an intermediate point in the primal simplex clean-up phase. The frequency of the callbacks is controlled by the *iparam.log_sim_freq* parameter.

`callbackcode.write_opf`
The callback function is called from the OPF writer.

`checkconvexitytype`
Types of convexity checks.

`checkconvexitytype.none`
No convexity check.

`checkconvexitytype.simple`
Perform simple and fast convexity check.

`checkconvexitytype.full`
Perform a full convexity check.

`compresstype`
Compression types

`compresstype.none`
No compression is used.

`compresstype.free`
The type of compression used is chosen automatically.

`compresstype.gzip`
The type of compression used is gzip compatible.

`compresstype.zstd`
The type of compression used is zstd compatible.

`conetype`
Cone types

`conetype.quad`
The cone is a quadratic cone.

`conetype.rquad`
The cone is a rotated quadratic cone.

`conetype.pexp`
A primal exponential cone.

`conetype.dexp`
A dual exponential cone.

`conetype.ppow`
A primal power cone.

`conetype.dpow`
A dual power cone.

`conetype.zero`
The zero cone.

`nametype`
Name types

`nametype.gen`
General names. However, no duplicate and blank names are allowed.

`nametype.mps`
MPS type names.

`nametype.lp`
LP type names.

`scopr`
SCopt operator types

`scopr.ent`
Entropy

`scopr.exp`
Exponential

`scopr.log`
Logarithm

`scopr.pow`
Power

`scopr.sqrt`
Square root

`symmattype`
Cone types

`symmattype.sparse`
Sparse symmetric matrix.

`dataformat`
Data format types

`dataformat.extension`
The file extension is used to determine the data file format.

`dataformat.mps`
The data file is MPS formatted.

`dataformat.lp`
The data file is LP formatted.

`dataformat.op`
The data file is an optimization problem formatted file.

`dataformat.free_mps`
The data a free MPS formatted file.

`dataformat.task`
Generic task dump file.

`dataformat.ptf`
(P)retty (T)ext (F)format.

`dataformat.cb`
 Conic benchmark format,

`dataformat.json_task`
 JSON based task format.

`dinfitem`
 Double information items

`dinfitem.bi_clean_dual_time`
 Time spent within the dual clean-up optimizer of the basis identification procedure since its invocation.

`dinfitem.bi_clean_primal_time`
 Time spent within the primal clean-up optimizer of the basis identification procedure since its invocation.

`dinfitem.bi_clean_time`
 Time spent within the clean-up phase of the basis identification procedure since its invocation.

`dinfitem.bi_dual_time`
 Time spent within the dual phase basis identification procedure since its invocation.

`dinfitem.bi_primal_time`
 Time spent within the primal phase of the basis identification procedure since its invocation.

`dinfitem.bi_time`
 Time spent within the basis identification procedure since its invocation.

`dinfitem.intpnt_dual_feas`
 Dual feasibility measure reported by the interior-point optimizer. (For the interior-point optimizer this measure is not directly related to the original problem because a homogeneous model is employed.)

`dinfitem.intpnt_dual_obj`
 Dual objective value reported by the interior-point optimizer.

`dinfitem.intpnt_factor_num_flops`
 An estimate of the number of flops used in the factorization.

`dinfitem.intpnt_opt_status`
 A measure of optimality of the solution. It should converge to +1 if the problem has a primal-dual optimal solution, and converge to -1 if the problem is (strictly) primal or dual infeasible. If the measure converges to another constant, or fails to settle, the problem is usually ill-posed.

`dinfitem.intpnt_order_time`
 Order time (in seconds).

`dinfitem.intpnt_primal_feas`
 Primal feasibility measure reported by the interior-point optimizer. (For the interior-point optimizer this measure is not directly related to the original problem because a homogeneous model is employed).

`dinfitem.intpnt_primal_obj`
 Primal objective value reported by the interior-point optimizer.

`dinfitem.intpnt_time`
 Time spent within the interior-point optimizer since its invocation.

`dinfitem.mio_clique_separation_time`
 Separation time for clique cuts.

`dinfitem.mio_cmir_separation_time`
 Separation time for CMIR cuts.

`dinfitem.mio_construct_solution_obj`
 If **MOSEK** has successfully constructed an integer feasible solution, then this item contains the optimal objective value corresponding to the feasible solution.

`dinfitem.mio_dual_bound_after_presolve`
 Value of the dual bound after presolve but before cut generation.

`dinfitem.mio_gmi_separation_time`

Separation time for GMI cuts.

`dinfitem.mio_implied_bound_time`

Separation time for implied bound cuts.

`dinfitem.mio_knapsack_cover_separation_time`

Separation time for knapsack cover.

`dinfitem.mio_obj_abs_gap`

Given the mixed-integer optimizer has computed a feasible solution and a bound on the optimal objective value, then this item contains the absolute gap defined by

$$|(\text{objective value of feasible solution}) - (\text{objective bound})|.$$

Otherwise it has the value -1.0.

`dinfitem.mio_obj_bound`

The best known bound on the objective function. This value is undefined until at least one relaxation has been solved: To see if this is the case check that `dinfitem.mio_num_relax` is strictly positive.

`dinfitem.mio_obj_int`

The primal objective value corresponding to the best integer feasible solution. Please note that at least one integer feasible solution must have been located i.e. check `dinfitem.mio_num_int_solutions`.

`dinfitem.mio_obj_rel_gap`

Given that the mixed-integer optimizer has computed a feasible solution and a bound on the optimal objective value, then this item contains the relative gap defined by

$$\frac{|(\text{objective value of feasible solution}) - (\text{objective bound})|}{\max(\delta, |(\text{objective value of feasible solution})|)}.$$

where δ is given by the parameter `dparam.mio_rel_gap_const`. Otherwise it has the value -1.0.

`dinfitem.mio_probing_time`

Total time for probing.

`dinfitem.mio_root_cutgen_time`

Total time for cut generation.

`dinfitem.mio_root_optimizer_time`

Time spent in the optimizer while solving the root node relaxation

`dinfitem.mio_root_presolve_time`

Time spent presolving the problem at the root node.

`dinfitem.mio_time`

Time spent in the mixed-integer optimizer.

`dinfitem.mio_user_obj_cut`

If the objective cut is used, then this information item has the value of the cut.

`dinfitem.optimizer_time`

Total time spent in the optimizer since it was invoked.

`dinfitem.presolve_eli_time`

Total time spent in the eliminator since the presolve was invoked.

`dinfitem.presolve_lindp_time`

Total time spent in the linear dependency checker since the presolve was invoked.

`dinfitem.presolve_time`

Total time (in seconds) spent in the presolve since it was invoked.

`dinfitem.primal_repair_penalty_obj`

The optimal objective value of the penalty function.

`dinfitem.qcqp_reformulate_max_perturbation`

Maximum absolute diagonal perturbation occurring during the QCQP reformulation.

`dinfitem.qcqp_reformulate_time`
Time spent with conic quadratic reformulation.

`dinfitem.qcqp_reformulate_worst_cholesky_column_scaling`
Worst Cholesky column scaling.

`dinfitem.qcqp_reformulate_worst_cholesky_diag_scaling`
Worst Cholesky diagonal scaling.

`dinfitem.rd_time`
Time spent reading the data file.

`dinfitem.sim_dual_time`
Time spent in the dual simplex optimizer since invoking it.

`dinfitem.sim_feas`
Feasibility measure reported by the simplex optimizer.

`dinfitem.sim_obj`
Objective value reported by the simplex optimizer.

`dinfitem.sim_primal_time`
Time spent in the primal simplex optimizer since invoking it.

`dinfitem.sim_time`
Time spent in the simplex optimizer since invoking it.

`dinfitem.sol_bas_dual_obj`
Dual objective value of the basic solution. Updated if *iparam.auto_update_sol_info* is set or by the method *Task.update_solutioninfo*.

`dinfitem.sol_bas_dviolcon`
Maximal dual bound violation for x^c in the basic solution. Updated if *iparam.auto_update_sol_info* is set or by the method *Task.update_solutioninfo*.

`dinfitem.sol_bas_dviolvar`
Maximal dual bound violation for x^x in the basic solution. Updated if *iparam.auto_update_sol_info* is set or by the method *Task.update_solutioninfo*.

`dinfitem.sol_bas_nrm_barx`
Infinity norm of \bar{X} in the basic solution.

`dinfitem.sol_bas_nrm_slc`
Infinity norm of s_l^c in the basic solution.

`dinfitem.sol_bas_nrm_slx`
Infinity norm of s_l^x in the basic solution.

`dinfitem.sol_bas_nrm_suc`
Infinity norm of s_u^c in the basic solution.

`dinfitem.sol_bas_nrm_sux`
Infinity norm of s_u^X in the basic solution.

`dinfitem.sol_bas_nrm_xc`
Infinity norm of x^c in the basic solution.

`dinfitem.sol_bas_nrm_xx`
Infinity norm of x^x in the basic solution.

`dinfitem.sol_bas_nrm_y`
Infinity norm of y in the basic solution.

`dinfitem.sol_bas_primal_obj`
Primal objective value of the basic solution. Updated if *iparam.auto_update_sol_info* is set or by the method *Task.update_solutioninfo*.

`dinfitem.sol_bas_pviolcon`
Maximal primal bound violation for x^c in the basic solution. Updated if *iparam.auto_update_sol_info* is set or by the method *Task.update_solutioninfo*.

`dinfitem.sol_bas_pviolvar`
Maximal primal bound violation for x^x in the basic solution. Updated if *iparam.auto_update_sol_info* is set or by the method *Task.update_solutioninfo*.

`dinfitem.sol_itg_nrm_barx`
Infinity norm of \bar{X} in the integer solution.

`dinfitem.sol_itg_nrm_xc`
Infinity norm of x^c in the integer solution.

`dinfitem.sol_itg_nrm_xx`
Infinity norm of x^x in the integer solution.

`dinfitem.sol_itg_primal_obj`
Primal objective value of the integer solution. Updated if `iparam.auto_update_sol_info` is set or by the method `Task.update_solution_info`.

`dinfitem.sol_itg_pviolbarvar`
Maximal primal bound violation for \bar{X} in the integer solution. Updated if `iparam.auto_update_sol_info` is set or by the method `Task.update_solution_info`.

`dinfitem.sol_itg_pviolcon`
Maximal primal bound violation for x^c in the integer solution. Updated if `iparam.auto_update_sol_info` is set or by the method `Task.update_solution_info`.

`dinfitem.sol_itg_pviolcones`
Maximal primal violation for primal conic constraints in the integer solution. Updated if `iparam.auto_update_sol_info` is set or by the method `Task.update_solution_info`.

`dinfitem.sol_itg_pviolitg`
Maximal violation for the integer constraints in the integer solution. Updated if `iparam.auto_update_sol_info` is set or by the method `Task.update_solution_info`.

`dinfitem.sol_itg_pviolvar`
Maximal primal bound violation for x^x in the integer solution. Updated if `iparam.auto_update_sol_info` is set or by the method `Task.update_solution_info`.

`dinfitem.sol_itr_dual_obj`
Dual objective value of the interior-point solution. Updated if `iparam.auto_update_sol_info` is set or by the method `Task.update_solution_info`.

`dinfitem.sol_itr_dviolbarvar`
Maximal dual bound violation for \bar{X} in the interior-point solution. Updated if `iparam.auto_update_sol_info` is set or by the method `Task.update_solution_info`.

`dinfitem.sol_itr_dviolcon`
Maximal dual bound violation for x^c in the interior-point solution. Updated if `iparam.auto_update_sol_info` is set or by the method `Task.update_solution_info`.

`dinfitem.sol_itr_dviolcones`
Maximal dual violation for dual conic constraints in the interior-point solution. Updated if `iparam.auto_update_sol_info` is set or by the method `Task.update_solution_info`.

`dinfitem.sol_itr_dviolvar`
Maximal dual bound violation for x^x in the interior-point solution. Updated if `iparam.auto_update_sol_info` is set or by the method `Task.update_solution_info`.

`dinfitem.sol_itr_nrmBars`
Infinity norm of \bar{S} in the interior-point solution.

`dinfitem.sol_itr_nrm_barx`
Infinity norm of \bar{X} in the interior-point solution.

`dinfitem.sol_itr_nrm_slc`
Infinity norm of s_l^c in the interior-point solution.

`dinfitem.sol_itr_nrm_slx`
Infinity norm of s_l^x in the interior-point solution.

`dinfitem.sol_itr_nrm_snx`
Infinity norm of s_n^x in the interior-point solution.

`dinfitem.sol_itr_nrm_suc`
Infinity norm of s_u^c in the interior-point solution.

`dinfitem.sol_itr_nrm_sux`
Infinity norm of s_u^X in the interior-point solution.

`dinfitem.sol_itr_nrm_xc`
Infinity norm of x^c in the interior-point solution.

`dinfitem.sol_itr_nrm_xx`
Infinity norm of x^x in the interior-point solution.

`dinfitem.sol_itr_nrm_y`
Infinity norm of y in the interior-point solution.

`dinfitem.sol_itr_primal_obj`
Primal objective value of the interior-point solution. Updated if *iparam.auto_update_sol_info* is set or by the method *Task.update_solution_info*.

`dinfitem.sol_itr_pviolbarvar`
Maximal primal bound violation for \bar{X} in the interior-point solution. Updated if *iparam.auto_update_sol_info* is set or by the method *Task.update_solution_info*.

`dinfitem.sol_itr_pviolcon`
Maximal primal bound violation for x^c in the interior-point solution. Updated if *iparam.auto_update_sol_info* is set or by the method *Task.update_solution_info*.

`dinfitem.sol_itr_pviolcones`
Maximal primal violation for primal conic constraints in the interior-point solution. Updated if *iparam.auto_update_sol_info* is set or by the method *Task.update_solution_info*.

`dinfitem.sol_itr_pviolvar`
Maximal primal bound violation for x^x in the interior-point solution. Updated if *iparam.auto_update_sol_info* is set or by the method *Task.update_solution_info*.

`dinfitem.to_conic_time`
Time spent in the last to conic reformulation.

feature
License feature

feature.pts
Base system.

feature.pton
Conic extension.

liinfitem
Long integer information items.

`liinfitem.bi_clean_dual_deg_iter`
Number of dual degenerate clean iterations performed in the basis identification.

`liinfitem.bi_clean_dual_iter`
Number of dual clean iterations performed in the basis identification.

`liinfitem.bi_clean_primal_deg_iter`
Number of primal degenerate clean iterations performed in the basis identification.

`liinfitem.bi_clean_primal_iter`
Number of primal clean iterations performed in the basis identification.

`liinfitem.bi_dual_iter`
Number of dual pivots performed in the basis identification.

`liinfitem.bi_primal_iter`
Number of primal pivots performed in the basis identification.

`liinfitem.intpnt_factor_num_nz`
Number of non-zeros in factorization.

`liinfitem.mio_anz`
Number of non-zero entries in the constraint matrix of the problem to be solved by the mixed-integer optimizer.

`liinfitem.mio_intpnt_iter`
Number of interior-point iterations performed by the mixed-integer optimizer.

`liinfitem.mio_presolved_anz`
Number of non-zero entries in the constraint matrix of the problem after the mixed-integer optimizer's presolve.

`liinfitem.mio_simplex_iter`
Number of simplex iterations performed by the mixed-integer optimizer.

`liinfitem.rd_numanz`
Number of non-zeros in A that is read.

`liinfitem.rd_numqnz`
Number of Q non-zeros.

`iinfitem`
Integer information items.

`iinfitem.ana_pro_num_con`
Number of constraints in the problem. This value is set by *Task.analyzeproblem*.

`iinfitem.ana_pro_num_con_eq`
Number of equality constraints. This value is set by *Task.analyzeproblem*.

`iinfitem.ana_pro_num_con_fr`
Number of unbounded constraints. This value is set by *Task.analyzeproblem*.

`iinfitem.ana_pro_num_con_lo`
Number of constraints with a lower bound and an infinite upper bound. This value is set by *Task.analyzeproblem*.

`iinfitem.ana_pro_num_con_ra`
Number of constraints with finite lower and upper bounds. This value is set by *Task.analyzeproblem*.

`iinfitem.ana_pro_num_con_up`
Number of constraints with an upper bound and an infinite lower bound. This value is set by *Task.analyzeproblem*.

`iinfitem.ana_pro_num_var`
Number of variables in the problem. This value is set by *Task.analyzeproblem*.

`iinfitem.ana_pro_num_var_bin`
Number of binary (0-1) variables. This value is set by *Task.analyzeproblem*.

`iinfitem.ana_pro_num_var_cont`
Number of continuous variables. This value is set by *Task.analyzeproblem*.

`iinfitem.ana_pro_num_var_eq`
Number of fixed variables. This value is set by *Task.analyzeproblem*.

`iinfitem.ana_pro_num_var_fr`
Number of free variables. This value is set by *Task.analyzeproblem*.

`iinfitem.ana_pro_num_var_int`
Number of general integer variables. This value is set by *Task.analyzeproblem*.

`iinfitem.ana_pro_num_var_lo`
Number of variables with a lower bound and an infinite upper bound. This value is set by *Task.analyzeproblem*.

`iinfitem.ana_pro_num_var_ra`
Number of variables with finite lower and upper bounds. This value is set by *Task.analyzeproblem*.

`iinfitem.ana_pro_num_var_up`
Number of variables with an upper bound and an infinite lower bound. This value is set by *Task.analyzeproblem*.

`iinfitem.intpnt_factor_dim_dense`
Dimension of the dense sub system in factorization.

`iinfitem.intpnt_iter`
Number of interior-point iterations since invoking the interior-point optimizer.

`iinfitem.intpnt_num_threads`
Number of threads that the interior-point optimizer is using.

`iinfitem.intpnt_solve_dual`
Non-zero if the interior-point optimizer is solving the dual problem.

`iinfitem.mio_absgap_satisfied`
Non-zero if absolute gap is within tolerances.

`iinfitem.mio_clique_table_size`
Size of the clique table.

`iinfitem.mio_construct_solution`
This item informs if **MOSEK** constructed an initial integer feasible solution.

- -1: tried, but failed,
- 0: no partial solution supplied by the user,
- 1: constructed feasible solution.

`iinfitem.mio_node_depth`
Depth of the last node solved.

`iinfitem.mio_num_active_nodes`
Number of active branch and bound nodes.

`iinfitem.mio_num_branch`
Number of branches performed during the optimization.

`iinfitem.mio_num_clique_cuts`
Number of clique cuts.

`iinfitem.mio_num_cmir_cuts`
Number of Complemented Mixed Integer Rounding (CMIR) cuts.

`iinfitem.mio_num_gomory_cuts`
Number of Gomory cuts.

`iinfitem.mio_num_implied_bound_cuts`
Number of implied bound cuts.

`iinfitem.mio_num_int_solutions`
Number of integer feasible solutions that have been found.

`iinfitem.mio_num_knapsack_cover_cuts`
Number of clique cuts.

`iinfitem.mio_num_relax`
Number of relaxations solved during the optimization.

`iinfitem.mio_num_repeated_presolve`
Number of times presolve was repeated at root.

`iinfitem.mio_numbin`
Number of binary variables in the problem to be solved by the mixed-integer optimizer.

`iinfitem.mio_numbinconevar`
Number of binary cone variables in the problem to be solved by the mixed-integer optimizer.

`iinfitem.mio_numcon`
Number of constraints in the problem to be solved by the mixed-integer optimizer.

`iinfitem.mio_numcone`
Number of cones in the problem to be solved by the mixed-integer optimizer.

`iinfitem.mio_numconevar`
Number of cone variables in the problem to be solved by the mixed-integer optimizer.

`iinfitem.mio_numcont`
Number of continuous variables in the problem to be solved by the mixed-integer optimizer.

`iinfitem.mio_numcontconevar`
Number of continuous cone variables in the problem to be solved by the mixed-integer optimizer.

`iinfitem.mio_numdexpcones`
Number of dual exponential cones in the problem to be solved by the mixed-integer optimizer.

`iinfitem.mio_numdpowcones`
Number of dual power cones in the problem to be solved by the mixed-integer optimizer.

`iinfitem.mio_numint`
Number of integer variables in the problem to be solved by the mixed-integer optimizer.

`iinfitem.mio_numintconevar`
Number of integer cone variables in the problem to be solved by the mixed-integer optimizer.

`iinfitem.mio_numpexpcones`
Number of primal exponential cones in the problem to be solved by the mixed-integer optimizer.

`iinfitem.mio_numppowcones`
Number of primal power cones in the problem to be solved by the mixed-integer optimizer.

`iinfitem.mio_numqcones`
Number of quadratic cones in the problem to be solved by the mixed-integer optimizer.

`iinfitem.mio_numrqcones`
Number of rotated quadratic cones in the problem to be solved by the mixed-integer optimizer.

`iinfitem.mio_numvar`
Number of variables in the problem to be solved by the mixed-integer optimizer.

`iinfitem.mio_obj_bound_defined`
Non-zero if a valid objective bound has been found, otherwise zero.

`iinfitem.mio_presolved_numbin`
Number of binary variables in the problem after the mixed-integer optimizer's presolve.

`iinfitem.mio_presolved_numbinconevar`
Number of binary cone variables in the problem after the mixed-integer optimizer's presolve.

`iinfitem.mio_presolved_numcon`
Number of constraints in the problem after the mixed-integer optimizer's presolve.

`iinfitem.mio_presolved_numcone`
Number of cones in the problem after the mixed-integer optimizer's presolve.

`iinfitem.mio_presolved_numconevar`
Number of cone variables in the problem after the mixed-integer optimizer's presolve.

`iinfitem.mio_presolved_numcont`
Number of continuous variables in the problem after the mixed-integer optimizer's presolve.

`iinfitem.mio_presolved_numcontconevar`
Number of continuous cone variables in the problem after the mixed-integer optimizer's presolve.

`iinfitem.mio_presolved_numdexpcones`
Number of dual exponential cones in the problem after the mixed-integer optimizer's presolve.

`iinfitem.mio_presolved_numdpowcones`
Number of dual power cones in the problem after the mixed-integer optimizer's presolve.

`iinfitem.mio_presolved_numint`
Number of integer variables in the problem after the mixed-integer optimizer's presolve.

`iinfitem.mio_presolved_numintconevar`
Number of integer cone variables in the problem after the mixed-integer optimizer's presolve.

`iinfitem.mio_presolved_numpexpcones`
Number of primal exponential cones in the problem after the mixed-integer optimizer's presolve.

`iinfitem.mio_presolved_numppowcones`
Number of primal power cones in the problem after the mixed-integer optimizer's presolve.

`iinfitem.mio_presolved_numqcones`
Number of quadratic cones in the problem after the mixed-integer optimizer's presolve.

`iinfitem.mio_presolved_numrqcones`
Number of rotated quadratic cones in the problem after the mixed-integer optimizer's presolve.

`iinfitem.mio_presolved_numvar`
Number of variables in the problem after the mixed-integer optimizer's presolve.

`iinfitem.mio_relgap_satisfied`
Non-zero if relative gap is within tolerances.

`iinfitem.mio_total_num_cuts`
Total number of cuts generated by the mixed-integer optimizer.

`iinfitem.mio_user_obj_cut`
If it is non-zero, then the objective cut is used.

`iinfitem.opt_numcon`
Number of constraints in the problem solved when the optimizer is called.

`iinfitem.opt_numvar`
Number of variables in the problem solved when the optimizer is called

`iinfitem.optimize_response`
The response code returned by optimize.

`iinfitem.purify_dual_success`
Is nonzero if the dual solution is purified.

`iinfitem.purify_primal_success`
Is nonzero if the primal solution is purified.

`iinfitem.rd_numbarvar`
Number of symmetric variables read.

`iinfitem.rd_numcon`
Number of constraints read.

`iinfitem.rd_numcone`
Number of conic constraints read.

`iinfitem.rd_numintvar`
Number of integer-constrained variables read.

`iinfitem.rd_numq`
Number of nonempty Q matrices read.

`iinfitem.rd_numvar`
Number of variables read.

`iinfitem.rd_prototype`
Problem type.

`iinfitem.sim_dual_deg_iter`
The number of dual degenerate iterations.

`iinfitem.sim_dual_hotstart`
If 1 then the dual simplex algorithm is solving from an advanced basis.

`iinfitem.sim_dual_hotstart_lu`
If 1 then a valid basis factorization of full rank was located and used by the dual simplex algorithm.

`iinfitem.sim_dual_inf_iter`
The number of iterations taken with dual infeasibility.

`iinfitem.sim_dual_iter`
Number of dual simplex iterations during the last optimization.

`iinfitem.sim_numcon`
Number of constraints in the problem solved by the simplex optimizer.

`iinfitem.sim_numvar`
Number of variables in the problem solved by the simplex optimizer.

`iinfitem.sim_primal_deg_iter`
The number of primal degenerate iterations.

iinfitem.sim_primal_hotstart
If 1 then the primal simplex algorithm is solving from an advanced basis.

iinfitem.sim_primal_hotstart_lu
If 1 then a valid basis factorization of full rank was located and used by the primal simplex algorithm.

iinfitem.sim_primal_inf_iter
The number of iterations taken with primal infeasibility.

iinfitem.sim_primal_iter
Number of primal simplex iterations during the last optimization.

iinfitem.sim_solve_dual
Is non-zero if dual problem is solved.

iinfitem.sol_bas_prosta
Problem status of the basic solution. Updated after each optimization.

iinfitem.sol_bas_solsta
Solution status of the basic solution. Updated after each optimization.

iinfitem.sol_itg_prosta
Problem status of the integer solution. Updated after each optimization.

iinfitem.sol_itg_solsta
Solution status of the integer solution. Updated after each optimization.

iinfitem.sol_itr_prosta
Problem status of the interior-point solution. Updated after each optimization.

iinfitem.sol_itr_solsta
Solution status of the interior-point solution. Updated after each optimization.

iinfitem.sto_num_a_realloc
Number of times the storage for storing A has been changed. A large value may indicate that memory fragmentation may occur.

inftype
Information item types

inftype.dou_type
Is a double information type.

inftype.int_type
Is an integer.

inftype.lint_type
Is a long integer.

iomode
Input/output modes

iomode.read
The file is read-only.

iomode.write
The file is write-only. If the file exists then it is truncated when it is opened. Otherwise it is created when it is opened.

iomode.readwrite
The file is to read and write.

branchdir
Specifies the branching direction.

branchdir.free
The mixed-integer optimizer decides which branch to choose.

branchdir.up
The mixed-integer optimizer always chooses the up branch first.

branchdir.down
The mixed-integer optimizer always chooses the down branch first.

branchdir.near
Branch in direction nearest to selected fractional variable.

branchdir.far
Branch in direction farthest from selected fractional variable.

branchdir.root_lp
Chose direction based on root lp value of selected variable.

branchdir.guided
Branch in direction of current incumbent.

branchdir.pseudocost
Branch based on the pseudocost of the variable.

miocontsoltype
Continuous mixed-integer solution type

miocontsoltype.none
No interior-point or basic solution are reported when the mixed-integer optimizer is used.

miocontsoltype.root
The reported interior-point and basic solutions are a solution to the root node problem when mixed-integer optimizer is used.

miocontsoltype.itg
The reported interior-point and basic solutions are a solution to the problem with all integer variables fixed at the value they have in the integer solution. A solution is only reported in case the problem has a primal feasible solution.

miocontsoltype.itg_rel
In case the problem is primal feasible then the reported interior-point and basic solutions are a solution to the problem with all integer variables fixed at the value they have in the integer solution. If the problem is primal infeasible, then the solution to the root node problem is reported.

miomode
Integer restrictions

miomode.ignored
The integer constraints are ignored and the problem is solved as a continuous problem.

miomode.satisfied
Integer restrictions should be satisfied.

mionodeseltype
Mixed-integer node selection types

mionodeseltype.free
The optimizer decides the node selection strategy.

mionodeseltype.first
The optimizer employs a depth first node selection strategy.

mionodeseltype.best
The optimizer employs a best bound node selection strategy.

mionodeseltype.pseudo
The optimizer employs selects the node based on a pseudo cost estimate.

mpsformat
MPS file format type

mpsformat.strict
It is assumed that the input file satisfies the MPS format strictly.

mpsformat.relaxed
It is assumed that the input file satisfies a slightly relaxed version of the MPS format.

mpsformat.free
It is assumed that the input file satisfies the free MPS format. This implies that spaces are not allowed in names. Otherwise the format is free.

mpsformat.cplex
The CPLEX compatible version of the MPS format is employed.

objsense
Objective sense types

objsense.minimize
The problem should be minimized.

objsense.maximize
The problem should be maximized.

onoffkey
On/off

onoffkey.on
Switch the option on.

onoffkey.off
Switch the option off.

optimizertype
Optimizer types

optimizertype.conic
The optimizer for problems having conic constraints.

optimizertype.dual_simplex
The dual simplex optimizer is used.

optimizertype.free
The optimizer is chosen automatically.

optimizertype.free_simplex
One of the simplex optimizers is used.

optimizertype.intpnt
The interior-point optimizer is used.

optimizertype.mixed_int
The mixed-integer optimizer.

optimizertype.primal_simplex
The primal simplex optimizer is used.

orderingtype
Ordering strategies

orderingtype.free
The ordering method is chosen automatically.

orderingtype.appminloc
Approximate minimum local fill-in ordering is employed.

orderingtype.experimental
This option should not be used.

orderingtype.try_graphpar
Always try the graph partitioning based ordering.

orderingtype.force_graphpar
Always use the graph partitioning based ordering even if it is worse than the approximate minimum local fill ordering.

orderingtype.none
No ordering is used.

presolvemode
Presolve method.

presolvemode.off
The problem is not presolved before it is optimized.

presolvemode.on
The problem is presolved before it is optimized.

presolvemode.free
It is decided automatically whether to presolve before the problem is optimized.

parametertype
 Parameter type

parametertype.invalid_type
 Not a valid parameter.

parametertype.dou_type
 Is a double parameter.

parametertype.int_type
 Is an integer parameter.

parametertype.str_type
 Is a string parameter.

problemitem
 Problem data items

problemitem.var
 Item is a variable.

problemitem.con
 Item is a constraint.

problemitem.cone
 Item is a cone.

problemtyp
 Problem types

problemtyp.lo
 The problem is a linear optimization problem.

problemtyp.qo
 The problem is a quadratic optimization problem.

problemtyp.qcqo
 The problem is a quadratically constrained optimization problem.

problemtyp.conic
 A conic optimization.

problemtyp.mixed
 General nonlinear constraints and conic constraints. This combination can not be solved by **MOSEK**.

prosta
 Problem status keys

prosta.unknown
 Unknown problem status.

prosta.prim_and_dual_feas
 The problem is primal and dual feasible.

prosta.prim_feas
 The problem is primal feasible.

prosta.dual_feas
 The problem is dual feasible.

prosta.prim_infeas
 The problem is primal infeasible.

prosta.dual_infeas
 The problem is dual infeasible.

prosta.prim_and_dual_infeas
 The problem is primal and dual infeasible.

prosta.ill_posed
 The problem is ill-posed. For example, it may be primal and dual feasible but have a positive duality gap.

`prosta.prim_infeas_or_unbounded`
 The problem is either primal infeasible or unbounded. This may occur for mixed-integer problems.

`xmlwriteroutputtype`
 XML writer output mode

`xmlwriteroutputtype.row`
 Write in row order.

`xmlwriteroutputtype.col`
 Write in column order.

`rescodetype`
 Response code type

`rescodetype.ok`
 The response code is OK.

`rescodetype.wrn`
 The response code is a warning.

`rescodetype.trm`
 The response code is an optimizer termination status.

`rescodetype.err`
 The response code is an error.

`rescodetype.unk`
 The response code does not belong to any class.

`scalingtype`
 Scaling type

`scalingtype.free`
 The optimizer chooses the scaling heuristic.

`scalingtype.none`
 No scaling is performed.

`scalingtype.moderate`
 A conservative scaling is performed.

`scalingtype.aggressive`
 A very aggressive scaling is performed.

`scalingmethod`
 Scaling method

`scalingmethod.pow2`
 Scales only with power of 2 leaving the mantissa untouched.

`scalingmethod.free`
 The optimizer chooses the scaling heuristic.

`sensitivitytype`
 Sensitivity types

`sensitivitytype.basis`
 Basis sensitivity analysis is performed.

`simseltype`
 Simplex selection strategy

`simseltype.free`
 The optimizer chooses the pricing strategy.

`simseltype.full`
 The optimizer uses full pricing.

`simseltype.ase`
 The optimizer uses approximate steepest-edge pricing.

`simseltype.devex`
 The optimizer uses devex steepest-edge pricing (or if it is not available an approximate steep-edge selection).

`simseltype.se`
The optimizer uses steepest-edge selection (or if it is not available an approximate steep-edge selection).

`simseltype.partial`
The optimizer uses a partial selection approach. The approach is usually beneficial if the number of variables is much larger than the number of constraints.

`solitem`
Solution items

`solitem.xc`
Solution for the constraints.

`solitem.xx`
Variable solution.

`solitem.y`
Lagrange multipliers for equations.

`solitem.slc`
Lagrange multipliers for lower bounds on the constraints.

`solitem.suc`
Lagrange multipliers for upper bounds on the constraints.

`solitem.slx`
Lagrange multipliers for lower bounds on the variables.

`solitem.sux`
Lagrange multipliers for upper bounds on the variables.

`solitem.snx`
Lagrange multipliers corresponding to the conic constraints on the variables.

`solsta`
Solution status keys

`solsta.unknown`
Status of the solution is unknown.

`solsta.optimal`
The solution is optimal.

`solsta.prim_feas`
The solution is primal feasible.

`solsta.dual_feas`
The solution is dual feasible.

`solsta.prim_and_dual_feas`
The solution is both primal and dual feasible.

`solsta.prim_infeas_cer`
The solution is a certificate of primal infeasibility.

`solsta.dual_infeas_cer`
The solution is a certificate of dual infeasibility.

`solsta.prim_illposed_cer`
The solution is a certificate that the primal problem is illposed.

`solsta.dual_illposed_cer`
The solution is a certificate that the dual problem is illposed.

`solsta.integer_optimal`
The primal solution is integer optimal.

`soltype`
Solution types

`soltype.bas`
The basic solution.

`soltype.itr`
The interior solution.

soltype.itg
The integer solution.

solveform
Solve primal or dual form

solveform.free
The optimizer is free to solve either the primal or the dual problem.

solveform.primal
The optimizer should solve the primal problem.

solveform.dual
The optimizer should solve the dual problem.

stakey
Status keys

stakey.unk
The status for the constraint or variable is unknown.

stakey.bas
The constraint or variable is in the basis.

stakey.supbas
The constraint or variable is super basic.

stakey.low
The constraint or variable is at its lower bound.

stakey.upr
The constraint or variable is at its upper bound.

stakey.fix
The constraint or variable is fixed.

stakey.inf
The constraint or variable is infeasible in the bounds.

startpointtype
Starting point types

startpointtype.free
The starting point is chosen automatically.

startpointtype.guess
The optimizer guesses a starting point.

startpointtype.constant
The optimizer constructs a starting point by assigning a constant value to all primal and dual variables. This starting point is normally robust.

startpointtype.satisfy_bounds
The starting point is chosen to satisfy all the simple bounds on nonlinear variables. If this starting point is employed, then more care than usual should be employed when choosing the bounds on the nonlinear variables. In particular very tight bounds should be avoided.

streamtype
Stream types

streamtype.log
Log stream. Contains the aggregated contents of all other streams. This means that a message written to any other stream will also be written to this stream.

streamtype.msg
Message stream. Log information relating to performance and progress of the optimization is written to this stream.

streamtype.err
Error stream. Error messages are written to this stream.

streamtype.wrn
Warning stream. Warning messages are written to this stream.

value
Integer values

`value.max_str_len`
 Maximum string length allowed in **MOSEK**.
`value.license_buffer_length`
 The length of a license key buffer.
`variabletype`
 Variable types
`variabletype.type_cont`
 Is a continuous variable.
`variabletype.type_int`
 Is an integer variable.

15.10 Class types

`mosek.Progress`
 A handler class for progress callbacks.
`Progress.progress`

```
int progress (mosek.callbackcode code)
```

The progress callback is a user-defined function which will be called by **MOSEK** occasionally during the optimization process. In particular, the callback function is called at the beginning of each iteration in the interior-point optimizer. For the simplex optimizers *iparam.log_sim_freq* controls how frequently the callback is called.

The user *must not* call any **MOSEK** function directly or indirectly from the callback function.

Parameters `code` (*callbackcode*) – Callback code indicating current operation of the solver. (input)

Return (`int`) – Non-zero if the optimizer should be stopped; zero otherwise.

`mosek.ItgSolutionCallback`
 A handler class for integer solution callbacks.
`ItgSolutionCallback.callback`

```
void callback (double[] xx)
```

The integer solution callback is a user-defined function which will be called by **MOSEK** when it improves the best mixed-integer solution.

The user *must not* call any **MOSEK** function directly or indirectly from the callback function.

Parameters `xx` (`double[]`) – An array with the values of all variables in the currently best solution. (input)

`mosek.DataCallback`
 A handler class for data callbacks.
`DataCallback.callback`

```
int callback
(mosek.callbackcode code,
 double[] dinf,
 int[] iinf,
 long[] liinf)
```

The data callback is a user-defined function which will be called by **MOSEK** occasionally during the optimization process. In particular, the callback function is called at the beginning of each

iteration in the interior-point optimizer. For the simplex optimizers *iparam.log_sim_freq* controls how frequently the callback is called.

The user *must not* call any **MOSEK** function directly or indirectly from the callback function. The only exception is the possibility to retrieve an integer solution, see *Progress and data callback*.

Parameters

- `code` (*callbackcode*) – Callback code indicating current operation of the solver. (input)
- `dinf` (`double[]`) – Array of double information items. (input)
- `iinf` (`int[]`) – Array of integer information items. (input)
- `liinf` (`long[]`) – Array of long integer information items. (input)

Return (`int`) – Non-zero if the optimizer should be stopped; zero otherwise.

`mosek.Stream`

A stream handler class.

`Stream.stream`

```
void stream (String msg)
```

The message-stream callback function is a user-defined function which can be linked to any of the **MOSEK** streams. Doing so, the function is called whenever **MOSEK** sends a message to the stream.

The user *must not* call any **MOSEK** function directly or indirectly from the callback function.

Parameters `msg` (`String`) – A message passed to the handler. (input)

15.11 Nonlinear interfaces (obsolete)

Important: This is a legacy document for users familiar with SCopt, DGopt, EXPopt, mskenopt, mskscopt and mskgpopt from previous versions of **MOSEK**. These interfaces have now been removed. We assume familiarity with documentation included in version 8. All problems expressible with this interface can (and should) be reformulated using the exponential cone and power cones.

New users should formulate problems involving powers, logarithms and exponentials directly in conic form.

Conversion tutorial

We recommend converting all nonlinear problems using SCopt, DGopt, EXPopt, mskenopt, mskscopt and mskgpopt into conic form. Depending on the values of f, g, h either the epigraph or hypograph of a SCopt function if convex, and a bounding variable can be introduced following the basic rules below. We assume all variables are within safe bounds where the SCopt operators are defined and convex. We also assume $f > 0$.

A more comprehensive modeling guide for these types of problems can be found in the **MOSEK Modeling Cookbook**.

Powers

Consider $f(x+h)^g$. This can be reformulated using the power cone.

- If $g > 1$ then $t \geq f(x+h)^g$ is equivalent to $(t/f)^{1/g} \geq |x+h|$, that is $(t/f, 1, x+h) \in \mathcal{P}_3^{1/g, 1-1/g}$.
- If $0 < g < 1$ then $|t| \leq f(x+h)^g$ is equivalent to $(x+h, 1, t/f) \in \mathcal{P}_3^{g, 1-g}$.
- If $g < 0$ then $t \geq f(x+h)^g$ is equivalent to $(t/f)(x+h)^{-g} \geq 1$, that is $(t/f, x+h, 1) \in \mathcal{P}_3^{1/(1-g), -g/(1-g)}$.

Logarithm

The bound $t \leq f \log(gx+h)$ is equivalent to $(gx+h, 1, t/f) \in K_{\text{exp}}$.

Entropy

The bound $t \geq f x \log x$ is equivalent to $(1, x, -t/f) \in K_{\text{exp}}$.

Exponential

The bound $t \geq f \exp(gx+h)$ is equivalent to $(t/f, 1, gx+h) \in K_{\text{exp}}$.

Exponential optimization (EXOpt), Geometric programming (mskgpopt)

For a basic tutorial in geometric programming (GP) see [Sec. 6.8](#).

An exponential optimization problem in standard form consists of constraints of the type:

$$t \geq \log \left(\sum_i \exp(a_i^T x + b_i) \right).$$

This log-sum-exp bound is equivalent to

$$\sum_i \exp(a_i^T x + b_i - t) \leq 1$$

and requires bounding each exponential function as explained above.

Dual geometric optimization (DGopt)

The objective function of a dual geometric problem involves maximizing expressions of the form

$$x \log \frac{c}{x} \quad \text{and} \quad x_i \log \frac{e^T x}{x_i},$$

which can be achieved using bounds $t \leq x \log \frac{y}{x}$, that is $(t, x, y) \in K_{\text{exp}}$.

Chapter 16

Supported File Formats

MOSEK supports a range of problem and solution formats listed in [Table 16.1](#) and [Table 16.2](#). The **Task format** is **MOSEK**'s native binary format and it supports all features that **MOSEK** supports. The **OPF format** is **MOSEK**'s human-readable alternative that supports nearly all features (everything except semidefinite problems). In general, text formats are significantly slower to read, but can be examined and edited directly in any text editor.

Problem formats

Table 16.1: List of supported file formats for optimization problems. The column *Conic* refers to conic problems involving the quadratic, rotated quadratic, power or exponential cone. The last two columns indicate if the format supports solutions and optimizer parameters.

Format Type	Ext.	Binary/Text	LP	QO	Conic	SDP	Sol	Param
<i>LP</i>	lp	plain text	X	X				
<i>MPS</i>	mps	plain text	X	X	X			
<i>OPF</i>	opf	plain text	X	X	X		X	X
<i>PTF</i>	ptf	plain text	X	X	X	X	X	
<i>CBF</i>	cbf	plain text	X		X	X		
<i>Task format</i>	task	binary	X	X	X	X	X	X
<i>Jtask format</i>	jtask	text	X	X	X	X	X	X

Solution formats

Table 16.2: List of supported solution formats.

Format Type	Ext.	Binary/Text	Description
<i>SOL</i>	sol	plain text	Interior Solution
	bas	plain text	Basic Solution
	int	plain text	Integer
<i>Jsol format</i>	jsol	text	Solution

Compression

MOSEK supports GZIP and Zstandard compression. Problem files with extension `.gz` (for GZIP) and `.zst` (for Zstandard) are assumed to be compressed when read, and are automatically compressed when written. For example, a file called

`problem.mps.gz`

will be considered as a GZIP compressed MPS file.

16.1 The LP File Format

MOSEK supports the LP file format with some extensions. The LP format is not a completely well-defined standard and hence different optimization packages may interpret the same LP file in slightly different ways. **MOSEK** tries to emulate as closely as possible CPLEX's behavior, but tries to stay backward compatible.

The LP file format can specify problems of the form

$$\begin{array}{ll} \text{minimize/maximize} & c^T x + \frac{1}{2} q^o(x) \\ \text{subject to} & \begin{array}{ll} l^c \leq Ax + \frac{1}{2} q(x) \leq u^c, \\ l^x \leq x \leq u^x, \\ x_{\mathcal{J}} \text{ integer,} \end{array} \end{array}$$

where

- $x \in \mathbb{R}^n$ is the vector of decision variables.
- $c \in \mathbb{R}^n$ is the linear term in the objective.
- $q^o : \mathbb{R}^n \rightarrow \mathbb{R}$ is the quadratic term in the objective where

$$q^o(x) = x^T Q^o x$$

and it is assumed that

$$Q^o = (Q^o)^T.$$

- $A \in \mathbb{R}^{m \times n}$ is the constraint matrix.
- $l^c \in \mathbb{R}^m$ is the lower limit on the activity for the constraints.
- $u^c \in \mathbb{R}^m$ is the upper limit on the activity for the constraints.
- $l^x \in \mathbb{R}^n$ is the lower limit on the activity for the variables.
- $u^x \in \mathbb{R}^n$ is the upper limit on the activity for the variables.
- $q : \mathbb{R}^n \rightarrow \mathbb{R}$ is a vector of quadratic functions. Hence,

$$q_i(x) = x^T Q^i x$$

where it is assumed that

$$Q^i = (Q^i)^T.$$

- $\mathcal{J} \subseteq \{1, 2, \dots, n\}$ is an index set of the integer constrained variables.

16.1.1 File Sections

An LP formatted file contains a number of sections specifying the objective, constraints, variable bounds, and variable types. The section keywords may be any mix of upper and lower case letters.

Objective Function

The first section beginning with one of the keywords

```
max
maximum
maximize
min
minimum
minimize
```

defines the objective sense and the objective function, i.e.

$$c^T x + \frac{1}{2} x^T Q x.$$

The objective may be given a name by writing

```
myname:
```

before the expressions. If no name is given, then the objective is named **obj**.

The objective function contains linear and quadratic terms. The linear terms are written as

```
4 x1 + x2 - 0.1 x3
```

and so forth. The quadratic terms are written in square brackets (`[]/2`) and are either squared or multiplied as in the examples

```
x1^2
```

and

```
x1 * x2
```

There may be zero or more pairs of brackets containing quadratic expressions.

An example of an objective section is

```
minimize
myobj: 4 x1 + x2 - 0.1 x3 + [ x1^2 + 2.1 x1 * x2 ]/2
```

Please note that the quadratic expressions are multiplied with $\frac{1}{2}$, so that the above expression means

$$\text{minimize } 4x_1 + x_2 - 0.1 \cdot x_3 + \frac{1}{2}(x_1^2 + 2.1 \cdot x_1 \cdot x_2)$$

If the same variable occurs more than once in the linear part, the coefficients are added, so that `4 x1 + 2 x1` is equivalent to `6 x1`. In the quadratic expressions `x1 * x2` is equivalent to `x2 * x1` and, as in the linear part, if the same variables multiplied or squared occur several times their coefficients are added.

Constraints

The second section beginning with one of the keywords

```
subj to
subject to
s.t.
st
```

defines the linear constraint matrix A and the quadratic matrices Q^i .

A constraint contains a name (optional), expressions adhering to the same rules as in the objective and a bound:

```
subject to
con1: x1 + x2 + [ x3^2 ]/2 <= 5.1
```

The bound type (here \leq) may be any of $<$, \leq , $=$, $>$, \geq ($<$ and \leq mean the same), and the bound may be any number.

In the standard LP format it is not possible to define more than one bound per line, but **MOSEK** supports defining ranged constraints by using double-colon ($::$) instead of a single-colon ($:$) after the constraint name, i.e.

$$-5 \leq x_1 + x_2 \leq 5 \quad (16.1)$$

may be written as

```
con:: -5 < x_1 + x_2 < 5
```

By default **MOSEK** writes ranged constraints this way.

If the files must adhere to the LP standard, ranged constraints must either be split into upper bounded and lower bounded constraints or be written as an equality with a slack variable. For example the expression (16.1) may be written as

$$x_1 + x_2 - sl_1 = 0, \quad -5 \leq sl_1 \leq 5.$$

Bounds

Bounds on the variables can be specified in the bound section beginning with one of the keywords

```
bound
bounds
```

The bounds section is optional but should, if present, follow the **subject to** section. All variables listed in the bounds section must occur in either the objective or a constraint.

The default lower and upper bounds are 0 and $+\infty$. A variable may be declared free with the keyword **free**, which means that the lower bound is $-\infty$ and the upper bound is $+\infty$. Furthermore it may be assigned a finite lower and upper bound. The bound definitions for a given variable may be written in one or two lines, and bounds can be any number or $\pm\infty$ (written as **+inf/-inf/+infinity/-infinity**) as in the example

```
bounds
x1 free
x2 <= 5
0.1 <= x2
x3 = 42
2 <= x4 < +inf
```

Variable Types

The final two sections are optional and must begin with one of the keywords

```
bin
binaries
binary
```

and

```
gen
general
```

Under **general** all integer variables are listed, and under **binary** all binary (integer variables with bounds 0 and 1) are listed:

```
general
x1 x2
binary
x3 x4
```

Again, all variables listed in the binary or general sections must occur in either the objective or a constraint.

Terminating Section

Finally, an LP formatted file must be terminated with the keyword

```
end
```

16.1.2 LP File Examples

Linear example lo1.lp

```
\ File: lo1.lp
maximize
obj: 3 x1 + x2 + 5 x3 + x4
subject to
c1: 3 x1 + x2 + 2 x3 = 30
c2: 2 x1 + x2 + 3 x3 + x4 >= 15
c3: 2 x2 + 3 x4 <= 25
bounds
0 <= x1 <= +infinity
0 <= x2 <= 10
0 <= x3 <= +infinity
0 <= x4 <= +infinity
end
```

Mixed integer example milo1.lp

```
maximize
obj: x1 + 6.4e-01 x2
subject to
c1: 5e+01 x1 + 3.1e+01 x2 <= 2.5e+02
c2: 3e+00 x1 - 2e+00 x2 >= -4e+00
bounds
0 <= x1 <= +infinity
0 <= x2 <= +infinity
general
x1 x2
end
```

16.1.3 LP Format peculiarities

Comments

Anything on a line after a \ is ignored and is treated as a comment.

Names

A name for an objective, a constraint or a variable may contain the letters **a-z**, **A-Z**, the digits **0-9** and the characters

!"#\$%&() / , . ; ? @ _ ' ` ~

The first character in a name must not be a number, a period or the letter **e** or **E**. Keywords must not be used as names.

MOSEK accepts any character as valid for names, except \0. A name that is not allowed in LP file will be changed and a warning will be issued.

The algorithm for making names LP valid works as follows: The name is interpreted as an **utf-8** string. For a Unicode character **c**:

- If **c**==_ (underscore), the output is __ (two underscores).
- If **c** is a valid LP name character, the output is just **c**.
- If **c** is another character in the ASCII range, the output is _XX, where XX is the hexadecimal code for the character.
- If **c** is a character in the range 127-65535, the output is _uXXXX, where XXXX is the hexadecimal code for the character.
- If **c** is a character above 65535, the output is _UXXXXXXXX, where XXXXXXXX is the hexadecimal code for the character.

Invalid **utf-8** substrings are escaped as _XX', and if a name starts with a period, **e** or **E**, that character is escaped as _XX.

Variable Bounds

Specifying several upper or lower bounds on one variable is possible but **MOSEK** uses only the tightest bounds. If a variable is fixed (with =), then it is considered the tightest bound.

MOSEK Extensions to the LP Format

Some optimization software packages employ a more strict definition of the LP format than the one used by **MOSEK**. The limitations imposed by the strict LP format are the following:

- Quadratic terms in the constraints are not allowed.
- Names can be only 16 characters long.
- Lines must not exceed 255 characters in length.

To get around some of the inconveniences converting from other problem formats, **MOSEK** allows lines to contain 1024 characters and names may have any length (shorter than the 1024 characters).

If an LP formatted file created by **MOSEK** should satisfy the strict definition, then the parameter *iparam.write_lp_strict_format* should be set; note, however, that some problems cannot be written correctly as a strict LP formatted file. For instance, all names are truncated to 16 characters and hence they may lose their uniqueness and change the problem.

Internally in **MOSEK** names may contain any (printable) character, many of which cannot be used in LP names. Setting the parameters *iparam.read_lp_quoted_names* and *iparam.write_lp_quoted_names* allows **MOSEK** to use quoted names. The first parameter tells **MOSEK** to remove quotes from quoted names e.g. "x1", when reading LP formatted files. The second parameter tells **MOSEK** to put quotes around any semi-illegal name (names beginning with a number or a period) and fully illegal name (containing illegal characters). As double quote is a legal character in the LP format, quoting semi-illegal names makes them legal in the pure LP format as long as they are still shorter than 16 characters. Fully illegal names are still illegal in a pure LP file.

The strict LP format

The LP format is not a formal standard and different vendors have slightly different interpretations of the LP format. To make **MOSEK**'s definition of the LP format more compatible with the definitions of other vendors set the parameter `iparam.write_lp_strict_format` to `onoffkey.on`.

This setting may lead to truncation of some names and hence to an invalid LP file. The simple solution to this problem is to set the parameter `iparam.write_generic_names` to `onoffkey.on` which will cause all names to be renamed systematically in the output file.

Formatting of an LP File

A few parameters control the visual formatting of LP files written by **MOSEK** in order to make it easier to read the files. These parameters are

- `iparam.write_lp_line_width` sets the maximum number of characters on a single line. The default value is 80 corresponding roughly to the width of a standard text document.
- `iparam.write_lp_terms_per_line` sets the maximum number of terms per line; a term means a sign, a coefficient, and a name (for example `+ 42 elephants`). The default value is 0, meaning that there is no maximum.

Unnamed Constraints

Reading and writing an LP file with **MOSEK** may change it superficially. If an LP file contains unnamed constraints or objective these are given their generic names when the file is read (however unnamed constraints in **MOSEK** are written without names).

16.2 The MPS File Format

MOSEK supports the standard MPS format with some extensions. For a detailed description of the MPS format see the book by Nazareth [Naz87].

16.2.1 MPS File Structure

The version of the MPS format supported by **MOSEK** allows specification of an optimization problem of the form

$$\begin{aligned} \text{maximize/minimize} \quad & c^T x + q_0(x) \\ l^c \leq \quad & Ax + q(x) \leq u^c, \\ l^x \leq \quad & x \leq u^x, \\ & x \in \mathcal{K}, \\ & x_{\mathcal{J}} \text{ integer}, \end{aligned} \tag{16.2}$$

where

- $x \in \mathbb{R}^n$ is the vector of decision variables.
- $A \in \mathbb{R}^{m \times n}$ is the constraint matrix.
- $l^c \in \mathbb{R}^m$ is the lower limit on the activity for the constraints.
- $u^c \in \mathbb{R}^m$ is the upper limit on the activity for the constraints.
- $l^x \in \mathbb{R}^n$ is the lower limit on the activity for the variables.
- $u^x \in \mathbb{R}^n$ is the upper limit on the activity for the variables.
- $q : \mathbb{R}^n \rightarrow \mathbb{R}$ is a vector of quadratic functions. Hence,

$$q_i(x) = \frac{1}{2} x^T Q^i x$$

where it is assumed that $Q^i = (Q^i)^T$. Please note the explicit $\frac{1}{2}$ in the quadratic term and that Q^i is required to be symmetric. The same applies to q_0 .

- \mathcal{K} is a convex cone.
- $\mathcal{J} \subseteq \{1, 2, \dots, n\}$ is an index set of the integer-constrained variables.
- c is the vector of objective coefficients.

An MPS file with one row and one column can be illustrated like this:

```
*          1          2          3          4          5          6
*23456789012345678901234567890123456789012345678901234567890
NAME          [name]
OBJSENSE
    [objsense]
OBJNAME          [objname]
ROWS
?  [cname1]
COLUMNS
    [vname1]  [cname1]  [value1]          [cname2]  [value2]
RHS
    [name]    [cname1]  [value1]          [cname2]  [value2]
RANGES
    [name]    [cname1]  [value1]          [cname2]  [value2]
QSECTION          [cname1]
    [vname1]  [vname2]  [value1]          [vname3]  [value2]
QMATRIX
    [vname1]  [vname2]  [value1]
QUADOBJ
    [vname1]  [vname2]  [value1]
QCMATRIX          [cname1]
    [vname1]  [vname2]  [value1]
BOUNDS
?? [name]      [vname1]  [value1]
CSECTION          [kname1]  [value1]          [ktype]
    [vname1]
ENDATA
```

Here the names in capitals are keywords of the MPS format and names in brackets are custom defined names or values. A couple of notes on the structure:

- Fields: All items surrounded by brackets appear in *fields*. The fields named “valueN” are numerical values. Hence, they must have the format

$[+|-]XXXXXXXX.XXXXXX[e|E][+|-]XXX$

where

$X = [0|1|2|3|4|5|6|7|8|9].$

- Sections: The MPS file consists of several sections where the names in capitals indicate the beginning of a new section. For example, COLUMNS denotes the beginning of the columns section.
- Comments: Lines starting with an * are comment lines and are ignored by **MOSEK**.
- Keys: The question marks represent keys to be specified later.
- Extensions: The sections QSECTION and CSECTION are specific **MOSEK** extensions of the MPS format. The sections QMATRIX, QUADOBJ and QCMATRIX are included for sake of compatibility with other vendors extensions to the MPS format.
- The standard MPS format is a fixed format, i.e. everything in the MPS file must be within certain fixed positions. **MOSEK** also supports a *free format*. See [Sec. 16.2.5](#) for details.

Linear example lo1.mps

A concrete example of a MPS file is presented below:

```
* File: lo1.mps
NAME          lo1
OBJSENSE
    MAX
ROWS
    N  obj
    E  c1
    G  c2
    L  c3
COLUMNS
    x1      obj      3
    x1      c1       3
    x1      c2       2
    x2      obj      1
    x2      c1       1
    x2      c2       1
    x2      c3       2
    x3      obj      5
    x3      c1       2
    x3      c2       3
    x4      obj      1
    x4      c2       1
    x4      c3       3
RHS
    rhs     c1      30
    rhs     c2      15
    rhs     c3      25
RANGES
BOUNDS
    UP bound    x2      10
ENDATA
```

Subsequently each individual section in the MPS format is discussed.

NAME (optional)

In this section a name ([name]) is assigned to the problem.

OBJSENSE (optional)

This is an optional section that can be used to specify the sense of the objective function. The **OBJSENSE** section contains one line at most which can be one of the following:

```
MIN
MINIMIZE
MAX
MAXIMIZE
```

It should be obvious what the implication is of each of these four lines.

OBJNAME (optional)

This is an optional section that can be used to specify the name of the row that is used as objective function. `objname` should be a valid row name.

ROWS

A record in the ROWS section has the form

? [cname1]

where the requirements for the fields are as follows:

Field	Starting Position	Max Width	required	Description
?	2	1	Yes	Constraint key
[cname1]	5	8	Yes	Constraint name

Hence, in this section each constraint is assigned a unique name denoted by `[cname1]`. Please note that `[cname1]` starts in position 5 and the field can be at most 8 characters wide. An initial key ? must be present to specify the type of the constraint. The key can have values E, G, L, or N with the following interpretation:

Constraint type	l_i^c	u_i^c
E (equal)	finite	$= l_i^c$
G (greater)	finite	∞
L (lower)	$-\infty$	finite
N (none)	$-\infty$	∞

In the MPS format the objective vector is not specified explicitly, but one of the constraints having the key N will be used as the objective vector c . In general, if multiple N type constraints are specified, then the first will be used as the objective vector c , unless something else was specified in the section OBJNAME.

COLUMNS

In this section the elements of A are specified using one or more records having the form:

[vname1]	[cname1]	[value1]	[cname2]	[value2]
----------	----------	----------	----------	----------

where the requirements for each field are as follows:

Field	Starting Position	Max Width	required	Description
[vname1]	5	8	Yes	Variable name
[cname1]	15	8	Yes	Constraint name
[value1]	25	12	Yes	Numerical value
[cname2]	40	8	No	Constraint name
[value2]	50	12	No	Numerical value

Hence, a record specifies one or two elements a_{ij} of A using the principle that `[vname1]` and `[cname1]` determines j and i respectively. Please note that `[cname1]` must be a constraint name specified in the ROWS section. Finally, `[value1]` denotes the numerical value of a_{ij} . Another optional element is specified by `[cname2]`, and `[value2]` for the variable specified by `[vname1]`. Some important comments are:

- All elements belonging to one variable must be grouped together.
- Zero elements of A should not be specified.
- At least one element for each variable should be specified.

RHS (optional)

A record in this section has the format

[name]	[cname1]	[value1]	[cname2]	[value2]
--------	----------	----------	----------	----------

where the requirements for each field are as follows:

Field	Starting Position	Max Width	required	Description
[name]	5	8	Yes	Name of the RHS vector
[cname1]	15	8	Yes	Constraint name
[value1]	25	12	Yes	Numerical value
[cname2]	40	8	No	Constraint name
[value2]	50	12	No	Numerical value

The interpretation of a record is that [name] is the name of the RHS vector to be specified. In general, several vectors can be specified. [cname1] denotes a constraint name previously specified in the ROWS section. Now, assume that this name has been assigned to the i -th constraint and v_1 denotes the value specified by [value1], then the interpretation of v_1 is:

Constraint	l_i^c	u_i^c
E	v_1	v_1
G	v_1	
L		v_1
N		

An optional second element is specified by [cname2] and [value2] and is interpreted in the same way. Please note that it is not necessary to specify zero elements, because elements are assumed to be zero.

RANGES (optional)

A record in this section has the form

[name]	[cname1]	[value1]	[cname2]	[value2]
--------	----------	----------	----------	----------

where the requirements for each fields are as follows:

Field	Starting Position	Max Width	required	Description
[name]	5	8	Yes	Name of the RANGE vector
[cname1]	15	8	Yes	Constraint name
[value1]	25	12	Yes	Numerical value
[cname2]	40	8	No	Constraint name
[value2]	50	12	No	Numerical value

The records in this section are used to modify the bound vectors for the constraints, i.e. the values in l^c and u^c . A record has the following interpretation: [name] is the name of the RANGE vector and [cname1] is a valid constraint name. Assume that [cname1] is assigned to the i -th constraint and let v_1 be the value specified by [value1], then a record has the interpretation:

Constraint type	Sign of v_1	l_i^c	u_i^c
E	—	$u_i^c + v_1$	
E	+		$l_i^c + v_1$
G	— or +		$l_i^c + v_1 $
L	— or +	$u_i^c - v_1 $	
N			

Another constraint bound can optionally be modified using [cname2] and [value2] the same way.

QSECTION (optional)

Within the QSECTION the label [cname1] must be a constraint name previously specified in the ROWS section. The label [cname1] denotes the constraint to which the quadratic terms belong. A record in the QSECTION has the form

[vname1]	[vname2]	[value1]	[vname3]	[value2]
----------	----------	----------	----------	----------

where the requirements for each field are:

Field	Starting Position	Max Width	required	Description
[vname1]	5	8	Yes	Variable name
[vname2]	15	8	Yes	Variable name
[value1]	25	12	Yes	Numerical value
[vname3]	40	8	No	Variable name
[value2]	50	12	No	Numerical value

A record specifies one or two elements in the lower triangular part of the Q^i matrix where [cname1] specifies the i . Hence, if the names [vname1] and [vname2] have been assigned to the k -th and j -th variable, then Q_{kj}^i is assigned the value given by [value1]. An optional second element is specified in the same way by the fields [vname1], [vname3], and [value2].

The example

$$\begin{array}{ll} \text{minimize} & -x_2 + \frac{1}{2}(2x_1^2 - 2x_1x_3 + 0.2x_2^2 + 2x_3^2) \\ \text{subject to} & x_1 + x_2 + x_3 \geq 1, \\ & x \geq 0 \end{array}$$

has the following MPS file representation

```
* File: qo1.mps
NAME          qo1
ROWS
  N  obj
  G  c1
COLUMNS
  x1      c1      1.0
  x2      obj     -1.0
  x2      c1      1.0
  x3      c1      1.0
RHS
  rhs     c1      1.0
QSECTION  obj
  x1      x1      2.0
  x1      x3     -1.0
  x2      x2      0.2
  x3      x3      2.0
ENDATA
```

Regarding the QSECTIONS please note that:

- Only one QSECTION is allowed for each constraint.
- The QSECTIONS can appear in an arbitrary order after the COLUMNS section.
- All variable names occurring in the QSECTION must already be specified in the COLUMNS section.
- All entries specified in a QSECTION are assumed to belong to the lower triangular part of the quadratic term of Q .

QMATRIX/QUADOBJ (optional)

The QMATRIX and QUADOBJ sections allow to define the quadratic term of the objective function. They differ in how the quadratic term of the objective function is stored:

- QMATRIX stores all the nonzeros coefficients, without taking advantage of the symmetry of the Q matrix.
- QUADOBJ stores the upper diagonal nonzero elements of the Q matrix.

A record in both sections has the form:

[vname1]	[vname2]	[value1]
----------	----------	----------

where the requirements for each field are:

Field	Starting Position	Max Width	required	Description
[vname1]	5	8	Yes	Variable name
[vname2]	15	8	Yes	Variable name
[value1]	25	12	Yes	Numerical value

A record specifies one elements of the Q matrix in the objective function . Hence, if the names [vname1] and [vname2] have been assigned to the k -th and j -th variable, then Q_{kj} is assigned the value given by [value1]. Note that a line must appear for each off-diagonal coefficient if using a QMATRIX section, while only one entry is required in a QUADOBJ section. The quadratic part of the objective function will be evaluated as $1/2x^T Qx$.

The example

$$\begin{array}{ll} \text{minimize} & -x_2 + \frac{1}{2}(2x_1^2 - 2x_1x_3 + 0.2x_2^2 + 2x_3^2) \\ \text{subject to} & x_1 + x_2 + x_3 \geq 1, \\ & x \geq 0 \end{array}$$

has the following MPS file representation using QMATRIX

```
* File: qo1_matrix.mps
NAME          qo1_qmatrix
ROWS
  N  obj
  G  c1
COLUMNS
  x1      c1      1.0
  x2      obj     -1.0
  x2      c1      1.0
  x3      c1      1.0
RHS
  rhs     c1      1.0
QMATRIX
  x1      x1      2.0
  x1      x3     -1.0
  x3      x1     -1.0
  x2      x2      0.2
  x3      x3      2.0
ENDATA
```

or the following using QUADOBJ

```
* File: qo1_quadobj.mps
NAME          qo1_quadobj
ROWS
  N  obj
  G  c1
```

(continues on next page)

(continued from previous page)

COLUMNS		
x1	c1	1.0
x2	obj	-1.0
x2	c1	1.0
x3	c1	1.0
RHS		
rhs	c1	1.0
QUADOBJ		
x1	x1	2.0
x1	x3	-1.0
x2	x2	0.2
x3	x3	2.0
ENDATA		

Please also note that:

- A QMATRIX/QUADOBJ section can appear in an arbitrary order after the COLUMNS section.
- All variable names occurring in the QMATRIX/QUADOBJ section must already be specified in the COLUMNS section.

QMATRIX (optional)

A QMATRIX section allows to specify the quadratic part of a given constraint. Within the QMATRIX the label [cname1] must be a constraint name previously specified in the ROWS section. The label [cname1] denotes the constraint to which the quadratic term belongs. A record in the QSECTION has the form

[vname1]	[vname2]	[value1]
----------	----------	----------

where the requirements for each field are:

Field	Starting Position	Max Width	required	Description
[vname1]	5	8	Yes	Variable name
[vname2]	15	8	Yes	Variable name
[value1]	25	12	Yes	Numerical value

A record specifies an entry of the Q^i matrix where [cname1] specifies the i . Hence, if the names [vname1] and [vname2] have been assigned to the k -th and j -th variable, then Q_{kj}^i is assigned the value given by [value1]. Moreover, the quadratic term is represented as $1/2x^T Qx$.

The example

$$\begin{aligned}
 &\text{minimize} && x_2 \\
 &\text{subject to} && x_1 + x_2 + x_3 \geq 1, \\
 &&& \frac{1}{2}(-2x_1x_3 + 0.2x_2^2 + 2x_3^2) \leq 10, \\
 &&& x \geq 0
 \end{aligned}$$

has the following MPS file representation

* File: qo1.mps		
NAME	qo1	
ROWS		
N	obj	
G	c1	
L	q1	
COLUMNS		
x1	c1	1.0
x2	obj	-1.0
x2	c1	1.0
x3	c1	1.0

(continues on next page)

```

RHS
  rhs      c1      1.0
  rhs      q1     10.0
QCMATRIX   q1
  x1      x1      2.0
  x1      x3     -1.0
  x3      x1     -1.0
  x2      x2      0.2
  x3      x3      2.0
ENDATA

```

Regarding the QCMATRIXs please note that:

- Only one QCMATRIX is allowed for each constraint.
- The QCMATRIXs can appear in an arbitrary order after the COLUMNS section.
- All variable names occurring in the QSECTION must already be specified in the COLUMNS section.
- QCMATRIX does not exploit the symmetry of Q : an off-diagonal entry (i, j) should appear twice.

BOUNDS (optional)

In the BOUNDS section changes to the default bounds vectors l^x and u^x are specified. The default bounds vectors are $l^x = 0$ and $u^x = \infty$. Moreover, it is possible to specify several sets of bound vectors. A record in this section has the form

```
?? [name]    [vname1]    [value1]
```

where the requirements for each field are:

Field	Starting Position	Max Width	Required	Description
??	2	2	Yes	Bound key
[name]	5	8	Yes	Name of the BOUNDS vector
[vname1]	15	8	Yes	Variable name
[value1]	25	12	No	Numerical value

Hence, a record in the BOUNDS section has the following interpretation: [name] is the name of the bound vector and [vname1] is the name of the variable for which the bounds are modified by the record. ?? and [value1] are used to modify the bound vectors according to the following table:

??	l_j^x	u_j^x	Made integer (added to \mathcal{J})
FR	$-\infty$	∞	No
FX	v_1	v_1	No
LO	v_1	unchanged	No
MI	$-\infty$	unchanged	No
PL	unchanged	∞	No
UP	unchanged	v_1	No
BV	0	1	Yes
LI	$\lceil v_1 \rceil$	unchanged	Yes
UI	unchanged	$\lfloor v_1 \rfloor$	Yes

Here v_1 is the value specified by [value1].

CSECTION (optional)

The purpose of the CSECTION is to specify the conic constraint

$$x \in \mathcal{K}$$

in (16.2). It is assumed that \mathcal{K} satisfies the following requirements. Let

$$x^t \in \mathbb{R}^{n^t}, \quad t = 1, \dots, k$$

be vectors comprised of parts of the decision variables x so that each decision variable is a member of exactly **one** vector x^t , for example

$$x^1 = \begin{bmatrix} x_1 \\ x_4 \\ x_7 \end{bmatrix} \quad \text{and} \quad x^2 = \begin{bmatrix} x_6 \\ x_5 \\ x_3 \\ x_2 \end{bmatrix}.$$

Next define

$$\mathcal{K} := \{x \in \mathbb{R}^n : x^t \in \mathcal{K}_t, \quad t = 1, \dots, k\}$$

where \mathcal{K}_t must have one of the following forms:

- \mathbb{R} set:

$$\mathcal{K}_t = \mathbb{R}^{n^t}.$$

- Zero cone:

$$\mathcal{K}_t = \{0\} \subseteq \mathbb{R}^{n^t}. \quad (16.3)$$

- Quadratic cone:

$$\mathcal{K}_t = \left\{ x \in \mathbb{R}^{n^t} : x_1 \geq \sqrt{\sum_{j=2}^{n^t} x_j^2} \right\}. \quad (16.4)$$

- Rotated quadratic cone:

$$\mathcal{K}_t = \left\{ x \in \mathbb{R}^{n^t} : 2x_1x_2 \geq \sum_{j=3}^{n^t} x_j^2, \quad x_1, x_2 \geq 0 \right\}. \quad (16.5)$$

- Primal exponential cone:

$$\mathcal{K}_t = \{x \in \mathbb{R}^3 : x_1 \geq x_2 \exp(x_3/x_2), \quad x_1, x_2 \geq 0\}. \quad (16.6)$$

- Primal power cone (with parameter $0 < \alpha < 1$):

$$\mathcal{K}_t = \left\{ x \in \mathbb{R}^{n^t} : x_1^\alpha x_2^{1-\alpha} \geq \sqrt{\sum_{j=3}^{n^t} x_j^2}, \quad x_1, x_2 \geq 0 \right\}. \quad (16.7)$$

- Dual exponential cone:

$$\mathcal{K}_t = \{x \in \mathbb{R}^3 : x_1 \geq -x_3 e^{-1} \exp(x_2/x_3), \quad x_3 \leq 0, x_1 \geq 0\}. \quad (16.8)$$

- Dual power cone (with parameter $0 < \alpha < 1$):

$$\mathcal{K}_t = \left\{ x \in \mathbb{R}^{n^t} : \left(\frac{x_1}{\alpha}\right)^\alpha \left(\frac{x_2}{1-\alpha}\right)^{1-\alpha} \geq \sqrt{\sum_{j=3}^{n^t} x_j^2}, \quad x_1, x_2 \geq 0 \right\}. \quad (16.9)$$

In general, membership in the \mathbb{R} set is not specified. If a variable is not a member of any other cone then it is assumed to be a member of the \mathbb{R} cone.

Next, let us study an example. Assume that the power cone

$$x_4^{1/3} x_5^{2/3} \geq |x_8|$$

and the rotated quadratic cone

$$2x_3x_7 \geq x_1^2 + x_0^2, \quad x_3, x_7 \geq 0,$$

should be specified in the MPS file. One CSECTION is required for each cone and they are specified as follows:

*	1	2	3	4	5	6
*234567890123456789012345678901234567890123456789012345678901234567890						
CSECTION	konea	3e-1		PPOW		
x4						
x5						
x8						
CSECTION	koneb	0.0		RQUAD		
x7						
x3						
x1						
x0						

In general, a CSECTION header has the format

CSECTION	[kname1]	[value1]	[ktype]
----------	----------	----------	---------

where the requirements for each field are as follows:

Field	Starting Position	Max Width	Required	Description
[kname1]	15	8	Yes	Name of the cone
[value1]	25	12	No	Cone parameter
[ktype]	40		Yes	Type of the cone.

The possible cone type keys are:

[ktype]	Members	[value1]	Interpretation.
ZERO	≥ 0	unused	Zero cone (16.3).
QUAD	≥ 1	unused	Quadratic cone (16.4).
RQUAD	≥ 2	unused	Rotated quadratic cone (16.5).
PEXP	3	unused	Primal exponential cone (16.6).
PPOW	≥ 2	α	Primal power cone (16.7).
DEXP	3	unused	Dual exponential cone (16.8).
DPOW	≥ 2	α	Dual power cone (16.9).

A record in the CSECTION has the format

[vname1]

where the requirements for each field are

Field	Starting Position	Max Width	required	Description
[vname1]	5	8	Yes	A valid variable name

A variable must occur in at most one CSECTION.

ENDATA

This keyword denotes the end of the MPS file.

16.2.2 Integer Variables

Using special bound keys in the **BOUNDS** section it is possible to specify that some or all of the variables should be integer-constrained i.e. be members of \mathcal{J} . However, an alternative method is available. This method is available only for backward compatibility and we recommend that it is not used. This method requires that markers are placed in the **COLUMNS** section as in the example:

COLUMNS				
x1	obj	-10.0	c1	0.7
x1	c2	0.5	c3	1.0
x1	c4	0.1		
* Start of integer-constrained variables.				
MARK000	'MARKER'		'INTORG'	
x2	obj	-9.0	c1	1.0
x2	c2	0.8333333333	c3	0.66666667
x2	c4	0.25		
x3	obj	1.0	c6	2.0
MARK001	'MARKER'		'INTEND'	
* End of integer-constrained variables.				

Please note that special marker lines are used to indicate the start and the end of the integer variables. Furthermore be aware of the following

- All variables between the markers are assigned a default lower bound of 0 and a default upper bound of 1. **This may not be what is intended.** If it is not intended, the correct bounds should be defined in the **BOUNDS** section of the MPS formatted file.
- **MOSEK** ignores field 1, i.e. MARK0001 and MARK001, however, other optimization systems require them.
- Field 2, i.e. **MARKER**, must be specified including the single quotes. This implies that no row can be assigned the name **MARKER**.
- Field 3 is ignored and should be left blank.
- Field 4, i.e. **INTORG** and **INTEND**, must be specified.
- It is possible to specify several such integer marker sections within the **COLUMNS** section.

16.2.3 General Limitations

- An MPS file should be an ASCII file.

16.2.4 Interpretation of the MPS Format

Several issues related to the MPS format are not well-defined by the industry standard. However, **MOSEK** uses the following interpretation:

- If a matrix element in the **COLUMNS** section is specified multiple times, then the multiple entries are added together.
- If a matrix element in a **QSECTION** section is specified multiple times, then the multiple entries are added together.

16.2.5 The Free MPS Format

MOSEK supports a free format variation of the MPS format. The free format is similar to the MPS file format but less restrictive, e.g. it allows longer names. However, a name must not contain any blanks.

Moreover, by default a line in the MPS file must not contain more than 1024 characters. By modifying the parameter `iparam.read_mps_width` an arbitrary large line width will be accepted.

The free MPS format is default. To change to the strict and other formats use the parameter `iparam.read_mps_format`.

16.3 The OPF Format

The *Optimization Problem Format (OPF)* is an alternative to LP and MPS files for specifying optimization problems. It is row-oriented, inspired by the CPLEX LP format.

Apart from containing objective, constraints, bounds etc. it may contain complete or partial solutions, comments and extra information relevant for solving the problem. It is designed to be easily read and modified by hand and to be forward compatible with possible future extensions.

Intended use

The OPF file format is meant to replace several other files:

- The LP file format: Any problem that can be written as an LP file can be written as an OPF file too; furthermore it naturally accommodates ranged constraints and variables as well as arbitrary characters in names, fixed expressions in the objective, empty constraints, and conic constraints.
- Parameter files: It is possible to specify integer, double and string parameters along with the problem (or in a separate OPF file).
- Solution files: It is possible to store a full or a partial solution in an OPF file and later reload it.

16.3.1 The File Format

The format uses tags to structure data. A simple example with the basic sections may look like this:

```
[comment]
This is a comment. You may write almost anything here...
[/comment]

# This is a single-line comment.

[objective min 'myobj']
x + 3 y + x^2 + 3 y^2 + z + 1
[/objective]

[constraints]
[con 'con01'] 4 <= x + y  [/con]
[/constraints]

[bounds]
[b] -10 <= x,y <= 10  [/b]

[cone quad] x,y,z [/cone]
[/bounds]
```

A scope is opened by a tag of the form `[tag]` and closed by a tag of the form `[/tag]`. An opening tag may accept a list of unnamed and named arguments, for examples:

```
[tag value] tag with one unnamed argument [/tag]
[tag arg=value] tag with one named argument [/tag]
```

Unnamed arguments are identified by their order, while named arguments may appear in any order, but never before an unnamed argument. The **value** can be a quoted, single-quoted or double-quoted text string, i.e.

```
[tag 'value']      single-quoted value [/tag]
[tag arg='value']  single-quoted value [/tag]
[tag "value"]      double-quoted value [/tag]
[tag arg="value"]  double-quoted value [/tag]
```

16.3.2 Sections

The recognized tags are

`[comment]`

A comment section. This can contain *almost* any text: Between single quotes (') or double quotes (") any text may appear. Outside quotes the markup characters ([and]) must be prefixed by backslashes. Both single and double quotes may appear alone or inside a pair of quotes if it is prefixed by a backslash.

`[objective]`

The objective function: This accepts one or two parameters, where the first one (in the above example `min`) is either `min` or `max` (regardless of case) and defines the objective sense, and the second one (above `myobj`), if present, is the objective name. The section may contain linear and quadratic expressions.

If several objectives are specified, all but the last are ignored.

`[constraints]`

This does not directly contain any data, but may contain subsections `con` defining a linear constraint.

`[con]`

Defines a single constraint; if an argument is present (`[con NAME]`) this is used as the name of the constraint, otherwise it is given a null-name. The section contains a constraint definition written as linear and quadratic expressions with a lower bound, an upper bound, with both or with an equality. Examples:

```
[constraints]
[con 'con1'] 0 <= x + y      [/con]
[con 'con2'] 0 >= x + y      [/con]
[con 'con3'] 0 <= x + y <= 10 [/con]
[con 'con4']      x + y = 10 [/con]
[/constraints]
```

Constraint names are unique. If a constraint is specified which has the same name as a previously defined constraint, the new constraint replaces the existing one.

`[bounds]`

This does not directly contain any data, but may contain subsections `b` (linear bounds on variables) and `cone` (cones).

[b]

Bound definition on one or several variables separated by comma (,). An upper or lower bound on a variable replaces any earlier defined bound on that variable. If only one bound (upper or lower) is given only this bound is replaced. This means that upper and lower bounds can be specified separately. So the OPF bound definition:

```
[b] x,y >= -10 [/b]
[b] x,y <= 10  [/b]
```

results in the bound $-10 \leq x, y \leq 10$.

[cone]

Specifies a cone. A cone is defined as a sequence of variables which belong to a single unique cone. The supported cone types are:

- **quad**: a quadratic cone of n variables x_1, \dots, x_n defines a constraint of the form

$$x_1^2 \geq \sum_{i=2}^n x_i^2, \quad x_1 \geq 0.$$

- **rquad**: a rotated quadratic cone of n variables x_1, \dots, x_n defines a constraint of the form

$$2x_1x_2 \geq \sum_{i=3}^n x_i^2, \quad x_1, x_2 \geq 0.$$

- **pexp**: primal exponential cone of 3 variables x_1, x_2, x_3 defines a constraint of the form

$$x_1 \geq x_2 \exp(x_3/x_2), \quad x_1, x_2 \geq 0.$$

- **ppow** with parameter $0 < \alpha < 1$: primal power cone of n variables x_1, \dots, x_n defines a constraint of the form

$$x_1^\alpha x_2^{1-\alpha} \geq \sqrt{\sum_{j=3}^n x_j^2}, \quad x_1, x_2 \geq 0.$$

- **dexp**: dual exponential cone of 3 variables x_1, x_2, x_3 defines a constraint of the form

$$x_1 \geq -x_3 e^{-1} \exp(x_2/x_3), \quad x_3 \leq 0, x_1 \geq 0.$$

- **dpow** with parameter $0 < \alpha < 1$: dual power cone of n variables x_1, \dots, x_n defines a constraint of the form

$$\left(\frac{x_1}{\alpha}\right)^\alpha \left(\frac{x_2}{1-\alpha}\right)^{1-\alpha} \geq \sqrt{\sum_{j=3}^n x_j^2}, \quad x_1, x_2 \geq 0.$$

- **zero**: zero cone of n variables x_1, \dots, x_n defines a constraint of the form

$$x_1 = \dots = x_n = 0$$

A [bounds]-section example:

```
[bounds]
[b] 0 <= x,y <= 10 [/b] # ranged bound
[b] 10 >= x,y >= 0 [/b] # ranged bound
[b] 0 <= x,y <= inf [/b] # using inf
[b] x,y free [/b] # free variables
# Let (x,y,z,w) belong to the cone K
[cone rquad] x,y,z,w [/cone] # rotated quadratic cone
[cone ppow '3e-01' 'a'] x1, x2, x3 [/cone] # power cone with alpha=1/3 and name 'a'
[/bounds]
```

By default all variables are free.

[variables]

This defines an ordering of variables as they should appear in the problem. This is simply a space-separated list of variable names.

[integer]

This contains a space-separated list of variables and defines the constraint that the listed variables must be integer-valued.

[hints]

This may contain only non-essential data; for example estimates of the number of variables, constraints and non-zeros. Placed before all other sections containing data this may reduce the time spent reading the file.

In the `hints` section, any subsection which is not recognized by **MOSEK** is simply ignored. In this section a hint is defined as follows:

```
[hint ITEM] value [/hint]
```

The hints recognized by **MOSEK** are:

- `numvar` (number of variables),
- `numcon` (number of linear/quadratic constraints),
- `numanz` (number of linear non-zeros in constraints),
- `numqnz` (number of quadratic non-zeros in constraints).

[solutions]

This section can contain a set of full or partial solutions to a problem. Each solution must be specified using a `[solution]`-section, i.e.

```
[solutions]
[solution]...[/solution] #solution 1
[solution]...[/solution] #solution 2
#other solutions....
[solution]...[/solution] #solution n
[/solutions]
```

The syntax of a `[solution]`-section is the following:

```
[solution SOLTYPE status=STATUS]...[/solution]
```

where `SOLTYPE` is one of the strings

- `interior`, a non-basic solution,
- `basic`, a basic solution,
- `integer`, an integer solution,

and `STATUS` is one of the strings

- `UNKNOWN`,
- `OPTIMAL`,
- `INTEGER_OPTIMAL`,
- `PRIM_FEAS`,
- `DUAL_FEAS`,
- `PRIM_AND_DUAL_FEAS`,

- NEAR_OPTIMAL,
- NEAR_PRIM_FEAS,
- NEAR_DUAL_FEAS,
- NEAR_PRIM_AND_DUAL_FEAS,
- PRIM_INFEAS_CER,
- DUAL_INFEAS_CER,
- NEAR_PRIM_INFEAS_CER,
- NEAR_DUAL_INFEAS_CER,
- NEAR_INTEGER_OPTIMAL.

Most of these values are irrelevant for input solutions; when constructing a solution for simplex hot-start or an initial solution for a mixed integer problem the safe setting is UNKNOWN.

A [solution]-section contains [con] and [var] sections. Each [con] and [var] section defines solution information for a single variable or constraint, specified as list of KEYWORD/value pairs, in any order, written as

KEYWORD=value

Allowed keywords are as follows:

- **sk**. The status of the item, where the **value** is one of the following strings:
 - LOW, the item is on its lower bound.
 - UPR, the item is on its upper bound.
 - FIX, it is a fixed item.
 - BAS, the item is in the basis.
 - SUPBAS, the item is super basic.
 - UNK, the status is unknown.
 - INF, the item is outside its bounds (infeasible).
- **lv1** Defines the level of the item.
- **s1** Defines the level of the dual variable associated with its lower bound.
- **su** Defines the level of the dual variable associated with its upper bound.
- **sn** Defines the level of the variable associated with its cone.
- **y** Defines the level of the corresponding dual variable (for constraints only).

A [var] section should always contain the items **sk**, **lv1**, **s1** and **su**. Items **s1** and **su** are not required for integer solutions.

A [con] section should always contain **sk**, **lv1**, **s1**, **su** and **y**.

An example of a solution section

```
[solution basic status=UNKNOWN]
[var x0] sk=LOW    lv1=5.0    [/var]
[var x1] sk=UPR    lv1=10.0   [/var]
[var x2] sk=SUPBAS lv1=2.0    s1=1.5 su=0.0 [/var]

[con c0] sk=LOW    lv1=3.0 y=0.0 [/con]
[con c0] sk=UPR    lv1=0.0 y=5.0 [/con]
[/solution]
```

- **[vendor]** This contains solver/vendor specific data. It accepts one argument, which is a vendor ID – for **MOSEK** the ID is simply **mosek** – and the section contains the subsection **parameters** defining solver parameters. When reading a vendor section, any unknown vendor can be safely ignored. This is described later.

Comments using the **#** may appear anywhere in the file. Between the **#** and the following line-break any text may be written, including markup characters.

16.3.3 Numbers

Numbers, when used for parameter values or coefficients, are written in the usual way by the **printf** function. That is, they may be prefixed by a sign (+ or -) and may contain an integer part, decimal part and an exponent. The decimal point is always **.** (a dot). Some examples are

```
1
1.0
.0
1.
1e10
1e+10
1e-10
```

Some *invalid* examples are

```
e10    # invalid, must contain either integer or decimal part
.       # invalid
.e10   # invalid
```

More formally, the following standard regular expression describes numbers as used:

```
[+|-]?([0-9]+[.][0-9]*|.[0-9]+)([eE][+|-]?[0-9]+)?
```

16.3.4 Names

Variable names, constraint names and objective name may contain arbitrary characters, which in some cases must be enclosed by quotes (single or double) that in turn must be preceded by a backslash. Unquoted names must begin with a letter (**a-z** or **A-Z**) and contain only the following characters: the letters **a-z** and **A-Z**, the digits **0-9**, braces (**{** and **}**) and underscore (**_**).

Some examples of legal names:

```
an_unquoted_name
another_name{123}
'single quoted name'
"double quoted name"
"name with \"quote\" in it"
"name with []s in it"
```

16.3.5 Parameters Section

In the **vendor** section solver parameters are defined inside the **parameters** subsection. Each parameter is written as

```
[p PARAMETER_NAME] value [/p]
```

where **PARAMETER_NAME** is replaced by a **MOSEK** parameter name, usually of the form **MSK_IPAR_...**, **MSK_DPAR_...** or **MSK_SPAR_...**, and the **value** is replaced by the value of that parameter; both integer values and named values may be used. Some simple examples are

```
[vendor mosek]
[parameters]
[p MSK_IPAR_OPF_MAX_TERMS_PER_LINE] 10      [/p]
[p MSK_IPAR_OPF_WRITE_PARAMETERS]    MSK_ON [/p]
[p MSK_DPAR_DATA_TOL_BOUND_INF]      1.0e18 [/p]
[/parameters]
[/vendor]
```

16.3.6 Writing OPF Files from MOSEK

To write an OPF file then make sure the file extension is .opf.

Then modify the following parameters to define what the file should contain:

<i>iparam.opf_write_sol_bas</i>	Include basic solution, if defined.
<i>iparam.opf_write_sol_itg</i>	Include integer solution, if defined.
<i>iparam.opf_write_sol_itr</i>	Include interior solution, if defined.
<i>iparam.opf_write_solutions</i>	Include solutions if they are defined. If this is off, no solutions are included.
<i>iparam.opf_write_header</i>	Include a small header with comments.
<i>iparam.opf_write_problem</i>	Include the problem itself — objective, constraints and bounds.
<i>iparam.opf_write_parameters</i>	Include all parameter settings.
<i>iparam.opf_write_hints</i>	Include hints about the size of the problem.

16.3.7 Examples

This section contains a set of small examples written in OPF and describing how to formulate linear, quadratic and conic problems.

Linear Example lo1.opf

Consider the example:

$$\begin{array}{rcll}
\text{maximize} & 3x_0 & + & 1x_1 & + & 5x_2 & + & 1x_3 \\
\text{subject to} & 3x_0 & + & 1x_1 & + & 2x_2 & & = & 30, \\
& 2x_0 & + & 1x_1 & + & 3x_2 & + & 1x_3 & \geq & 15, \\
& & & 2x_1 & & & + & 3x_3 & \leq & 25,
\end{array}$$

having the bounds

$$\begin{array}{rcl}
0 & \leq & x_0 \leq \infty, \\
0 & \leq & x_1 \leq 10, \\
0 & \leq & x_2 \leq \infty, \\
0 & \leq & x_3 \leq \infty.
\end{array}$$

In the OPF format the example is displayed as shown in [Listing 16.1](#).

Listing 16.1: Example of an OPF file for a linear problem.

```
[comment]
  The lo1 example in OPF format
[/comment]

[hints]
  [hint NUMVAR] 4 [/hint]
  [hint NUMCON] 3 [/hint]
  [hint NUMANZ] 9 [/hint]
[/hints]
```

(continues on next page)

```

[variables disallow_new_variables]
  x1 x2 x3 x4
[/variables]

[objective maximize 'obj']
  3 x1 + x2 + 5 x3 + x4
[/objective]

[constraints]
  [con 'c1'] 3 x1 +   x2 + 2 x3           = 30 [/con]
  [con 'c2'] 2 x1 +   x2 + 3 x3 +   x4 >= 15 [/con]
  [con 'c3']      2 x2           + 3 x4 <= 25 [/con]
[/constraints]

[bounds]
  [b] 0 <= * [/b]
  [b] 0 <= x2 <= 10 [/b]
[/bounds]

```

Quadratic Example qo1.opf

An example of a quadratic optimization problem is

$$\begin{aligned}
 &\text{minimize} && x_1^2 + 0.1x_2^2 + x_3^2 - x_1x_3 - x_2 \\
 &\text{subject to} && 1 \leq x_1 + x_2 + x_3, \\
 &&& x \geq 0.
 \end{aligned}$$

This can be formulated in `opf` as shown below.

Listing 16.2: Example of an OPF file for a quadratic problem.

```

[comment]
  The qo1 example in OPF format
[/comment]

[hints]
  [hint NUMVAR] 3 [/hint]
  [hint NUMCON] 1 [/hint]
  [hint NUMANZ] 3 [/hint]
  [hint NUMQNZ] 4 [/hint]
[/hints]

[variables disallow_new_variables]
  x1 x2 x3
[/variables]

[objective minimize 'obj']
  # The quadratic terms are often written with a factor of 1/2 as here,
  # but this is not required.

  - x2 + 0.5 ( 2.0 x1 ^ 2 - 2.0 x3 * x1 + 0.2 x2 ^ 2 + 2.0 x3 ^ 2 )
[/objective]

[constraints]
  [con 'c1'] 1.0 <= x1 + x2 + x3 [/con]
[/constraints]

```

(continues on next page)

```
[bounds]
[b] 0 <= * [/b]
[/bounds]
```

Conic Quadratic Example `cqo1.opf`

Consider the example:

$$\begin{aligned}
&\text{minimize} && x_3 + x_4 + x_5 \\
&\text{subject to} && x_0 + x_1 + 2x_2 = 1, \\
& && x_0, x_1, x_2 \geq 0, \\
& && x_3 \geq \sqrt{x_0^2 + x_1^2}, \\
& && 2x_4x_5 \geq x_2^2.
\end{aligned}$$

Please note that the type of the cones is defined by the parameter to `[cone ...]`; the content of the `cone`-section is the names of variables that belong to the cone. The resulting OPF file is in [Listing 16.3](#).

Listing 16.3: Example of an OPF file for a conic quadratic problem.

```
[comment]
  The cqo1 example in OPF format.
[/comment]

[hints]
[ hint NUMVAR] 6 [/hint]
[ hint NUMCON] 1 [/hint]
[ hint NUMANZ] 3 [/hint]
[/hints]

[variables disallow_new_variables]
  x1 x2 x3 x4 x5 x6
[/variables]

[objective minimize 'obj']
  x4 + x5 + x6
[/objective]

[constraints]
[con 'c1']  x1 + x2 + 2e+00 x3 = 1e+00 [/con]
[/constraints]

[bounds]
# We let all variables default to the positive orthant
[b] 0 <= * [/b]

# ...and change those that differ from the default
[b] x4,x5,x6 free [/b]

# Define quadratic cone: x4 >= sqrt( x1^2 + x2^2 )
[cone quad 'k1'] x4, x1, x2 [/cone]

# Define rotated quadratic cone: 2 x5 x6 >= x3^2
[cone rquad 'k2'] x5, x6, x3 [/cone]
[/bounds]
```

Mixed Integer Example milo1.opf

Consider the mixed integer problem:

$$\begin{array}{llll} \text{maximize} & x_0 + 0.64x_1 & & \\ \text{subject to} & 50x_0 + 31x_1 & \leq & 250, \\ & 3x_0 - 2x_1 & \geq & -4, \\ & x_0, x_1 \geq 0 & & \text{and integer} \end{array}$$

This can be implemented in OPF with the file in [Listing 16.4](#).

Listing 16.4: Example of an OPF file for a mixed-integer linear problem.

```
[comment]
  The milo1 example in OPF format
[/comment]

[hints]
  [hint NUMVAR] 2 [/hint]
  [hint NUMCON] 2 [/hint]
  [hint NUMANZ] 4 [/hint]
[/hints]

[variables disallow_new_variables]
  x1 x2
[/variables]

[objective maximize 'obj']
  x1 + 6.4e-1 x2
[/objective]

[constraints]
  [con 'c1'] 5e+1 x1 + 3.1e+1 x2 <= 2.5e+2 [/con]
  [con 'c2'] -4 <= 3 x1 - 2 x2 [/con]
[/constraints]

[bounds]
  [b] 0 <= * [/b]
[/bounds]

[integer]
  x1 x2
[/integer]
```

16.4 The CBF Format

This document constitutes the technical reference manual of the *Conic Benchmark Format* with file extension: `.cbf` or `.CBF`. It unifies linear, second-order cone (also known as conic quadratic) and semidefinite optimization with mixed-integer variables. The format has been designed with benchmark libraries in mind, and therefore focuses on compact and easily parsable representations. The problem structure is separated from the problem data, and the format moreover facilitates benchmarking of hotstart capability through sequences of changes.

16.4.1 How Instances Are Specified

This section defines the spectrum of conic optimization problems that can be formulated in terms of the keywords of the CBF format.

In the CBF format, conic optimization problems are considered in the following form:

$$\begin{aligned} \min / \max \quad & g^{obj} \\ \text{s.t.} \quad & g_i \in \mathcal{K}_i, \quad i \in \mathcal{I}, \\ & G_i \in \mathcal{K}_i, \quad i \in \mathcal{I}^{PSD}, \\ & x_j \in \mathcal{K}_j, \quad j \in \mathcal{J}, \\ & \bar{X}_j \in \mathcal{K}_j, \quad j \in \mathcal{J}^{PSD}. \end{aligned} \tag{16.10}$$

- **Variables** are either scalar variables, x_j for $j \in \mathcal{J}$, or variables, \bar{X}_j for $j \in \mathcal{J}^{PSD}$. Scalar variables can also be declared as integer.
- **Constraints** are affine expressions of the variables, either scalar-valued g_i for $i \in \mathcal{I}$, or matrix-valued G_i for $i \in \mathcal{I}^{PSD}$

$$\begin{aligned} g_i &= \sum_{j \in \mathcal{J}^{PSD}} \langle F_{ij}, X_j \rangle + \sum_{j \in \mathcal{J}} a_{ij} x_j + b_i, \\ G_i &= \sum_{j \in \mathcal{J}} x_j H_{ij} + D_i. \end{aligned}$$

- The **objective function** is a scalar-valued affine expression of the variables, either to be minimized or maximized. We refer to this expression as g^{obj}

$$g^{obj} = \sum_{j \in \mathcal{J}^{PSD}} \langle F_j^{obj}, X_j \rangle + \sum_{j \in \mathcal{J}} a_j^{obj} x_j + b^{obj}.$$

CBF format can represent the following cones \mathcal{K} :

- **Free domain** - A cone in the linear family defined by

$$\{x \in \mathbb{R}^n\}, \text{ for } n \geq 1.$$

- **Positive orthant** - A cone in the linear family defined by

$$\{x \in \mathbb{R}^n \mid x_j \geq 0 \text{ for } j = 1, \dots, n\}, \text{ for } n \geq 1.$$

- **Negative orthant** - A cone in the linear family defined by

$$\{x \in \mathbb{R}^n \mid x_j \leq 0 \text{ for } j = 1, \dots, n\}, \text{ for } n \geq 1.$$

- **Fixpoint zero** - A cone in the linear family defined by

$$\{x \in \mathbb{R}^n \mid x_j = 0 \text{ for } j = 1, \dots, n\}, \text{ for } n \geq 1.$$

- **Quadratic cone** - A cone in the second-order cone family defined by

$$\left\{ \begin{pmatrix} p \\ x \end{pmatrix} \in \mathbb{R} \times \mathbb{R}^{n-1}, p^2 \geq x^T x, p \geq 0 \right\}, \text{ for } n \geq 2.$$

- **Rotated quadratic cone** - A cone in the second-order cone family defined by

$$\left\{ \begin{pmatrix} p \\ q \\ x \end{pmatrix} \in \mathbb{R} \times \mathbb{R} \times \mathbb{R}^{n-2}, 2pq \geq x^T x, p \geq 0, q \geq 0 \right\}, \text{ for } n \geq 3.$$

16.4.2 The Structure of CBF Files

This section defines how information is written in the CBF format, without being specific about the type of information being communicated.

All information items belong to exactly one of the three groups of information. These information groups, and the order they must appear in, are:

1. File format.
2. Problem structure.
3. Problem data.

The first group, file format, provides information on how to interpret the file. The second group, problem structure, provides the information needed to deduce the type and size of the problem instance. Finally, the third group, problem data, specifies the coefficients and constants of the problem instance.

Information items

The format is composed as a list of information items. The first line of an information item is the **KEYWORD**, revealing the type of information provided. The second line - of some keywords only - is the **HEADER**, typically revealing the size of information that follows. The remaining lines are the **BODY** holding the actual information to be specified.

KEYWORD
BODY

KEYWORD
HEADER
BODY

The **KEYWORD** determines how each line in the **HEADER** and **BODY** is structured. Moreover, the number of lines in the **BODY** follows either from the **KEYWORD**, the **HEADER**, or from another information item required to precede it.

Embedded hotstart-sequences

A sequence of problem instances, based on the same problem structure, is within a single file. This is facilitated via the **CHANGE** within the problem data information group, as a separator between the information items of each instance. The information items following a **CHANGE** keyword are appending to, or changing (e.g., setting coefficients back to their default value of zero), the problem data of the preceding instance.

The sequence is intended for benchmarking of hotstart capability, where the solvers can reuse their internal state and solution (subject to the achieved accuracy) as warmpoint for the succeeding instance. Whenever this feature is unsupported or undesired, the keyword **CHANGE** should be interpreted as the end of file.

File encoding and line width restrictions

The format is based on the US-ASCII printable character set with two extensions as listed below. Note, by definition, that none of these extensions can be misinterpreted as printable US-ASCII characters:

- A line feed marks the end of a line, carriage returns are ignored.
- Comment-lines may contain unicode characters in UTF-8 encoding.

The line width is restricted to 512 bytes, with 3 bytes reserved for the potential carriage return, line feed and null-terminator.

Integers and floating point numbers must follow the ISO C decimal string representation in the standard C locale. The format does not impose restrictions on the magnitude of, or number of significant digits in numeric data, but the use of 64-bit integers and 64-bit IEEE 754 floating point numbers should be sufficient to avoid loss of precision.

Comment-line and whitespace rules

The format allows single-line comments respecting the following rule:

- Lines having first byte equal to '#' (US-ASCII 35) are comments, and should be ignored. Comments are only allowed between information items.

Given that a line is not a comment-line, whitespace characters should be handled according to the following rules:

- Leading and trailing whitespace characters should be ignored.
 - The separator between multiple pieces of information on one line, is either one or more whitespace characters.
- Lines containing only whitespace characters are empty, and should be ignored. Empty lines are only allowed between information items.

16.4.3 Problem Specification

The problem structure

The problem structure defines the objective sense, whether it is minimization and maximization. It also defines the index sets, \mathcal{J} , \mathcal{J}^{PSD} , \mathcal{I} and \mathcal{I}^{PSD} , which are all numbered from zero, $\{0, 1, \dots\}$, and empty until explicitly constructed.

- **Scalar variables** are constructed in vectors restricted to a conic domain, such as $(x_0, x_1) \in \mathbb{R}_+^2$, $(x_2, x_3, x_4) \in \mathcal{Q}^3$, etc. In terms of the Cartesian product, this generalizes to

$$x \in \mathcal{K}_1^{n_1} \times \mathcal{K}_2^{n_2} \times \dots \times \mathcal{K}_k^{n_k}$$

which in the CBF format becomes:

```
VAR
n k
K1 n1
K2 n2
...
Kk nk
```

where $\sum_i n_i = n$ is the total number of scalar variables. The list of supported cones is found in Table 16.3. Integrality of scalar variables can be specified afterwards.

- **PSD variables** are constructed one-by-one. That is, $X_j \succeq \mathbf{0}^{n_j \times n_j}$ for $j \in \mathcal{J}^{PSD}$, constructs a matrix-valued variable of size $n_j \times n_j$ restricted to be symmetric positive semidefinite. In the CBF format, this list of constructions becomes:

```

PSDVAR
N
n1
n2
...
nN

```

where N is the total number of PSD variables.

- **Scalar constraints** are constructed in vectors restricted to a conic domain, such as $(g_0, g_1) \in \mathbb{R}_+^2$, $(g_2, g_3, g_4) \in \mathcal{Q}^3$, etc. In terms of the Cartesian product, this generalizes to

$$g \in \mathcal{K}_1^{m_1} \times \mathcal{K}_2^{m_2} \times \dots \times \mathcal{K}_k^{m_k}$$

which in the CBF format becomes:

```

CON
m k
K1 m1
K2 m2
..
Kk mk

```

where $\sum_i m_i = m$ is the total number of scalar constraints. The list of supported cones is found in [Table 16.3](#).

- **PSD constraints** are constructed one-by-one. That is, $G_i \succeq \mathbf{0}^{m_i \times m_i}$ for $i \in \mathcal{I}^{PSD}$, constructs a matrix-valued affine expressions of size $m_i \times m_i$ restricted to be symmetric positive semidefinite. In the CBF format, this list of constructions becomes

```

PSDCON
M
m1
m2
..
mM

```

where M is the total number of PSD constraints.

With the objective sense, variables (with integer indications) and constraints, the definitions of the many affine expressions follow in problem data.

Problem data

The problem data defines the coefficients and constants of the affine expressions of the problem instance. These are considered zero until explicitly defined, implying that instances with no keywords from this information group are, in fact, valid. Duplicating or conflicting information is a failure to comply with the standard. Consequently, two coefficients written to the same position in a matrix (or to transposed positions in a symmetric matrix) is an error.

The affine expressions of the objective, g^{obj} , of the scalar constraints, g_i , and of the PSD constraints, G_i , are defined separately. The following notation uses the standard trace inner product for matrices, $\langle X, Y \rangle = \sum_{i,j} X_{ij} Y_{ij}$.

- The affine expression of the objective is defined as

$$g^{obj} = \sum_{j \in \mathcal{J}^{PSD}} \langle F_j^{obj}, X_j \rangle + \sum_{j \in \mathcal{J}} a_j^{obj} x_j + b^{obj},$$

in terms of the symmetric matrices, F_j^{obj} , and scalars, a_j^{obj} and b^{obj} .

- The affine expressions of the scalar constraints are defined, for $i \in \mathcal{I}$, as

$$g_i = \sum_{j \in \mathcal{J}^{PSD}} \langle F_{ij}, X_j \rangle + \sum_{j \in \mathcal{J}} a_{ij} x_j + b_i,$$

in terms of the symmetric matrices, F_{ij} , and scalars, a_{ij} and b_i .

- The affine expressions of the PSD constraints are defined, for $i \in \mathcal{I}^{PSD}$, as

$$G_i = \sum_{j \in \mathcal{J}} x_j H_{ij} + D_i,$$

in terms of the symmetric matrices, H_{ij} and D_i .

List of cones

The format uses an explicit syntax for symmetric positive semidefinite cones as shown above. For scalar variables and constraints, constructed in vectors, the supported conic domains and their minimum sizes are given as follows.

Table 16.3: Cones available in the CBF format

Name	CBF keyword	Cone family
Free domain	F	linear
Positive orthant	L+	linear
Negative orthant	L-	linear
Fixpoint zero	L=	linear
Quadratic cone	Q	second-order
Rotated quadratic cone	QR	second-order

16.4.4 File Format Keywords

VER

Description: The version of the Conic Benchmark Format used to write the file.

HEADER: None

BODY: One line formatted as:

INT

This is the version number.

Must appear exactly once in a file, as the first keyword.

OBJSENSE

Description: Define the objective sense.

HEADER: None

BODY: One line formatted as:

STR

having MIN indicates minimize, and MAX indicates maximize. Capital letters are required.

Must appear exactly once in a file.

PSDVAR

Description: Construct the PSD variables.

HEADER: One line formatted as:

INT

This is the number of PSD variables in the problem.

BODY: A list of lines formatted as:

INT

This indicates the number of rows (equal to the number of columns) in the matrix-valued PSD variable. The number of lines should match the number stated in the header.

VAR

Description: Construct the scalar variables.

HEADER: One line formatted as:

INT INT

This is the number of scalar variables, followed by the number of conic domains they are restricted to.

BODY: A list of lines formatted as:

STR INT

This indicates the cone name (see [Table 16.3](#)), and the number of scalar variables restricted to this cone. These numbers should add up to the number of scalar variables stated first in the header. The number of lines should match the second number stated in the header.

INT

Description: Declare integer requirements on a selected subset of scalar variables.

HEADER: one line formatted as:

INT

This is the number of integer scalar variables in the problem.

BODY: a list of lines formatted as:

INT

This indicates the scalar variable index $j \in \mathcal{J}$. The number of lines should match the number stated in the header.

Can only be used after the keyword **VAR**.

PSDCON

Description: Construct the PSD constraints.

HEADER: One line formatted as:

INT

This is the number of PSD constraints in the problem.

BODY: A list of lines formatted as:

INT

This indicates the number of rows (equal to the number of columns) in the matrix-valued affine expression of the PSD constraint. The number of lines should match the number stated in the header.

Can only be used after these keywords: **PSDVAR**, **VAR**.

CON

Description: Construct the scalar constraints.

HEADER: One line formatted as:

INT INT

This is the number of scalar constraints, followed by the number of conic domains they restrict to.

BODY: A list of lines formatted as:

STR INT

This indicates the cone name (see Table 16.3), and the number of affine expressions restricted to this cone. These numbers should add up to the number of scalar constraints stated first in the header. The number of lines should match the second number stated in the header.

Can only be used after these keywords: PSDVAR, VAR

OBJFCOORD

Description: Input sparse coordinates (quadruplets) to define the symmetric matrices F_j^{obj} , as used in the objective.

HEADER: One line formatted as:

INT

This is the number of coordinates to be specified.

BODY: A list of lines formatted as:

INT INT INT REAL

This indicates the PSD variable index $j \in \mathcal{J}^{PSD}$, the row index, the column index and the coefficient value. The number of lines should match the number stated in the header.

OBJACOORD

Description: Input sparse coordinates (pairs) to define the scalars, a_j^{obj} , as used in the objective.

HEADER: One line formatted as:

INT

This is the number of coordinates to be specified.

BODY: A list of lines formatted as:

INT REAL

This indicates the scalar variable index $j \in \mathcal{J}$ and the coefficient value. The number of lines should match the number stated in the header.

OBJBCOORD

Description: Input the scalar, b^{obj} , as used in the objective.

HEADER: None.

BODY: One line formatted as:

REAL

This indicates the coefficient value.

FCOORD

Description: Input sparse coordinates (quintuplets) to define the symmetric matrices, F_{ij} , as used in the scalar constraints.

HEADER: One line formatted as:

INT

This is the number of coordinates to be specified.

BODY: A list of lines formatted as:

INT INT INT INT REAL

This indicates the scalar constraint index $i \in \mathcal{I}$, the PSD variable index $j \in \mathcal{J}^{PSD}$, the row index, the column index and the coefficient value. The number of lines should match the number stated in the header.

ACOORD

Description: Input sparse coordinates (triplets) to define the scalars, a_{ij} , as used in the scalar constraints.

HEADER: One line formatted as:

INT

This is the number of coordinates to be specified.

BODY: A list of lines formatted as:

INT INT REAL

This indicates the scalar constraint index $i \in \mathcal{I}$, the scalar variable index $j \in \mathcal{J}$ and the coefficient value. The number of lines should match the number stated in the header.

BCOORD

Description: Input sparse coordinates (pairs) to define the scalars, b_i , as used in the scalar constraints.

HEADER: One line formatted as:

INT

This is the number of coordinates to be specified.

BODY: A list of lines formatted as:

INT REAL

This indicates the scalar constraint index $i \in \mathcal{I}$ and the coefficient value. The number of lines should match the number stated in the header.

HCOORD

Description: Input sparse coordinates (quintuplets) to define the symmetric matrices, H_{ij} , as used in the PSD constraints.

HEADER: One line formatted as:

INT

This is the number of coordinates to be specified.

BODY: A list of lines formatted as

INT INT INT INT REAL

This indicates the PSD constraint index $i \in \mathcal{I}^{PSD}$, the scalar variable index $j \in \mathcal{J}$, the row index, the column index and the coefficient value. The number of lines should match the number stated in the header.

DCOORD

Description: Input sparse coordinates (quadruplets) to define the symmetric matrices, D_i , as used in the PSD constraints.

HEADER: One line formatted as

INT

This is the number of coordinates to be specified.

BODY: A list of lines formatted as:

INT INT INT REAL

This indicates the PSD constraint index $i \in \mathcal{I}^{PSD}$, the row index, the column index and the coefficient value. The number of lines should match the number stated in the header.

CHANGE

Start of a new instance specification based on changes to the previous. Can be interpreted as the end of file when the hotstart-sequence is unsupported or undesired.

BODY: None

Header: None

16.4.5 CBF Format Examples

Minimal Working Example

The conic optimization problem (16.11), has three variables in a quadratic cone - first one is integer - and an affine expression in domain 0 (equality constraint).

$$\begin{aligned} & \text{minimize} && 5.1 x_0 \\ & \text{subject to} && 6.2 x_1 + 7.3 x_2 - 8.4 \in \{0\} \\ & && x \in \mathcal{Q}^3, x_0 \in \mathbb{Z}. \end{aligned} \tag{16.11}$$

Its formulation in the Conic Benchmark Format begins with the version of the CBF format used, to safeguard against later revisions.

VER
1

Next follows the problem structure, consisting of the objective sense, the number and domain of variables, the indices of integer variables, and the number and domain of scalar-valued affine expressions (i.e., the equality constraint).

OBJSENSE
MIN
VAR
3 1
Q 3
INT
1
0
CON
1 1
L= 1

Finally follows the problem data, consisting of the coefficients of the objective, the coefficients of the constraints, and the constant terms of the constraints. All data is specified on a sparse coordinate form.

```
OBJCOORD
```

```
1
0 5.1
```

```
ACCOORD
```

```
2
0 1 6.2
0 2 7.3
```

```
BCCOORD
```

```
1
0 -8.4
```

This concludes the example.

Mixing Linear, Second-order and Semidefinite Cones

The conic optimization problem (16.12), has a semidefinite cone, a quadratic cone over unordered subindices, and two equality constraints.

$$\begin{aligned} & \text{minimize} && \left\langle \begin{bmatrix} 2 & 1 & 0 \\ 1 & 2 & 1 \\ 0 & 1 & 2 \end{bmatrix}, X_1 \right\rangle + x_1 \\ & \text{subject to} && \left\langle \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}, X_1 \right\rangle + x_1 &= 1.0, \\ & && \left\langle \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}, X_1 \right\rangle + x_0 + x_2 &= 0.5, \\ & && x_1 \geq \sqrt{x_0^2 + x_2^2}, \\ & && X_1 \succeq \mathbf{0}. \end{aligned} \tag{16.12}$$

The equality constraints are easily rewritten to the conic form, $(g_0, g_1) \in \{0\}^2$, by moving constants such that the right-hand-side becomes zero. The quadratic cone does not fit under the `VAR` keyword in this variable permutation. Instead, it takes a scalar constraint $(g_2, g_3, g_4) = (x_1, x_0, x_2) \in \mathcal{Q}^3$, with scalar variables constructed as $(x_0, x_1, x_2) \in \mathbb{R}^3$. Its formulation in the CBF format is reported in the following list

```
# File written using this version of the Conic Benchmark Format:
#   | Version 1.
VER
1

# The sense of the objective is:
#   | Minimize.
OBJSENSE
MIN

# One PSD variable of this size:
#   | Three times three.
PSDVAR
1
3

# Three scalar variables in this one conic domain:
#   | Three are free.
VAR
3 1
```

(continues on next page)

```

F 3

# Five scalar constraints with affine expressions in two conic domains:
#   | Two are fixed to zero.
#   | Three are in conic quadratic domain.
CON
5 2
L= 2
Q 3

# Five coordinates in  $F^{\{obj\}}_j$  coefficients:
#   |  $F^{\{obj\}}[0][0,0] = 2.0$ 
#   |  $F^{\{obj\}}[0][1,0] = 1.0$ 
#   | and more...
OBJFCOORD
5
0 0 0 2.0
0 1 0 1.0
0 1 1 2.0
0 2 1 1.0
0 2 2 2.0

# One coordinate in  $a^{\{obj\}}_j$  coefficients:
#   |  $a^{\{obj\}}[1] = 1.0$ 
OBJACOORD
1
1 1.0

# Nine coordinates in  $F_{ij}$  coefficients:
#   |  $F[0,0][0,0] = 1.0$ 
#   |  $F[0,0][1,1] = 1.0$ 
#   | and more...
FCOORD
9
0 0 0 0 1.0
0 0 1 1 1.0
0 0 2 2 1.0
1 0 0 0 1.0
1 0 1 0 1.0
1 0 2 0 1.0
1 0 1 1 1.0
1 0 2 1 1.0
1 0 2 2 1.0

# Six coordinates in  $a_{ij}$  coefficients:
#   |  $a[0,1] = 1.0$ 
#   |  $a[1,0] = 1.0$ 
#   | and more...
ACOORD
6
0 1 1.0
1 0 1.0
1 2 1.0
2 1 1.0
3 0 1.0
4 2 1.0

```

(continues on next page)

(continued from previous page)

```
# Two coordinates in b_i coefficients:
#      | b[0] = -1.0
#      | b[1] = -0.5
BCOORD
2
0 -1.0
1 -0.5
```

Mixing Semidefinite Variables and Linear Matrix Inequalities

The standard forms in semidefinite optimization are usually based either on semidefinite variables or linear matrix inequalities. In the CBF format, both forms are supported and can even be mixed as shown in.

$$\begin{aligned} \text{minimize} \quad & \left\langle \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}, X_1 \right\rangle + x_1 + x_2 + 1 \\ \text{subject to} \quad & \left\langle \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}, X_1 \right\rangle - x_1 - x_2 \geq 0.0, \\ & x_1 \begin{bmatrix} 0 & 1 \\ 1 & 3 \end{bmatrix} + x_2 \begin{bmatrix} 3 & 1 \\ 1 & 0 \end{bmatrix} - \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \preceq \mathbf{0}, \\ & X_1 \succeq \mathbf{0}. \end{aligned} \tag{16.13}$$

Its formulation in the CBF format is written in what follows

```
# File written using this version of the Conic Benchmark Format:
#      | Version 1.
VER
1

# The sense of the objective is:
#      | Minimize.
OBJSENSE
MIN

# One PSD variable of this size:
#      | Two times two.
PSDVAR
1
2

# Two scalar variables in this one conic domain:
#      | Two are free.
VAR
2 1
F 2

# One PSD constraint of this size:
#      | Two times two.
PSDCON
1
2

# One scalar constraint with an affine expression in this one conic domain:
#      | One is greater than or equal to zero.
CON
1 1
```

(continues on next page)

```

L+ 1

# Two coordinates in  $F^{\{obj\}}_j$  coefficients:
#   |  $F^{\{obj\}}[0][0,0] = 1.0$ 
#   |  $F^{\{obj\}}[0][1,1] = 1.0$ 
OBJFCOORD
2
0 0 0 1.0
0 1 1 1.0

# Two coordinates in  $a^{\{obj\}}_j$  coefficients:
#   |  $a^{\{obj\}}[0] = 1.0$ 
#   |  $a^{\{obj\}}[1] = 1.0$ 
OBJACOORD
2
0 1.0
1 1.0

# One coordinate in  $b^{\{obj\}}$  coefficient:
#   |  $b^{\{obj\}} = 1.0$ 
OBJBCOORD
1.0

# One coordinate in  $F_{ij}$  coefficients:
#   |  $F[0,0][1,0] = 1.0$ 
FCOORD
1
0 0 1 0 1.0

# Two coordinates in  $a_{ij}$  coefficients:
#   |  $a[0,0] = -1.0$ 
#   |  $a[0,1] = -1.0$ 
ACCOORD
2
0 0 -1.0
0 1 -1.0

# Four coordinates in  $H_{ij}$  coefficients:
#   |  $H[0,0][1,0] = 1.0$ 
#   |  $H[0,0][1,1] = 3.0$ 
#   | and more...
HCOORD
4
0 0 1 0 1.0
0 0 1 1 3.0
0 1 0 0 3.0
0 1 1 0 1.0

# Two coordinates in  $D_i$  coefficients:
#   |  $D[0][0,0] = -1.0$ 
#   |  $D[0][1,1] = -1.0$ 
DCOORD
2
0 0 0 -1.0
0 1 1 -1.0

```

Optimization Over a Sequence of Objectives

The linear optimization problem (16.14), is defined for a sequence of objectives such that hotstarting from one to the next might be advantages.

$$\begin{aligned} & \text{maximize}_k && g_k^{obj} \\ & \text{subject to} && 50x_0 + 31 \leq 250, \\ & && 3x_0 - 2x_1 \geq -4, \\ & && x \in \mathbb{R}_+^2, \end{aligned} \tag{16.14}$$

given,

1. $g_0^{obj} = x_0 + 0.64x_1$.
2. $g_1^{obj} = 1.11x_0 + 0.76x_1$.
3. $g_2^{obj} = 1.11x_0 + 0.85x_1$.

Its formulation in the CBF format is reported in Listing 16.5.

Listing 16.5: Problem (16.14) in CBF format.

```
# File written using this version of the Conic Benchmark Format:
#   | Version 1.
VER
1

# The sense of the objective is:
#   | Maximize.
OBJSENSE
MAX

# Two scalar variables in this one conic domain:
#   | Two are nonnegative.
VAR
2 1
L+ 2

# Two scalar constraints with affine expressions in these two conic domains:
#   | One is in the nonpositive domain.
#   | One is in the nonnegative domain.
CON
2 2
L- 1
L+ 1

# Two coordinates in a^{obj}_j coefficients:
#   | a^{obj}[0] = 1.0
#   | a^{obj}[1] = 0.64
OBJACoord
2
0 1.0
1 0.64

# Four coordinates in a_ij coefficients:
#   | a[0,0] = 50.0
#   | a[1,0] = 3.0
#   | and more...
ACoord
4
```

(continues on next page)

```

0 0 50.0
1 0 3.0
0 1 31.0
1 1 -2.0

# Two coordinates in b_i coefficients:
#     | b[0] = -250.0
#     | b[1] = 4.0
BCOORD
2
0 -250.0
1 4.0

# New problem instance defined in terms of changes.
CHANGE

# Two coordinate changes in a^{obj}_j coefficients. Now it is:
#     | a^{obj}[0] = 1.11
#     | a^{obj}[1] = 0.76
OBJACoord
2
0 1.11
1 0.76

# New problem instance defined in terms of changes.
CHANGE

# One coordinate change in a^{obj}_j coefficients. Now it is:
#     | a^{obj}[0] = 1.11
#     | a^{obj}[1] = 0.85
OBJACoord
1
1 0.85

```

16.5 The PTF Format

The PTF format is a new human-readable, natural text format. Its features and structure are similar to the *OPF* format, with the difference that the PTF format **does** support semidefinite terms.

16.5.1 The overall format

The format is indentation based, where each section is started by a head line and followed by a section body with deeper indentation than the head line. For example:

```

Header line
  Body line 1
  Body line 1
  Body line 1

```

Section can also be nested:

```

Header line A
  Body line in A
  Header line A.1
    Body line in A.1

```

(continues on next page)

```

    Body line in A.1
Body line in A

```

The indentation of blank lines is ignored, so a subsection can contain a blank line with no indentation. The character # defines a line comment and anything between the # character and the end of the line is ignored.

In a PTF file, the first section must be a **Task** section. The order of the remaining section is arbitrary, and sections may occur multiple times or not at all.

MOSEK will ignore any top-level section it does not recognize.

Names

In the description of the format we use following definitions for name strings:

```

NAME: PLAIN_NAME | QUOTED_NAME
PLAIN_NAME: [a-zA-Z_] [a-zA-Z0-9_-.!|]
QUOTED_NAME: '"' ( [^'\\r\n] | "\\" ( [\\rn] | "x" [0-9a-fA-F] [0-9a-fA-F] ) ) * '"'

```

Expressions

An expression is a sum of terms. A term is either a linear term (a coefficient and a variable name, where the coefficient can be left out if it is 1.0), or a matrix inner product.

An expression:

```

EXPR: EMPTY | [+]? TERM ( [+]? TERM ) *
TERM: LINEAR_TERM | MATRIX_TERM

```

A linear term

```

LINEAR_TERM: FLOAT? NAME

```

A matrix term

```

MATRIX_TERM: "<" FLOAT? NAME ( [+]? FLOAT? NAME ) * ";" NAME ">"

```

Here the right-hand name is the name of a (semidefinite) matrix variable, and the left-hand side is a sum of symmetric matrixes. The actual matrixes are defined in a separate section.

Expressions can span multiple lines by giving subsequent lines a deeper indentation.

For example following two section are equivalent:

```

# Everything on one line:
x1 + x2 + x3 + x4

# Split into multiple lines:
x1
+ x2
+ x3
+ x4

```

16.5.2 Task section

The first section of the file must be a **Task**. The text in this section is not used and may contain comments, or meta-information from the writer or about the content.

Format:

```
Task NAME
  Anything goes here...
```

NAME is a the task name.

16.5.3 Objective section

The **Objective** section defines the objective name, sense and function. The format:

```
"Objective" NAME?
  ( "Minimize" | "Maximize" ) EXPR
```

For example:

```
Objective 'obj'
  Minimize x1 + 0.2 x2 + < M1 ; X1 >
```

16.5.4 Constraints section

The constraints section defines a series of constraints. A constraint defines a term $A \cdot x + b \in K$. For linear constraints **A** is just one row, while for conic constraints it can be multiple rows. If a constraint spans multiple rows these can either be written inline separated by semi-colons, or each expression in a separate sub-section.

Simple linear constraints:

```
"Constraints"
NAME? "[" [-+] (FLOAT | "Inf") (";" [-+] (FLOAT | "Inf"))? "]" EXPR
```

If the brackets contain two values, they are used as upper and lower bounds. If they contain one value the constraint is an equality.

For example:

```
Constraints
'c1' [0;10] x1 + x2 + x3
[0] x1 + x2 + x3
```

Constraint blocks put the expression either in a subsection or inline. The cone type (domain) is written in the brackets, and **MOSEK** currently supports following types:

- SOC(N) Second order cone of dimension N
- RSOC(N) Rotated second order cone of dimension N
- PSD(N) Symmetric positive semidefinite cone of dimension N. This contains $N*(N+1)/2$ elements.
- PEXP Primal exponential cone of dimension 3
- DEXP Dual exponential cone of dimension 3
- PPOW(N,P) Primal power cone of dimension N with parameter P
- DPOW(N,P) Dual power cone of dimension N with parameter P
- ZERO(N) The zero-cone of dimension N.

```
"Constraints"
NAME? "[" DOMAIN "]" EXPR_LIST
```

For example:

```
Constraints
'K1' [SOC(3)] x1 + x2 ; x2 + x3 ; x3 + x1
'K2' [RSOC(3)]
    x1 + x2
    x2 + x3
    x3 + x1
```

16.5.5 Variables section

Any variable used in an expression must be defined in a variable section. The variable section defines each variable domain.

```
"Variables"
NAME "[" [-+] (FLOAT | "Inf") (";" [-+] (FLOAT | "Inf"))? "]"
NAME "[" DOMAIN "]" NAMES

For example, a linear variable
```

```
Variables
x1 [0;Inf]
```

As with constraints, members of a conic domain can be listed either inline or in a subsection:

```
Variables
k1 [SOC(3)] x1 ; x2 ; x3
k2 [RSOC(3)]
    x1
    x2
    x3
```

16.5.6 Integer section

This section contains a list of variables that are integral. For example:

```
Integer
x1 x2 x3
```

16.5.7 SymmetricMatrixes section

This section defines the symmetric matrixes used for matrix coefficients in matrix inner product terms. The section lists named matrixes, each with a size and a number of non-zeros. Only non-zeros in the lower triangular part should be defined.

```
"SymmetricMatrixes"
NAME "SYMMAT" "(" INT ")" ( "(" INT "," INT "," FLOAT ")" ) *
...
```

For example:

```
SymmetricMatrixes
M1 SYMMAT(3) (0,0,1.0) (1,1,2.0) (2,1,0.5)
M2 SYMMAT(3)
    (0,0,1.0)
    (1,1,2.0)
    (2,1,0.5)
```

16.5.8 Solutions section

Each subsection defines a solution. A solution defines for each constraint and for each variable exactly one primal value and either one (for conic domains) or two (for linear domains) dual values. The values follow the same logic as in the **MOSEK C API**. A primal and a dual solution status defines the meaning of the values primal and dual (solution, certificate, unknown, etc.)

The format is this:

```
"Solutions"
  "Solution" WHICHSOL
    "ProblemStatus" PROSTA PROSTA?
  "SolutionStatus" SOLSTA SOLSTA?
  "Objective" FLOAT FLOAT
  "Variables"
    # Linear variable status: level, slx, sux
    NAME "[" STATUS "]" FLOAT (FLOAT FLOAT)?
    # Conic variable status: level, snx
    NAME
      "[" STATUS "]" FLOAT FLOAT?
    ...
  "Constraints"
    # Linear variable status: level, slx, sux
    NAME "[" STATUS "]" FLOAT (FLOAT FLOAT)?
    # Conic variable status: level, snx
    NAME
      "[" STATUS "]" FLOAT FLOAT?
    ...
```

Following values for **WHICHSOL** are supported:

- **interior** Interior solution, the result of an interior-point solver.
- **basic** Basic solution, as produced by a simplex solver.
- **integer** Integer solution, the solution to a mixed-integer problem. This does not define a dual solution.

Following values for **PROSTA** are supported:

- **unknown** The problem status is unknown
- **feasible** The problem has been proven feasible
- **infeasible** The problem has been proven infeasible
- **illposed** The problem has been proved to be ill posed
- **infeasible_or_unbounded** The problem is infeasible or unbounded

Following values for **SOLSTA** are supported:

- **unknown** The solution status is unknown
- **feasible** The solution is feasible
- **optimal** The solution is optimal
- **infeas_cert** The solution is a certificate of infeasibility
- **illposed_cert** The solution is a certificate of illposedness

Following values for **STATUS** are supported:

- **unknown** The value is unknown
- **super_basic** The value is super basic

- `at_lower` The value is basic and at its lower bound
- `at_upper` The value is basic and at its upper bound
- `fixed` The value is basic fixed
- `infinite` The value is at infinity

16.6 The Task Format

The Task format is **MOSEK**'s native binary format. It contains a complete image of a **MOSEK** task, i.e.

- Problem data: Linear, conic, semidefinite and quadratic data
- Problem item names: Variable names, constraints names, cone names etc.
- Parameter settings
- Solutions

There are a few things to be aware of:

- Status of a solution read from a file will *always* be unknown.
- Parameter settings in a task file *always override* any parameters set on the command line or in a parameter file.

The format is based on the *TAR* (USTar) file format. This means that the individual pieces of data in a `.task` file can be examined by unpacking it as a *TAR* file. Please note that the inverse may not work: Creating a file using *TAR* will most probably not create a valid **MOSEK** Task file since the order of the entries is important.

16.7 The JSON Format

MOSEK provides the possibility to read/write problems in valid JSON format.

JSON (JavaScript Object Notation) is a lightweight data-interchange format. It is easy for humans to read and write. It is easy for machines to parse and generate. It is based on a subset of the JavaScript Programming Language, Standard ECMA-262 3rd Edition - December 1999. JSON is a text format that is completely language independent but uses conventions that are familiar to programmers of the C-family of languages, including C, C++, C#, Java, JavaScript, Perl, Python, and many others. These properties make JSON an ideal data-interchange language.

The official JSON website <http://www.json.org> provides plenty of information along with the format definition.

MOSEK defines two JSON-like formats:

- `jtask`
- `jsol`

Despite being text-based human-readable formats, *jtask* and *jsol* files will include no indentation and no new-lines, in order to keep the files as compact as possible. We therefore strongly advise to use JSON viewer tools to inspect *jtask* and *jsol* files.

16.7.1 *jtask* format

It stores a problem instance. The *jtask* format contains the same information as a *task format*. Even though a *jtask* file is human-readable, we do not recommend users to create it by hand, but to rely on MOSEK.

16.7.2 *jsol* format

It stores a problem solution. The *jsol* format contains all solutions and information items.

You can write a *jsol* file using `Task.writejsonsol`. You **can not** read a *jsol* file into MOSEK.

16.7.3 A *jtask* example

In Listing 16.6 we present a file in the *jtask* format that corresponds to the sample problem from 1o1.1p. The listing has been formatted for readability.

Listing 16.6: A formatted *jtask* file for the 1o1.1p example.

```
{
  "$schema": "http://mosek.com/json/schema#",
  "Task/INFO": {
    "taskname": "1o1",
    "numvar": 4,
    "numcon": 3,
    "numcone": 0,
    "numbarvar": 0,
    "numanz": 9,
    "numsymmat": 0,
    "mosekver": [
      8,
      0,
      0,
      9
    ]
  },
  "Task/data": {
    "var": {
      "name": [
        "x1",
        "x2",
        "x3",
        "x4"
      ],
      "bk": [
        "lo",
        "ra",
        "lo",
        "lo"
      ],
      "b1": [
        0.0,
        0.0,
        0.0,
        0.0
      ],
      "bu": [
        1e+30,
        1e+1,

```

(continues on next page)

```

        1e+30,
        1e+30
    ],
    "type": [
        "cont",
        "cont",
        "cont",
        "cont"
    ]
},
"con": {
    "name": [
        "c1",
        "c2",
        "c3"
    ],
    "bk": [
        "fx",
        "lo",
        "up"
    ],
    "bl": [
        3e+1,
        1.5e+1,
        -1e+30
    ],
    "bu": [
        3e+1,
        1e+30,
        2.5e+1
    ]
},
"objective": {
    "sense": "max",
    "name": "obj",
    "c": {
        "subj": [
            0,
            1,
            2,
            3
        ],
        "val": [
            3e+0,
            1e+0,
            5e+0,
            1e+0
        ]
    },
    "cfix": 0.0
},
"A": {
    "subi": [
        0,
        0,
        0,

```

```

        1,
        1,
        1,
        1,
        2,
        2
    ],
    "subj": [
        0,
        1,
        2,
        0,
        1,
        2,
        3,
        1,
        3
    ],
    "val": [
        3e+0,
        1e+0,
        2e+0,
        2e+0,
        1e+0,
        3e+0,
        1e+0,
        2e+0,
        3e+0
    ]
    }
},
"Task/parameters": {
    "iparam": {
        "ANA_SOL_BASIS": "ON",
        "ANA_SOL_PRINT_VIOLATED": "OFF",
        "AUTO_SORT_A_BEFORE_OPT": "OFF",
        "AUTO_UPDATE_SOL_INFO": "OFF",
        "BASIS_SOLVE_USE_PLUS_ONE": "OFF",
        "BI_CLEAN_OPTIMIZER": "OPTIMIZER_FREE",
        "BI_IGNORE_MAX_ITER": "OFF",
        "BI_IGNORE_NUM_ERROR": "OFF",
        "BI_MAX_ITERATIONS": 1000000,
        "CACHE_LICENSE": "ON",
        "CHECK_CONVEXITY": "CHECK_CONVEXITY_FULL",
        "COMPRESS_STATFILE": "ON",
        "CONCURRENT_NUM_OPTIMIZERS": 2,
        "CONCURRENT_PRIORITY_DUAL_SIMPLEX": 2,
        "CONCURRENT_PRIORITY_FREE_SIMPLEX": 3,
        "CONCURRENT_PRIORITY_INTPNT": 4,
        "CONCURRENT_PRIORITY_PRIMAL_SIMPLEX": 1,
        "FEASREPAIR_OPTIMIZE": "FEASREPAIR_OPTIMIZE_NONE",
        "INFEAS_GENERIC_NAMES": "OFF",
        "INFEAS_PREFER_PRIMAL": "ON",
        "INFEAS_REPORT_AUTO": "OFF",
        "INFEAS_REPORT_LEVEL": 1,
        "INTPNT_BASIS": "BI_ALWAYS",
    }
}

```

```

"INTPNT_DIFF_STEP": "ON",
"INTPNT_FACTOR_DEBUG_LVL": 0,
"INTPNT_FACTOR_METHOD": 0,
"INTPNT_HOTSTART": "INTPNT_HOTSTART_NONE",
"INTPNT_MAX_ITERATIONS": 400,
"INTPNT_MAX_NUM_COR": -1,
"INTPNT_MAX_NUM_REFINEMENT_STEPS": -1,
"INTPNT_OFF_COL_TRH": 40,
"INTPNT_ORDER_METHOD": "ORDER_METHOD_FREE",
"INTPNT_REGULARIZATION_USE": "ON",
"INTPNT_SCALING": "SCALING_FREE",
"INTPNT_SOLVE_FORM": "SOLVE_FREE",
"INTPNT_STARTING_POINT": "STARTING_POINT_FREE",
"LIC_TRH_EXPIRY_WRN": 7,
"LICENSE_DEBUG": "OFF",
"LICENSE_PAUSE_TIME": 0,
"LICENSE_SUPPRESS_EXPIRE_WRNS": "OFF",
"LICENSE_WAIT": "OFF",
"LOG": 10,
"LOG_ANA_PRO": 1,
"LOG_BI": 4,
"LOG_BI_FREQ": 2500,
"LOG_CHECK_CONVEXITY": 0,
"LOG_CONCURRENT": 1,
"LOG_CUT_SECOND_OPT": 1,
"LOG_EXPAND": 0,
"LOG_FACTOR": 1,
"LOG_FEAS_REPAIR": 1,
"LOG_FILE": 1,
"LOG_HEAD": 1,
"LOG_INFEAS_ANA": 1,
"LOG_INTPNT": 4,
"LOG_MIO": 4,
"LOG_MIO_FREQ": 1000,
"LOG_OPTIMIZER": 1,
"LOG_ORDER": 1,
"LOG_PRESOLVE": 1,
"LOG_RESPONSE": 0,
"LOG_SENSITIVITY": 1,
"LOG_SENSITIVITY_OPT": 0,
"LOG_SIM": 4,
"LOG_SIM_FREQ": 1000,
"LOG_SIM_MINOR": 1,
"LOG_STORAGE": 1,
"MAX_NUM_WARNINGS": 10,
"MIO_BRANCH_DIR": "BRANCH_DIR_FREE",
"MIO_CONSTRUCT_SOL": "OFF",
"MIO_CUT_CLIQUE": "ON",
"MIO_CUT_CMIR": "ON",
"MIO_CUT_GMI": "ON",
"MIO_CUT_KNAPSACK_COVER": "OFF",
"MIO_HEURISTIC_LEVEL": -1,
"MIO_MAX_NUM_BRANCHES": -1,
"MIO_MAX_NUM_RELAXS": -1,
"MIO_MAX_NUM_SOLUTIONS": -1,
"MIO_MODE": "MIO_MODE_SATISFIED",

```

```

"MIO_MT_USER_CB": "ON",
"MIO_NODE_OPTIMIZER": "OPTIMIZER_FREE",
"MIO_NODE_SELECTION": "MIO_NODE_SELECTION_FREE",
"MIO_PERSPECTIVE_REFORMULATE": "ON",
"MIO_PROBING_LEVEL": -1,
"MIO_RINS_MAX_NODES": -1,
"MIO_ROOT_OPTIMIZER": "OPTIMIZER_FREE",
"MIO_ROOT_REPEAT_PRESOLVE_LEVEL": -1,
"MT_SPINCOUNT": 0,
"NUM_THREADS": 0,
"OPF_MAX_TERMS_PER_LINE": 5,
"OPF_WRITE_HEADER": "ON",
"OPF_WRITE_HINTS": "ON",
"OPF_WRITE_PARAMETERS": "OFF",
"OPF_WRITE_PROBLEM": "ON",
"OPF_WRITE_SOL_BAS": "ON",
"OPF_WRITE_SOL_ITG": "ON",
"OPF_WRITE_SOL_ITR": "ON",
"OPF_WRITE_SOLUTIONS": "OFF",
"OPTIMIZER": "OPTIMIZER_FREE",
"PARAM_READ_CASE_NAME": "ON",
"PARAM_READ_IGN_ERROR": "OFF",
"PRESOLVE_ELIMINATOR_MAX_FILL": -1,
"PRESOLVE_ELIMINATOR_MAX_NUM_TRIES": -1,
"PRESOLVE_LEVEL": -1,
"PRESOLVE_LINDEP_ABS_WORK_TRH": 100,
"PRESOLVE_LINDEP_REL_WORK_TRH": 100,
"PRESOLVE_LINDEP_USE": "ON",
"PRESOLVE_MAX_NUM_REDUCTIONS": -1,
"PRESOLVE_USE": "PRESOLVE_MODE_FREE",
"PRIMAL_REPAIR_OPTIMIZER": "OPTIMIZER_FREE",
"QO_SEPARABLE_REFORMULATION": "OFF",
"READ_DATA_COMPRESSED": "COMPRESS_FREE",
"READ_DATA_FORMAT": "DATA_FORMAT_EXTENSION",
"READ_DEBUG": "OFF",
"READ_KEEP_FREE_CON": "OFF",
"READ_LP_DROP_NEW_VARS_IN_BOU": "OFF",
"READ_LP_QUOTED_NAMES": "ON",
"READ_MPS_FORMAT": "MPS_FORMAT_FREE",
"READ_MPS_WIDTH": 1024,
"READ_TASK_IGNORE_PARAM": "OFF",
"SENSITIVITY_ALL": "OFF",
"SENSITIVITY_OPTIMIZER": "OPTIMIZER_FREE_SIMPLEX",
"SENSITIVITY_TYPE": "SENSITIVITY_TYPE_BASIS",
"SIM_BASIS_FACTOR_USE": "ON",
"SIM_DEGEN": "SIM_DEGEN_FREE",
"SIM_DUAL_CRASH": 90,
"SIM_DUAL_PHASEONE_METHOD": 0,
"SIM_DUAL_RESTRICT_SELECTION": 50,
"SIM_DUAL_SELECTION": "SIM_SELECTION_FREE",
"SIM_EXPLOIT_DUPVEC": "SIM_EXPLOIT_DUPVEC_OFF",
"SIM_HOTSTART": "SIM_HOTSTART_FREE",
"SIM_HOTSTART_LU": "ON",
"SIM_INTEGER": 0,
"SIM_MAX_ITERATIONS": 10000000,
"SIM_MAX_NUM_SETBACKS": 250,

```

```

"SIM_NON_SINGULAR": "ON",
"SIM_PRIMAL_CRASH": 90,
"SIM_PRIMAL_PHASEONE_METHOD": 0,
"SIM_PRIMAL_RESTRICT_SELECTION": 50,
"SIM_PRIMAL_SELECTION": "SIM_SELECTION_FREE",
"SIM_REFACTOR_FREQ": 0,
"SIM_REFORMULATION": "SIM_REFORMULATION_OFF",
"SIM_SAVE_LU": "OFF",
"SIM_SCALING": "SCALING_FREE",
"SIM_SCALING_METHOD": "SCALING_METHOD_POW2",
"SIM_SOLVE_FORM": "SOLVE_FREE",
"SIM_STABILITY_PRIORITY": 50,
"SIM_SWITCH_OPTIMIZER": "OFF",
"SOL_FILTER_KEEP_BASIC": "OFF",
"SOL_FILTER_KEEP_RANGED": "OFF",
"SOL_READ_NAME_WIDTH": -1,
"SOL_READ_WIDTH": 1024,
"SOLUTION_CALLBACK": "OFF",
"TIMING_LEVEL": 1,
"WRITE_BAS_CONSTRAINTS": "ON",
"WRITE_BAS_HEAD": "ON",
"WRITE_BAS_VARIABLES": "ON",
"WRITE_DATA_COMPRESSED": 0,
"WRITE_DATA_FORMAT": "DATA_FORMAT_EXTENSION",
"WRITE_DATA_PARAM": "OFF",
"WRITE_FREE_CON": "OFF",
"WRITE_GENERIC_NAMES": "OFF",
"WRITE_GENERIC_NAMES_IO": 1,
"WRITE_IGNORE_INCOMPATIBLE_CONIC_ITEMS": "OFF",
"WRITE_IGNORE_INCOMPATIBLE_ITEMS": "OFF",
"WRITE_IGNORE_INCOMPATIBLE_NL_ITEMS": "OFF",
"WRITE_IGNORE_INCOMPATIBLE_PSD_ITEMS": "OFF",
"WRITE_INT_CONSTRAINTS": "ON",
"WRITE_INT_HEAD": "ON",
"WRITE_INT_VARIABLES": "ON",
"WRITE_LP_FULL_OBJ": "ON",
"WRITE_LP_LINE_WIDTH": 80,
"WRITE_LP_QUOTED_NAMES": "ON",
"WRITE_LP_STRICT_FORMAT": "OFF",
"WRITE_LP_TERMS_PER_LINE": 10,
"WRITE_MPS_FORMAT": "MPS_FORMAT_FREE",
"WRITE_MPS_INT": "ON",
"WRITE_PRECISION": 15,
"WRITE_SOL_BARVARIABLES": "ON",
"WRITE_SOL_CONSTRAINTS": "ON",
"WRITE_SOL_HEAD": "ON",
"WRITE_SOL_IGNORE_INVALID_NAMES": "OFF",
"WRITE_SOL_VARIABLES": "ON",
"WRITE_TASK_INC_SOL": "ON",
"WRITE_XML_MODE": "WRITE_XML_MODE_ROW"
},
"dparam": {
  "ANA_SOL_INFEAS_TOL": 1e-6,
  "BASIS_REL_TOL_S": 1e-12,
  "BASIS_TOL_S": 1e-6,
  "BASIS_TOL_X": 1e-6,

```

```

"CHECK_CONVEXITY_REL_TOL":1e-10,
"DATA_TOL_AIJ":1e-12,
"DATA_TOL_AIJ_HUGE":1e+20,
"DATA_TOL_AIJ_LARGE":1e+10,
"DATA_TOL_BOUND_INF":1e+16,
"DATA_TOL_BOUND_WRN":1e+8,
"DATA_TOL_C_HUGE":1e+16,
"DATA_TOL_CJ_LARGE":1e+8,
"DATA_TOL_QIJ":1e-16,
"DATA_TOL_X":1e-8,
"FEASREPAIR_TOL":1e-10,
"INTPNT_CO_TOL_DFEAS":1e-8,
"INTPNT_CO_TOL_INFEAS":1e-10,
"INTPNT_CO_TOL_MU_RED":1e-8,
"INTPNT_CO_TOL_NEAR_REL":1e+3,
"INTPNT_CO_TOL_PFEAS":1e-8,
"INTPNT_CO_TOL_REL_GAP":1e-7,
"INTPNT_NL_MERIT_BAL":1e-4,
"INTPNT_NL_TOL_DFEAS":1e-8,
"INTPNT_NL_TOL_MU_RED":1e-12,
"INTPNT_NL_TOL_NEAR_REL":1e+3,
"INTPNT_NL_TOL_PFEAS":1e-8,
"INTPNT_NL_TOL_REL_GAP":1e-6,
"INTPNT_NL_TOL_REL_STEP":9.95e-1,
"INTPNT_QO_TOL_DFEAS":1e-8,
"INTPNT_QO_TOL_INFEAS":1e-10,
"INTPNT_QO_TOL_MU_RED":1e-8,
"INTPNT_QO_TOL_NEAR_REL":1e+3,
"INTPNT_QO_TOL_PFEAS":1e-8,
"INTPNT_QO_TOL_REL_GAP":1e-8,
"INTPNT_TOL_DFEAS":1e-8,
"INTPNT_TOL_DSAFE":1e+0,
"INTPNT_TOL_INFEAS":1e-10,
"INTPNT_TOL_MU_RED":1e-16,
"INTPNT_TOL_PATH":1e-8,
"INTPNT_TOL_PFEAS":1e-8,
"INTPNT_TOL_PSAFE":1e+0,
"INTPNT_TOL_REL_GAP":1e-8,
"INTPNT_TOL_REL_STEP":9.999e-1,
"INTPNT_TOL_STEP_SIZE":1e-6,
"LOWER_OBJ_CUT":-1e+30,
"LOWER_OBJ_CUT_FINITE_TRH":-5e+29,
"MIO_DISABLE_TERM_TIME":-1e+0,
"MIO_MAX_TIME":-1e+0,
"MIO_MAX_TIME_APRX_OPT":6e+1,
"MIO_NEAR_TOL_ABS_GAP":0.0,
"MIO_NEAR_TOL_REL_GAP":1e-3,
"MIO_REL_GAP_CONST":1e-10,
"MIO_TOL_ABS_GAP":0.0,
"MIO_TOL_ABS_RELAX_INT":1e-5,
"MIO_TOL_FEAS":1e-6,
"MIO_TOL_REL_DUAL_BOUND_IMPROVEMENT":0.0,
"MIO_TOL_REL_GAP":1e-4,
"MIO_TOL_X":1e-6,
"OPTIMIZER_MAX_TIME":-1e+0,
"PRESOLVE_TOL_ABS_LINDEP":1e-6,

```

```

        "PRESOLVE_TOL_AIJ":1e-12,
        "PRESOLVE_TOL_REL_LINDEP":1e-10,
        "PRESOLVE_TOL_S":1e-8,
        "PRESOLVE_TOL_X":1e-8,
        "QCQO_REFORMULATE_REL_DROP_TOL":1e-15,
        "SEMIDEFINITE_TOL_APPROX":1e-10,
        "SIM_LU_TOL_REL_PIV":1e-2,
        "SIMPLEX_ABS_TOL_PIV":1e-7,
        "UPPER_OBJ_CUT":1e+30,
        "UPPER_OBJ_CUT_FINITE_TRH":5e+29
    },
    "sparam":{
        "BAS_SOL_FILE_NAME":"",
        "DATA_FILE_NAME":"examples/tools/data/lo1.mps",
        "DEBUG_FILE_NAME":"",
        "INT_SOL_FILE_NAME":"",
        "ITR_SOL_FILE_NAME":"",
        "MIO_DEBUG_STRING":"",
        "PARAM_COMMENT_SIGN":"%",
        "PARAM_READ_FILE_NAME":"",
        "PARAM_WRITE_FILE_NAME":"",
        "READ_MPS_BOU_NAME":"",
        "READ_MPS_OBJ_NAME":"",
        "READ_MPS_RAN_NAME":"",
        "READ_MPS_RHS_NAME":"",
        "SENSITIVITY_FILE_NAME":"",
        "SENSITIVITY_RES_FILE_NAME":"",
        "SOL_FILTER_XC_LOW":"",
        "SOL_FILTER_XC_UPR":"",
        "SOL_FILTER_XX_LOW":"",
        "SOL_FILTER_XX_UPR":"",
        "STAT_FILE_NAME":"",
        "STAT_KEY":"",
        "STAT_NAME":"",
        "WRITE_LP_GEN_VAR_NAME":"XMSKGEN"
    }
}
}

```

16.8 The Solution File Format

MOSEK provides several solution files depending on the problem type and the optimizer used:

- *basis solution file* (extension `.bas`) if the problem is optimized using the simplex optimizer or basis identification is performed,
- *interior solution file* (extension `.sol`) if a problem is optimized using the interior-point optimizer and no basis identification is required,
- *integer solution file* (extension `.int`) if the problem contains integer constrained variables.

All solution files have the format:

NAME	: <problem name>
PROBLEM STATUS	: <status of the problem>
SOLUTION STATUS	: <status of the solution>
OBJECTIVE NAME	: <name of the objective function>

(continues on next page)

(continued from previous page)

PRIMAL OBJECTIVE : <primal objective value corresponding to the solution>						
DUAL OBJECTIVE : <dual objective value corresponding to the solution>						
CONSTRAINTS						
INDEX	NAME	AT	ACTIVITY	LOWER LIMIT	UPPER LIMIT	DUAL LOWER
?	<name>	??	<a value>	<a value>	<a value>	<a value>
VARIABLES						
INDEX	NAME	AT	ACTIVITY	LOWER LIMIT	UPPER LIMIT	DUAL LOWER
?	<name>	??	<a value>	<a value>	<a value>	<a value>
↪ CONIC DUAL						
?	<name>	??	<a value>	<a value>	<a value>	<a value>
↪ <a value>						

In the example the fields ? and <> will be filled with problem and solution specific information. As can be observed a solution report consists of three sections, i.e.

- **HEADER** In this section, first the name of the problem is listed and afterwards the problem and solution status are shown. Next the primal and dual objective values are displayed.
- **CONSTRAINTS** For each constraint i of the form

$$l_i^c \leq \sum_{j=1}^n a_{ij}x_j \leq u_i^c, \quad (16.15)$$

the following information is listed:

- **INDEX**: A sequential index assigned to the constraint by **MOSEK**
- **NAME**: The name of the constraint assigned by the user.
- **AT**: The status of the constraint. In Table 16.4 the possible values of the status keys and their interpretation are shown.

Table 16.4: Status keys.

Status key	Interpretation
UN	Unknown status
BS	Is basic
SB	Is superbasic
LL	Is at the lower limit (bound)
UL	Is at the upper limit (bound)
EQ	Lower limit is identical to upper limit
**	Is infeasible i.e. the lower limit is greater than the upper limit.

- **ACTIVITY**: the quantity $\sum_{j=1}^n a_{ij}x_j^*$, where x^* is the value of the primal solution.
- **LOWER LIMIT**: the quantity l_i^c (see (16.15).)
- **UPPER LIMIT**: the quantity u_i^c (see (16.15).)
- **DUAL LOWER**: the dual multiplier corresponding to the lower limit on the constraint.
- **DUAL UPPER**: the dual multiplier corresponding to the upper limit on the constraint.
- **VARIABLES** The last section of the solution report lists information about the variables. This information has a similar interpretation as for the constraints. However, the column with the header **CONIC DUAL** is included for problems having one or more conic constraints. This column shows the dual variables corresponding to the conic constraints.

Example: lo1.sol

In Listing 16.7 we show the solution file for the lo1.opf problem.

Listing 16.7: An example of .sol file.

```

NAME          :
PROBLEM STATUS : PRIMAL_AND_DUAL_FEASIBLE
SOLUTION STATUS : OPTIMAL
OBJECTIVE NAME  : obj
PRIMAL OBJECTIVE : 8.33333333e+01
DUAL OBJECTIVE  : 8.33333332e+01

CONSTRAINTS
INDEX      NAME          AT ACTIVITY          LOWER LIMIT      UPPER LIMIT
↪      DUAL LOWER          DUAL UPPER
0          c1              EQ 3.00000000000000e+01      3.00000000e+01      3.
↪00000000e+01      -0.00000000000000e+00      -2.49999999741653e+00
1          c2              SB 5.33333333049187e+01      1.50000000e+01      NONE
↪          2.09159033069640e-10      -0.00000000000000e+00
2          c3              UL 2.49999999842049e+01      NONE              2.
↪50000000e+01      -0.00000000000000e+00      -3.33333332895108e-01

VARIABLES
INDEX      NAME          AT ACTIVITY          LOWER LIMIT      UPPER LIMIT
↪      DUAL LOWER          DUAL UPPER
0          x1              LL 1.67020427038537e-09      0.00000000e+00      NONE
↪          -4.49999999528054e+00      -0.00000000000000e+00
1          x2              LL 2.93510446211883e-09      0.00000000e+00      1.
↪00000000e+01      -2.16666666494915e+00      6.20868657679896e-10
2          x3              SB 1.49999999899424e+01      0.00000000e+00      NONE
↪          -8.79123177245553e-10      -0.00000000000000e+00
3          x4              SB 8.33333332273115e+00      0.00000000e+00      NONE
↪          -1.69795978848200e-09      -0.00000000000000e+00

```

Chapter 17

List of examples

List of examples shipped in the distribution of Optimizer API for Java:

Table 17.1: List of distributed examples

File	Description
blas_lapack. java	Demonstrates the MOSEK interface to BLAS/LAPACK linear algebra routines
callback.java	An example of data/progress callback
ceo1.java	A simple conic exponential problem
concurrent1. java	Implementation of a concurrent optimizer for linear and mixed-integer problems
cqo1.java	A simple conic quadratic problem
feasrepairex1. java	A simple example of how to repair an infeasible problem
gp1.java	A simple geometric program (GP) in conic form
lo1.java	A simple linear problem
lo2.java	A simple linear problem
logistic.java	Implements logistic regression and simple log-sum-exp (CEO)
mico1.java	A simple mixed-integer conic problem
milol.java	A simple mixed-integer linear problem
miointsol. java	A simple mixed-integer linear problem with an initial guess
modelLib.java	Library of implementations of basic functions
opt_server_async. java	Uses MOSEK OptServer to solve an optimization problem asynchronously
opt_server_sync. java	Uses MOSEK OptServer to solve an optimization problem synchronously
parallel.java	Demonstrates parallel optimization
parameters. java	Shows how to set optimizer parameters and read information items
portfolio_1_basic. java	Portfolio optimization - basic Markowitz model
portfolio_2_frontier. java	Portfolio optimization - efficient frontier
portfolio_3_impact. java	Portfolio optimization - market impact costs
portfolio_4_transaction. java	Portfolio optimization - transaction costs
portfolio_5_card. java	Portfolio optimization - cardinality constraints
pow1.java	A simple power cone problem
qcqo1.java	A simple quadratically constrained quadratic problem

continues on next page

Table 17.1 – continued from previous page

File	Description
qo1.java	A simple quadratic problem
reoptimization.java	Demonstrate how to modify and re-optimize a linear problem
response.java	Demonstrates proper response handling
sdo1.java	A simple semidefinite problem with one matrix variable and a quadratic cone
sdo2.java	A simple semidefinite problem with two matrix variables
sensitivity.java	Sensitivity analysis performed on a small linear problem
simple.java	A simple I/O example: read problem from a file, solve and write solutions
solutionquality.java	Demonstrates how to examine the quality of a solution
solvebasis.java	Demonstrates solving a linear system with the basis matrix
solvelinear.java	Demonstrates solving a general linear system

Additional examples can be found on the **MOSEK** website and in other **MOSEK** publications.

Chapter 18

Interface changes

The section shows interface-specific changes to the **MOSEK** Optimizer API for Java in version 9.3 compared to version 8. See the [release notes](#) for general changes and new features of the **MOSEK** Optimization Suite.

18.1 Backwards compatibility

- **Parameters.** Users who set parameters to tune the performance and numerical properties of the solver (termination criteria, tolerances, solving primal or dual, presolve etc.) are recommended to reevaluate such tuning. It may be that other, or default, parameter settings will be more beneficial in the current version. The hints in [Sec. 8](#) may be useful for some cases.
- All functions using the enum `accmode` were removed. Use corresponding separate functions for manipulating variables and constraints. For example, instead of

```
task.putbound(accmode.var, ...);  
task.putbound(accmode.con, ...);
```

use

```
task.putvarbound(...);  
task.putconbound(...);
```

and so on.

- Removed all Near problem and solution statuses i.e. `solsta.near_optimal`, `solsta.near_prim_infeas_cer`, etc. See [Sec. 13.3.3](#).
- All functions related to the general nonlinear optimizer and `Scopt` have been removed. See [Sec. 15.11](#).

18.2 Functions

Added

- *`Env.setupthreads`*
- *`Task.appendsparsesymmatlist`*
- *`Task.generateconenames`*
- *`Task.generateconnames`*
- *`Task.generatevarnames`*
- *`Task.getacolslice`*

- *Task.getacolslicenumz*
- *Task.getarowslice*
- *Task.getarowslicenumz*
- *Task.getatruncatetol*
- *Task.getbarsslice*
- *Task.getbaræslice*
- *Task.getclist*
- *Task.getskn*
- *Task.putatruncatetol*
- *Task.putbaraijlist*
- *Task.putbararowlist*
- *Task.putconboundlistconst*
- *Task.putconboundsliceconst*
- *Task.putconsolutioni*
- *Task.putoptserverhost*
- *Task.putvarboundlistconst*
- *Task.putvarboundsliceconst*
- *Task.putvarsolutionj*
- *Task.readjsonstring*
- *Task.readlpstring*
- *Task.readopfstring*
- *Task.readptfstring*

Removed

- *Task.checkconvexity*
- *Task.chgbound*
- *Task.getaslice*
- *Task.getaslicenumz*
- *Task.getbound*
- *Task.getboundslice*
- *Task.getsolutioni*
- *Task.putbound*
- *Task.putboundlist*
- *Task.putboundslice*
- *Task.putsolutioni*

18.3 Parameters

Added

- *iparam.intpnt_order_gp_num_seeds*
- *iparam.intpnt_purify*
- *iparam.log_include_summary*
- *iparam.log_local_info*
- *iparam.mio_conic_outer_approximation*
- *iparam.mio_feaspump_level*
- *iparam.mio_max_num_root_cut_rounds*
- *iparam.mio_propagate_objective_constraint*
- *iparam.mio_seed*
- *iparam.opf_write_line_length*
- *iparam.presolve_max_num_pass*
- *iparam.ptf_write_transform*
- *iparam.sim_seed*
- *iparam.write_compression*

Removed

- *dparam.data_tol_aij*
- *dparam.intpnt_nl_merit_bal*
- *dparam.intpnt_nl_tol_dfeas*
- *dparam.intpnt_nl_tol_mu_red*
- *dparam.intpnt_nl_tol_near_rel*
- *dparam.intpnt_nl_tol_pfeas*
- *dparam.intpnt_nl_tol_rel_gap*
- *dparam.intpnt_nl_tol_rel_step*
- *dparam.mio_disable_term_time*
- *dparam.mio_near_tol_abs_gap*
- *dparam.mio_near_tol_rel_gap*
- *iparam.mio_construct_sol*
- *iparam.mio_mt_user_cb*
- *iparam.opf_max_terms_per_line*
- *iparam.read_data_compressed*
- *iparam.read_data_format*
- *iparam.write_data_compressed*
- *iparam.write_data_format*

18.4 Constants

Added

- *compresstype.zstd*
- *conetype.dexp*
- *conetype.dpow*
- *conetype.pexp*
- *conetype.ppow*
- *conetype.zero*
- *dataformat.ptf*
- *infitem.mio_numbin*
- *infitem.mio_numbinconevar*
- *infitem.mio_numcone*
- *infitem.mio_numconevar*
- *infitem.mio_numcont*
- *infitem.mio_numcontconevar*
- *infitem.mio_numdexpcones*
- *infitem.mio_numdpowcones*
- *infitem.mio_numintconevar*
- *infitem.mio_numpepcones*
- *infitem.mio_numppowcones*
- *infitem.mio_numqcones*
- *infitem.mio_numrqcones*
- *infitem.mio_presolved_numbinconevar*
- *infitem.mio_presolved_numcone*
- *infitem.mio_presolved_numconevar*
- *infitem.mio_presolved_numcontconevar*
- *infitem.mio_presolved_numdexpcones*
- *infitem.mio_presolved_numdpowcones*
- *infitem.mio_presolved_numintconevar*
- *infitem.mio_presolved_numpepcones*
- *infitem.mio_presolved_numppowcones*
- *infitem.mio_presolved_numqcones*
- *infitem.mio_presolved_numrqcones*
- *infitem.purify_dual_success*
- *infitem.purify_primal_success*
- *linfitem.mio_anz*

Removed

- `constant.dataformat.xml`
- `constant.dinfitem.mio_heuristic_time`
- `constant.dinfitem.mio_optimizer_time`
- `constant.iinfitem.mio_construct_num_roundings`
- `constant.iinfitem.mio_initial_solution`
- `constant.iinfitem.mio_near_absgap_satisfied`
- `constant.iinfitem.mio_near_relgap_satisfied`
- `constant.liinfitem.mio_sim_maxiter_setbacks`
- `constant.mionodeseltype.hybrid`
- `constant.mionodeseltype.worst`
- `constant.problemtype.geco`
- `constant.prosta.near_dual_feas`
- `constant.prosta.near_prim_and_dual_feas`
- `constant.prosta.near_prim_feas`
- `constant.sensitivitytype.optimal_partition`
- `constant.solsta.near_dual_feas`
- `constant.solsta.near_dual_infeas_cer`
- `constant.solsta.near_integer_optimal`
- `constant.solsta.near_optimal`
- `constant.solsta.near_prim_and_dual_feas`
- `constant.solsta.near_prim_feas`
- `constant.solsta.near_prim_infeas_cer`

18.5 Response Codes

Added

- *`rescode.err_appending_too_big_cone`*
- *`rescode.err_cbf_duplicate_pow_cones`*
- *`rescode.err_cbf_duplicate_pow_star_cones`*
- *`rescode.err_cbf_invalid_dimension_of_cones`*
- *`rescode.err_cbf_invalid_exp_dimension`*
- *`rescode.err_cbf_invalid_number_of_cones`*
- *`rescode.err_cbf_invalid_power`*
- *`rescode.err_cbf_invalid_power_cone_index`*

- `rescode.err_cbf_invalid_power_star_cone_index`
- `rescode.err_cbf_power_cone_is_too_long`
- `rescode.err_cbf_power_cone_mismatch`
- `rescode.err_cbf_power_star_cone_mismatch`
- `rescode.err_cbf_unhandled_power_cone_type`
- `rescode.err_cbf_unhandled_power_star_cone_type`
- `rescode.err_cone_parameter`
- `rescode.err_format_string`
- `rescode.err_invalid_cj`
- `rescode.err_invalid_file_format_for_cfix`
- `rescode.err_invalid_file_format_for_free_constraints`
- `rescode.err_invalid_file_format_for_nonlinear`
- `rescode.err_invalid_file_format_for_ranged_constraints`
- `rescode.err_num_arguments`
- `rescode.err_ptf_format`
- `rescode.err_server_problem_size`
- `rescode.err_shape_is_too_large`
- `rescode.err_slice_size`
- `rescode.err_too_small_a_truncation_value`
- `rescode.wrn_exp_cones_with_variables_fixed_at_zero`
- `rescode.wrn_pow_cones_with_root_fixed_at_zero`

Removed

- `rescode.err_cannot_clone_nl`
- `rescode.err_cannot_handle_nl`
- `rescode.err_invalid_accmode`
- `rescode.err_invalid_file_format_for_general_nl`
- `rescode.err_nonlinear_functions_not_allowed`
- `rescode.err_nr_arguments`
- `rescode.err_open_dl`
- `rescode.err_user_func_ret`
- `rescode.err_user_func_ret_data`
- `rescode.err_user_nlo_eval`
- `rescode.err_user_nlo_eval_hessubi`
- `rescode.err_user_nlo_eval_hessubj`

- `rescode.err_user_nlo_func`
- `rescode.trm_mio_near_abs_gap`
- `rescode.trm_mio_near_rel_gap`
- `rescode.wrn_construct_invalid_sol_itg`
- `rescode.wrn_construct_no_sol_itg`
- `rescode.wrn_construct_solution_infeas`
- `rescode.wrn_no_nonlinear_function_write`

Bibliography

- [AA95] E. D. Andersen and K. D. Andersen. Presolving in linear programming. *Math. Programming*, 71(2):221–245, 1995.
- [AGMeszarosX96] E. D. Andersen, J. Gondzio, Cs. Mészáros, and X. Xu. Implementation of interior point methods for large scale linear programming. In T. Terlaky, editor, *Interior-point methods of mathematical programming*, pages 189–252. Kluwer Academic Publishers, 1996.
- [ART03] E. D. Andersen, C. Roos, and T. Terlaky. On implementing a primal-dual interior-point method for conic quadratic optimization. *Math. Programming*, February 2003.
- [AY96] E. D. Andersen and Y. Ye. Combining interior-point and pivoting algorithms. *Management Sci.*, 42(12):1719–1731, December 1996.
- [And09] Erling D. Andersen. The homogeneous and self-dual model and algorithm for linear optimization. Technical Report TR-1-2009, MOSEK ApS, 2009. URL: <http://docs.mosek.com/whitepapers/homolo.pdf>.
- [And13] Erling D. Andersen. On formulating quadratic functions in optimization models. Technical Report TR-1-2013, MOSEK ApS, 2013. Last revised 23-feb-2016. URL: <http://docs.mosek.com/whitepapers/qmodel.pdf>.
- [BKVH07] S. Boyd, S.J. Kim, L. Vandenbergh, and A. Hassibi. A Tutorial on Geometric Programming. *Optimization and Engineering*, 8(1):67–127, 2007. Available at http://www.stanford.edu/boyd/gp_tutorial.html.
- [Chvatal83] V. Chvátal. *Linear programming*. W.H. Freeman and Company, 1983.
- [GK00] Richard C. Grinold and Ronald N. Kahn. *Active portfolio management*. McGraw-Hill, New York, 2 edition, 2000.
- [Naz87] J. L. Nazareth. *Computer Solution of Linear Programs*. Oxford University Press, New York, 1987.
- [RTV97] C. Roos, T. Terlaky, and J. -Ph. Vial. *Theory and algorithms for linear optimization: an interior point approach*. John Wiley and Sons, New York, 1997.
- [Ste98] G. W. Stewart. *Matrix Algorithms. Volume 1: Basic decompositions*. SIAM, 1998.
- [Wal00] S. W. Wallace. Decision making under uncertainty: is sensitivity of any use. *Oper. Res.*, 48(1):20–25, January 2000.
- [Wol98] L. A. Wolsey. *Integer programming*. John Wiley and Sons, 1998.

Symbol Index

Classes

DataCallback, 412
DataCallback.callback, 412
Env, 208
Env.syrk, 217
Env.syevd, 216
Env.syeig, 216
Env.sparsetriangularsolvedense, 215
Env.setupthreads, 215
Env.set_Stream, 215
Env.putlicensewait, 214
Env.putlicensepath, 214
Env.putlicensedebug, 214
Env.putlicensecode, 214
Env.potrf, 214
Env.linkfiletostream, 213
Env.licensecleanup, 213
Env.getversion, 213
Env.getcodedesc, 213
Env.gemv, 212
Env.gemm, 211
Env.Env, 208
Env.echointro, 211
Env.dot, 211
Env.dispose, 210
Env.computesparseseholesky, 209
Env.checkoutlicense, 209
Env.checkinlicense, 209
Env.checkinall, 208
Env.axy, 208
ItgSolutionCallback, 412
ItgSolutionCallback.callback, 412
Progress, 412
Progress.progress, 412
Stream, 413
Stream.stream, 413
Task, 217
Task.writetask, 312
Task.writesolution, 312
Task.writeparamfile, 312
Task.writejsonsol, 312
Task.writedata, 311
Task.updatesolutioninfo, 311
Task.unset_Progress, 311
Task.toconic, 311
Task.Task, 217
Task.strtosk, 311
Task.strtoconetype, 310
Task.solvewithbasis, 309
Task.solutionsummary, 309
Task.solutiondef, 309
Task.setdefaults, 309
Task.set_Stream, 308
Task.set_Progress, 308
Task.set_ItgSolutionCallback, 308
Task.set_InfoCallback, 308
Task.sensitivityreport, 307
Task.resizetask, 307
Task.removevars, 307
Task.removecons, 307
Task.removecones, 307
Task.removebarvars, 306
Task.readtask, 306
Task.readsummary, 306
Task.readsolution, 306
Task.readptfstring, 305
Task.readparamfile, 305
Task.readopfstring, 305
Task.readlpstring, 305
Task.readjsonstring, 305
Task.readdataformat, 304
Task.readdata, 304
Task.putyslice, 304
Task.puty, 304
Task.putxxslice, 304
Task.putxx, 303
Task.putxcslice, 303
Task.putxc, 303
Task.putvartypelist, 302
Task.putvartype, 302
Task.putvarsolutionj, 302
Task.putvarname, 302
Task.putvarboundsliceconst, 301
Task.putvarboundslice, 301
Task.putvarboundlistconst, 301
Task.putvarboundlist, 300
Task.putvarbound, 300
Task.puttaskname, 300
Task.putsuxslice, 299
Task.putsux, 299
Task.putsucslice, 299
Task.putsuc, 299
Task.putstrparam, 298
Task.putsolutionyi, 298
Task.putsolution, 297
Task.putsnxslice, 297
Task.putsnx, 297

Task.putslxslice, 296
 Task.putslx, 296
 Task.putslcslice, 296
 Task.putslc, 296
 Task.putskxslice, 295
 Task.putskx, 295
 Task.putskcslice, 295
 Task.putskc, 295
 Task.putqobjij, 294
 Task.putqobj, 294
 Task.putqconk, 293
 Task.putqcon, 293
 Task.putparam, 293
 Task.putoptserverhost, 292
 Task.putobjsense, 292
 Task.putobjname, 292
 Task.putnastrparam, 292
 Task.putnaintparam, 292
 Task.putnadoupparam, 291
 Task.putmaxnumvar, 291
 Task.putmaxnumqnz, 291
 Task.putmaxnumcone, 290
 Task.putmaxnumcon, 290
 Task.putmaxnumbarvar, 290
 Task.putmaxnumanz, 290
 Task.putintparam, 289
 Task.putdoupparam, 289
 Task.putcslice, 289
 Task.putconsolutioni, 288
 Task.putconname, 288
 Task.putconename, 288
 Task.putcone, 287
 Task.putconboundsliceconst, 287
 Task.putconboundslice, 287
 Task.putconboundlistconst, 286
 Task.putconboundlist, 286
 Task.putconbound, 286
 Task.putclist, 285
 Task.putcj, 285
 Task.putcfix, 285
 Task.putbarxj, 285
 Task.putbarvarname, 284
 Task.putbarsj, 284
 Task.putbarcj, 284
 Task.putbarcblocktriplet, 283
 Task.putbararowlist, 283
 Task.putbaraijlist, 282
 Task.putbaraij, 282
 Task.putbarablocktriplet, 282
 Task.putatruncatetol, 281
 Task.putarowslice, 281
 Task.putarowlist, 280
 Task.putarow, 280
 Task.putaijlist, 279
 Task.putaij, 279
 Task.putacolslice, 279
 Task.putacollist, 278
 Task.putacol, 278
 Task.primalsensitivity, 277
 Task.primalrepair, 276
 Task.optimizersummary, 276
 Task.optimizermt, 276
 Task.optimize, 275
 Task.onesolutionsummary, 275
 Task.linkfiletostream, 275
 Task.isstrparname, 275
 Task.isintparname, 274
 Task.isdoupname, 274
 Task.inputdata, 273
 Task.initbasissolve, 273
 Task.getyslice, 272
 Task.gety, 272
 Task.getxxslice, 272
 Task.getxx, 272
 Task.getxcslice, 271
 Task.getxc, 271
 Task.getvartypelist, 271
 Task.getvartype, 270
 Task.getvarnamelen, 270
 Task.getvarnameindex, 270
 Task.getvarname, 269
 Task.getvarboundslice, 269
 Task.getvarbound, 269
 Task.gettasknamelen, 268
 Task.gettaskname, 268
 Task.getsymmatinfo, 268
 Task.getsuxslice, 268
 Task.getsux, 267
 Task.getsucslice, 267
 Task.getsuc, 267
 Task.getstrparamlen, 266
 Task.getstrparam, 266
 Task.getsparsesymmat, 266
 Task.getsolutionslice, 265
 Task.getsolutioninfo, 265
 Task.getsolution, 263
 Task.getsolsta, 263
 Task.getsnxslice, 262
 Task.getsnx, 262
 Task.getslxslice, 262
 Task.getslx, 261
 Task.getslcslice, 261
 Task.getslc, 261
 Task.getskxslice, 261
 Task.getskx, 260
 Task.getskn, 260
 Task.getskcslice, 260
 Task.getskc, 260
 Task.getreducedcosts, 259
 Task.getqobjij, 259
 Task.getqobj, 259
 Task.getqconk, 258
 Task.getpviolvar, 258
 Task.getpviolcones, 257
 Task.getpviolcon, 257
 Task.getpviolbarvar, 257

Task.getprosta, 256
 Task.getprobtype, 256
 Task.getprimalsolutionnorms, 256
 Task.getprimalobj, 255
 Task.getparamname, 255
 Task.getparammax, 255
 Task.getobjsense, 255
 Task.getobjnamelen, 254
 Task.getobjname, 254
 Task.getnumvar, 254
 Task.getnumsymmat, 254
 Task.getnumqobjnz, 254
 Task.getnumqconknz, 253
 Task.getnumparam, 253
 Task.getnumintvar, 253
 Task.getnumconemem, 253
 Task.getnumcone, 252
 Task.getnumcon, 252
 Task.getnumbarvar, 252
 Task.getnumbarcnz, 252
 Task.getnumbarcbloctriplets, 251
 Task.getnumbaranz, 251
 Task.getnumbarablocktriplets, 251
 Task.getnumanz64, 251
 Task.getnumanz, 250
 Task.getmemusage, 250
 Task.getmaxnumvar, 250
 Task.getmaxnumqnz, 250
 Task.getmaxnumcone, 250
 Task.getmaxnumcon, 249
 Task.getmaxnumbarvar, 249
 Task.getmaxnumanz, 249
 Task.getlintinf, 249
 Task.getlenbarvarj, 248
 Task.getintparam, 248
 Task.getintinf, 248
 Task.getinfname, 247
 Task.getinfmax, 247
 Task.getinfindex, 247
 Task.getinfeasiblesubproblem, 247
 Task.getdviolvar, 246
 Task.getdviolcones, 246
 Task.getdviolcon, 245
 Task.getdviolbarvar, 245
 Task.getdualsolutionnorms, 244
 Task.getdualobj, 244
 Task.getdoupparam, 244
 Task.getdouinf, 243
 Task.getdimbarvarj, 243
 Task.getcslice, 243
 Task.getconnamelen, 243
 Task.getconnameindex, 242
 Task.getconname, 242
 Task.getconenamelen, 241
 Task.getconenameindex, 241
 Task.getconename, 241
 Task.getconeinfo, 240
 Task.getcone, 240
 Task.getconboundslice, 240
 Task.getconbound, 239
 Task.getclist, 239
 Task.getcj, 239
 Task.getcfix, 239
 Task.getc, 238
 Task.getbarxslice, 238
 Task.getbarxj, 238
 Task.getbarvarnamelen, 238
 Task.getbarvarnameindex, 237
 Task.getbarvarname, 237
 Task.getbarsslice, 236
 Task.getbarsj, 236
 Task.getbarcsparsity, 236
 Task.getbarcidxj, 236
 Task.getbarcidxinfo, 235
 Task.getbarcidx, 235
 Task.getbarcbloctriplet, 234
 Task.getbarasparsity, 234
 Task.getbaraidxinfo, 234
 Task.getbaraidxj, 233
 Task.getbaraidx, 233
 Task.getbarablocktriplet, 232
 Task.getatruncatetol, 232
 Task.getarowslicetrip, 232
 Task.getarowslicenumnz, 231
 Task.getarowslice, 231
 Task.getarownumnz, 231
 Task.getarow, 230
 Task.getapiecenumnz, 230
 Task.getaij, 229
 Task.getacolslicetrip, 229
 Task.getacolslicenumnz, 229
 Task.getacolslice, 228
 Task.getacolnumnz, 228
 Task.getacol, 228
 Task.generatevarnames, 227
 Task.generateconnames, 227
 Task.generateconenames, 227
 Task.dualsensitivity, 226
 Task.dispose, 226
 Task.deletesolution, 226
 Task.commitchanges, 226
 Task.chgvarbound, 225
 Task.chgconbound, 225
 Task.checkmem, 225
 Task.basiscond, 224
 Task.asyncstop, 224
 Task.asyncpoll, 223
 Task.asyncoptimize, 223
 Task.asyncgetresult, 222
 Task.appendvars, 222
 Task.appendsparsesymmatlist, 221
 Task.appendsparsesymmat, 221
 Task.appendcons, 221
 Task.appendconesseq, 220
 Task.appendconeseq, 220
 Task.appendcone, 219

- Task.appendbarvars, 219
- Task.analyzesolution, 218
- Task.analyzeproblem, 218
- Task.analyzenames, 218

Enumerations

- basindtype, 388
- basindtype.reserved, 388
- basindtype.no_error, 388
- basindtype.never, 388
- basindtype.if_feasible, 388
- basindtype.always, 388
- boundkey, 388
- boundkey.up, 388
- boundkey.ra, 388
- boundkey.lo, 388
- boundkey.fx, 388
- boundkey.fr, 388
- branchdir, 405
- branchdir.up, 405
- branchdir.root_lp, 406
- branchdir.pseudocost, 406
- branchdir.near, 405
- branchdir.guided, 406
- branchdir.free, 405
- branchdir.far, 406
- branchdir.down, 405
- callbackcode, 390
- callbackcode.write_opf, 394
- callbackcode.update_primal_simplex_bi, 394
- callbackcode.update_primal_simplex, 394
- callbackcode.update_primal_bi, 394
- callbackcode.update_presolve, 394
- callbackcode.update_dual_simplex_bi, 394
- callbackcode.update_dual_simplex, 394
- callbackcode.update_dual_bi, 394
- callbackcode.solving_remote, 394
- callbackcode.read_opf_section, 394
- callbackcode.read_opf, 394
- callbackcode.primal_simplex, 393
- callbackcode.new_int_mio, 393
- callbackcode.intpnt, 393
- callbackcode.im_simplex_bi, 393
- callbackcode.im_simplex, 393
- callbackcode.im_root_cutgen, 393
- callbackcode.im_read, 393
- callbackcode.im_qo_reformulate, 393
- callbackcode.im_primal_simplex, 393
- callbackcode.im_primal_sensitivity, 393
- callbackcode.im_primal_bi, 393
- callbackcode.im_presolve, 393
- callbackcode.im_order, 393
- callbackcode.im_mio_primal_simplex, 393
- callbackcode.im_mio_intpnt, 393
- callbackcode.im_mio_dual_simplex, 393
- callbackcode.im_mio, 393
- callbackcode.im_lu, 393
- callbackcode.im_license_wait, 393

- callbackcode.im_intpnt, 393
- callbackcode.im_full_convexity_check, 392
- callbackcode.im_dual_simplex, 392
- callbackcode.im_dual_sensitivity, 392
- callbackcode.im_dual_bi, 392
- callbackcode.im_conic, 392
- callbackcode.im_bi, 392
- callbackcode.end_write, 392
- callbackcode.end_to_conic, 392
- callbackcode.end_simplex_bi, 392
- callbackcode.end_simplex, 392
- callbackcode.end_root_cutgen, 392
- callbackcode.end_read, 392
- callbackcode.end_qcqp_reformulate, 392
- callbackcode.end_primal_simplex_bi, 392
- callbackcode.end_primal_simplex, 392
- callbackcode.end_primal_setup_bi, 392
- callbackcode.end_primal_sensitivity, 392
- callbackcode.end_primal_repair, 392
- callbackcode.end_primal_bi, 392
- callbackcode.end_presolve, 392
- callbackcode.end_optimizer, 392
- callbackcode.end_mio, 392
- callbackcode.end_license_wait, 391
- callbackcode.end_intpnt, 391
- callbackcode.end_infeas_ana, 391
- callbackcode.end_full_convexity_check, 391
- callbackcode.end_dual_simplex_bi, 391
- callbackcode.end_dual_simplex, 391
- callbackcode.end_dual_setup_bi, 391
- callbackcode.end_dual_sensitivity, 391
- callbackcode.end_dual_bi, 391
- callbackcode.end_conic, 391
- callbackcode.end_bi, 391
- callbackcode.dual_simplex, 391
- callbackcode.conic, 391
- callbackcode.begin_write, 391
- callbackcode.begin_to_conic, 391
- callbackcode.begin_simplex_bi, 391
- callbackcode.begin_simplex, 391
- callbackcode.begin_root_cutgen, 391
- callbackcode.begin_read, 391
- callbackcode.begin_qcqp_reformulate, 391
- callbackcode.begin_primal_simplex_bi, 391
- callbackcode.begin_primal_simplex, 391
- callbackcode.begin_primal_setup_bi, 391
- callbackcode.begin_primal_sensitivity, 390
- callbackcode.begin_primal_repair, 390
- callbackcode.begin_primal_bi, 390
- callbackcode.begin_presolve, 390
- callbackcode.begin_optimizer, 390
- callbackcode.begin_mio, 390
- callbackcode.begin_license_wait, 390
- callbackcode.begin_intpnt, 390
- callbackcode.begin_infeas_ana, 390
- callbackcode.begin_full_convexity_check, 390
- callbackcode.begin_dual_simplex_bi, 390

callbackcode.begin_dual_simplex, 390
 callbackcode.begin_dual_setup_bi, 390
 callbackcode.begin_dual_sensitivity, 390
 callbackcode.begin_dual_bi, 390
 callbackcode.begin_conic, 390
 callbackcode.begin_bi, 390
 checkconvexitytype, 394
 checkconvexitytype.simple, 394
 checkconvexitytype.none, 394
 checkconvexitytype.full, 394
 compresstype, 394
 compresstype.zstd, 394
 compresstype.none, 394
 compresstype.gzip, 394
 compresstype.free, 394
 conetype, 394
 conetype.zero, 395
 conetype.rquad, 394
 conetype.quad, 394
 conetype.ppow, 395
 conetype.pexp, 395
 conetype.dpow, 395
 conetype.dexp, 395
 dataformat, 395
 dataformat.task, 395
 dataformat.ptf, 395
 dataformat.op, 395
 dataformat.mps, 395
 dataformat.lp, 395
 dataformat.json_task, 396
 dataformat.free_mps, 395
 dataformat.extension, 395
 dataformat.cb, 396
 dinfitem, 396
 dinfitem.to_conic_time, 400
 dinfitem.sol_itr_pviolvar, 400
 dinfitem.sol_itr_pviolcones, 400
 dinfitem.sol_itr_pviolcon, 400
 dinfitem.sol_itr_pviolbarvar, 400
 dinfitem.sol_itr_primal_obj, 400
 dinfitem.sol_itr_nrm_y, 400
 dinfitem.sol_itr_nrm_xx, 400
 dinfitem.sol_itr_nrm_xc, 400
 dinfitem.sol_itr_nrm_sux, 399
 dinfitem.sol_itr_nrm_suc, 399
 dinfitem.sol_itr_nrm_snx, 399
 dinfitem.sol_itr_nrm_slx, 399
 dinfitem.sol_itr_nrm_slc, 399
 dinfitem.sol_itr_nrm_barx, 399
 dinfitem.sol_itr_nrm_bars, 399
 dinfitem.sol_itr_dviolvar, 399
 dinfitem.sol_itr_dviolcones, 399
 dinfitem.sol_itr_dviolcon, 399
 dinfitem.sol_itr_dviolbarvar, 399
 dinfitem.sol_itr_dual_obj, 399
 dinfitem.sol_itg_pviolvar, 399
 dinfitem.sol_itg_pviolitg, 399
 dinfitem.sol_itg_pviolcones, 399
 dinfitem.sol_itg_pviolcon, 399
 dinfitem.sol_itg_pviolbarvar, 399
 dinfitem.sol_itg_primal_obj, 399
 dinfitem.sol_itg_nrm_xx, 399
 dinfitem.sol_itg_nrm_xc, 399
 dinfitem.sol_itg_nrm_barx, 399
 dinfitem.sol_bas_pviolvar, 398
 dinfitem.sol_bas_pviolcon, 398
 dinfitem.sol_bas_primal_obj, 398
 dinfitem.sol_bas_nrm_y, 398
 dinfitem.sol_bas_nrm_xx, 398
 dinfitem.sol_bas_nrm_xc, 398
 dinfitem.sol_bas_nrm_sux, 398
 dinfitem.sol_bas_nrm_suc, 398
 dinfitem.sol_bas_nrm_slx, 398
 dinfitem.sol_bas_nrm_slc, 398
 dinfitem.sol_bas_nrm_barx, 398
 dinfitem.sol_bas_dviolvar, 398
 dinfitem.sol_bas_dviolcon, 398
 dinfitem.sol_bas_dual_obj, 398
 dinfitem.sim_time, 398
 dinfitem.sim_primal_time, 398
 dinfitem.sim_obj, 398
 dinfitem.sim_feas, 398
 dinfitem.sim_dual_time, 398
 dinfitem.rd_time, 398
 dinfitem.qcqq_reformulate_worst_cholesky_diag_scaling, 398
 dinfitem.qcqq_reformulate_worst_cholesky_column_scaling, 398
 dinfitem.qcqq_reformulate_time, 398
 dinfitem.qcqq_reformulate_max_perturbation, 397
 dinfitem.primal_repair_penalty_obj, 397
 dinfitem.presolve_time, 397
 dinfitem.presolve_lindep_time, 397
 dinfitem.presolve_eli_time, 397
 dinfitem.optimizer_time, 397
 dinfitem.mio_user_obj_cut, 397
 dinfitem.mio_time, 397
 dinfitem.mio_root_presolve_time, 397
 dinfitem.mio_root_optimizer_time, 397
 dinfitem.mio_root_cutgen_time, 397
 dinfitem.mio_probing_time, 397
 dinfitem.mio_obj_rel_gap, 397
 dinfitem.mio_obj_int, 397
 dinfitem.mio_obj_bound, 397
 dinfitem.mio_obj_abs_gap, 397
 dinfitem.mio_knapsack_cover_separation_time, 397
 dinfitem.mio_implied_bound_time, 397
 dinfitem.mio_gmi_separation_time, 396
 dinfitem.mio_dual_bound_after_presolve, 396
 dinfitem.mio_construct_solution_obj, 396
 dinfitem.mio_cmir_separation_time, 396
 dinfitem.mio_clique_separation_time, 396
 dinfitem.intpnt_time, 396
 dinfitem.intpnt_primal_obj, 396

dinfitem.intpnt_primal_feas, 396
 dinfitem.intpnt_order_time, 396
 dinfitem.intpnt_opt_status, 396
 dinfitem.intpnt_factor_num_flops, 396
 dinfitem.intpnt_dual_obj, 396
 dinfitem.intpnt_dual_feas, 396
 dinfitem.bi_time, 396
 dinfitem.bi_primal_time, 396
 dinfitem.bi_dual_time, 396
 dinfitem.bi_clean_time, 396
 dinfitem.bi_clean_primal_time, 396
 dinfitem.bi_clean_dual_time, 396
 dparam, 325
 feature, 400
 feature.pts, 400
 feature.pton, 400
 iinfitem, 401
 iinfitem.sto_num_a_realloc, 405
 iinfitem.sol_itr_solsta, 405
 iinfitem.sol_itr_prosta, 405
 iinfitem.sol_itg_solsta, 405
 iinfitem.sol_itg_prosta, 405
 iinfitem.sol_bas_solsta, 405
 iinfitem.sol_bas_prosta, 405
 iinfitem.sim_solve_dual, 405
 iinfitem.sim_primal_iter, 405
 iinfitem.sim_primal_inf_iter, 405
 iinfitem.sim_primal_hotstart_lu, 405
 iinfitem.sim_primal_hotstart, 405
 iinfitem.sim_primal_deg_iter, 404
 iinfitem.sim_numvar, 404
 iinfitem.sim_numcon, 404
 iinfitem.sim_dual_iter, 404
 iinfitem.sim_dual_inf_iter, 404
 iinfitem.sim_dual_hotstart_lu, 404
 iinfitem.sim_dual_hotstart, 404
 iinfitem.sim_dual_deg_iter, 404
 iinfitem.rd_prototype, 404
 iinfitem.rd_numvar, 404
 iinfitem.rd_numq, 404
 iinfitem.rd_numintvar, 404
 iinfitem.rd_numcone, 404
 iinfitem.rd_numcon, 404
 iinfitem.rd_numbarvar, 404
 iinfitem.purify_primal_success, 404
 iinfitem.purify_dual_success, 404
 iinfitem.optimize_response, 404
 iinfitem.opt_numvar, 404
 iinfitem.opt_numcon, 404
 iinfitem.mio_user_obj_cut, 404
 iinfitem.mio_total_num_cuts, 404
 iinfitem.mio_relgap_satisfied, 404
 iinfitem.mio_presolved_numvar, 404
 iinfitem.mio_presolved_numrqcones, 403
 iinfitem.mio_presolved_numqcones, 403
 iinfitem.mio_presolved_numppowcones, 403
 iinfitem.mio_presolved_numpexpcones, 403
 iinfitem.mio_presolved_numintconevar, 403
 iinfitem.mio_presolved_numint, 403
 iinfitem.mio_presolved_numdpowcones, 403
 iinfitem.mio_presolved_numdexpcones, 403
 iinfitem.mio_presolved_numcontconevar, 403
 iinfitem.mio_presolved_numcont, 403
 iinfitem.mio_presolved_numconevar, 403
 iinfitem.mio_presolved_numcone, 403
 iinfitem.mio_presolved_numcon, 403
 iinfitem.mio_presolved_numbinconevar, 403
 iinfitem.mio_presolved_numbin, 403
 iinfitem.mio_obj_bound_defined, 403
 iinfitem.mio_numvar, 403
 iinfitem.mio_numrqcones, 403
 iinfitem.mio_numqcones, 403
 iinfitem.mio_numppowcones, 403
 iinfitem.mio_numpexpcones, 403
 iinfitem.mio_numintconevar, 403
 iinfitem.mio_numint, 403
 iinfitem.mio_numdpowcones, 403
 iinfitem.mio_numdexpcones, 402
 iinfitem.mio_numcontconevar, 402
 iinfitem.mio_numcont, 402
 iinfitem.mio_numconevar, 402
 iinfitem.mio_numcone, 402
 iinfitem.mio_numcon, 402
 iinfitem.mio_numbinconevar, 402
 iinfitem.mio_numbin, 402
 iinfitem.mio_num_repeated_presolve, 402
 iinfitem.mio_num_relax, 402
 iinfitem.mio_num_knapsack_cover_cuts, 402
 iinfitem.mio_num_int_solutions, 402
 iinfitem.mio_num_implied_bound_cuts, 402
 iinfitem.mio_num_gomory_cuts, 402
 iinfitem.mio_num_cmir_cuts, 402
 iinfitem.mio_num_clique_cuts, 402
 iinfitem.mio_num_branch, 402
 iinfitem.mio_num_active_nodes, 402
 iinfitem.mio_node_depth, 402
 iinfitem.mio_construct_solution, 402
 iinfitem.mio_clique_table_size, 402
 iinfitem.mio_absgap_satisfied, 402
 iinfitem.intpnt_solve_dual, 402
 iinfitem.intpnt_num_threads, 401
 iinfitem.intpnt_iter, 401
 iinfitem.intpnt_factor_dim_dense, 401
 iinfitem.ana_pro_num_var_up, 401
 iinfitem.ana_pro_num_var_ra, 401
 iinfitem.ana_pro_num_var_lo, 401
 iinfitem.ana_pro_num_var_int, 401
 iinfitem.ana_pro_num_var_fr, 401
 iinfitem.ana_pro_num_var_eq, 401
 iinfitem.ana_pro_num_var_cont, 401
 iinfitem.ana_pro_num_var_bin, 401
 iinfitem.ana_pro_num_var, 401
 iinfitem.ana_pro_num_con_up, 401
 iinfitem.ana_pro_num_con_ra, 401
 iinfitem.ana_pro_num_con_lo, 401
 iinfitem.ana_pro_num_con_fr, 401

- iinfitem.ana_pro_num_con_eq, 401
- iinfitem.ana_pro_num_con, 401
- inftype, 405
- inftype.lint_type, 405
- inftype.int_type, 405
- inftype.dou_type, 405
- intpnthotstart, 389
- intpnthotstart.primal_dual, 389
- intpnthotstart.primal, 389
- intpnthotstart.none, 389
- intpnthotstart.dual, 389
- iomode, 405
- iomode.write, 405
- iomode.readwrite, 405
- iomode.read, 405
- iparam, 335
- liinfitem, 400
- liinfitem.rd_numqnz, 401
- liinfitem.rd_numanz, 401
- liinfitem.mio_simplex_iter, 401
- liinfitem.mio_presolved_anz, 401
- liinfitem.mio_intpnt_iter, 400
- liinfitem.mio_anz, 400
- liinfitem.intpnt_factor_num_nz, 400
- liinfitem.bi_primal_iter, 400
- liinfitem.bi_dual_iter, 400
- liinfitem.bi_clean_primal_iter, 400
- liinfitem.bi_clean_primal_deg_iter, 400
- liinfitem.bi_clean_dual_iter, 400
- liinfitem.bi_clean_dual_deg_iter, 400
- mark, 388
- mark.up, 388
- mark.lo, 388
- miocontsoltype, 406
- miocontsoltype.root, 406
- miocontsoltype.none, 406
- miocontsoltype.itg_rel, 406
- miocontsoltype.itg, 406
- miomode, 406
- miomode.satisfied, 406
- miomode.ignored, 406
- mionodeseltype, 406
- mionodeseltype.pseudo, 406
- mionodeseltype.free, 406
- mionodeseltype.first, 406
- mionodeseltype.best, 406
- mpsformat, 406
- mpsformat.strict, 406
- mpsformat.relaxed, 406
- mpsformat.free, 406
- mpsformat.cplex, 406
- nametype, 395
- nametype.mps, 395
- nametype.lp, 395
- nametype.gen, 395
- objsense, 407
- objsense.minimize, 407
- objsense.maximize, 407
- onoffkey, 407
- onoffkey.on, 407
- onoffkey.off, 407
- optimizertype, 407
- optimizertype.primal_simplex, 407
- optimizertype.mixed_int, 407
- optimizertype.intpnt, 407
- optimizertype.free_simplex, 407
- optimizertype.free, 407
- optimizertype.dual_simplex, 407
- optimizertype.conic, 407
- orderingtype, 407
- orderingtype.try_graphpar, 407
- orderingtype.none, 407
- orderingtype.free, 407
- orderingtype.force_graphpar, 407
- orderingtype.experimental, 407
- orderingtype.appminloc, 407
- parametertype, 407
- parametertype.str_type, 408
- parametertype.invalid_type, 408
- parametertype.int_type, 408
- parametertype.dou_type, 408
- presolvemode, 407
- presolvemode.on, 407
- presolvemode.off, 407
- presolvemode.free, 407
- problemitem, 408
- problemitem.var, 408
- problemitem.cone, 408
- problemitem.con, 408
- problemttype, 408
- problemttype.qo, 408
- problemttype.qcqp, 408
- problemttype.mixed, 408
- problemttype.lo, 408
- problemttype.conic, 408
- prosta, 408
- prosta.unknown, 408
- prosta.prim_infeas_or_unbounded, 408
- prosta.prim_infeas, 408
- prosta.prim_feas, 408
- prosta.prim_and_dual_infeas, 408
- prosta.prim_and_dual_feas, 408
- prosta.ill_posed, 408
- prosta.dual_infeas, 408
- prosta.dual_feas, 408
- purify, 390
- purify.primal_dual, 390
- purify.primal, 390
- purify.none, 390
- purify.dual, 390
- purify.auto, 390
- rescode, 370
- rescodetype, 409
- rescodetype.wrn, 409
- rescodetype.unk, 409
- rescodetype.trm, 409

- rescodetype.ok, 409
- rescodetype.err, 409
- scalingmethod, 409
- scalingmethod.pow2, 409
- scalingmethod.free, 409
- scalingtype, 409
- scalingtype.none, 409
- scalingtype.moderate, 409
- scalingtype.free, 409
- scalingtype.aggressive, 409
- scopr, 395
- scopr.sqrt, 395
- scopr.pow, 395
- scopr.log, 395
- scopr.exp, 395
- scopr.ent, 395
- sensitivitytype, 409
- sensitivitytype.basis, 409
- simdegen, 388
- simdegen.none, 388
- simdegen.moderate, 389
- simdegen.minimum, 389
- simdegen.free, 388
- simdegen.aggressive, 388
- simdupvec, 389
- simdupvec.on, 389
- simdupvec.off, 389
- simdupvec.free, 389
- simhotstart, 389
- simhotstart.status_keys, 389
- simhotstart.none, 389
- simhotstart.free, 389
- simreform, 389
- simreform.on, 389
- simreform.off, 389
- simreform.free, 389
- simreform.aggressive, 389
- simseltype, 409
- simseltype.se, 409
- simseltype.partial, 410
- simseltype.full, 409
- simseltype.free, 409
- simseltype.devex, 409
- simseltype.ase, 409
- solitem, 410
- solitem.y, 410
- solitem.xx, 410
- solitem.xc, 410
- solitem.sux, 410
- solitem.suc, 410
- solitem.snx, 410
- solitem.slx, 410
- solitem.slc, 410
- solsta, 410
- solsta.unknown, 410
- solsta.prim_infeas_cer, 410
- solsta.prim_illposed_cer, 410
- solsta.prim_feas, 410

- solsta.prim_and_dual_feas, 410
- solsta.optimal, 410
- solsta.integer_optimal, 410
- solsta.dual_infeas_cer, 410
- solsta.dual_illposed_cer, 410
- solsta.dual_feas, 410
- solttype, 410
- solttype.itr, 410
- solttype.itg, 410
- solttype.bas, 410
- solveform, 411
- solveform.primal, 411
- solveform.free, 411
- solveform.dual, 411
- sparam, 366
- stakey, 411
- stakey.upr, 411
- stakey.unk, 411
- stakey.supbas, 411
- stakey.low, 411
- stakey.inf, 411
- stakey.fix, 411
- stakey.bas, 411
- startpointtype, 411
- startpointtype.satisfy_bounds, 411
- startpointtype.guess, 411
- startpointtype.free, 411
- startpointtype.constant, 411
- streamtype, 411
- streamtype.wrn, 411
- streamtype.msg, 411
- streamtype.log, 411
- streamtype.err, 411
- symmatttype, 395
- symmatttype.sparse, 395
- transpose, 389
- transpose.yes, 389
- transpose.no, 389
- uplo, 389
- uplo.up, 389
- uplo.lo, 389
- value, 411
- value.max_str_len, 411
- value.license_buffer_length, 412
- variabletype, 412
- variabletype.type_int, 412
- variabletype.type_cont, 412
- xmlwriteroutputtype, 409
- xmlwriteroutputtype.row, 409
- xmlwriteroutputtype.col, 409

Exceptions

- ArrayLengthException, 313
- Error, 313
- Exception, 313
- MosekException, 313
- NullArrayException, 313
- Warning, 313

Parameters

Double parameters, 325

dparam.ana_sol_infeas_tol, 325

dparam.basis_rel_tol_s, 325

dparam.basis_tol_s, 325

dparam.basis_tol_x, 325

dparam.check_convexity_rel_tol, 325

dparam.data_sym_mat_tol, 326

dparam.data_sym_mat_tol_huge, 326

dparam.data_sym_mat_tol_large, 326

dparam.data_tol_aij_huge, 326

dparam.data_tol_aij_large, 326

dparam.data_tol_bound_inf, 327

dparam.data_tol_bound_wrn, 327

dparam.data_tol_c_huge, 327

dparam.data_tol_cj_large, 327

dparam.data_tol_qij, 327

dparam.data_tol_x, 327

dparam.intpnt_co_tol_dfeas, 327

dparam.intpnt_co_tol_infeas, 328

dparam.intpnt_co_tol_mu_red, 328

dparam.intpnt_co_tol_near_rel, 328

dparam.intpnt_co_tol_pfeas, 328

dparam.intpnt_co_tol_rel_gap, 328

dparam.intpnt_qo_tol_dfeas, 329

dparam.intpnt_qo_tol_infeas, 329

dparam.intpnt_qo_tol_mu_red, 329

dparam.intpnt_qo_tol_near_rel, 329

dparam.intpnt_qo_tol_pfeas, 329

dparam.intpnt_qo_tol_rel_gap, 329

dparam.intpnt_tol_dfeas, 330

dparam.intpnt_tol_dsafe, 330

dparam.intpnt_tol_infeas, 330

dparam.intpnt_tol_mu_red, 330

dparam.intpnt_tol_path, 330

dparam.intpnt_tol_pfeas, 330

dparam.intpnt_tol_psafe, 331

dparam.intpnt_tol_rel_gap, 331

dparam.intpnt_tol_rel_step, 331

dparam.intpnt_tol_step_size, 331

dparam.lower_obj_cut, 331

dparam.lower_obj_cut_finite_trh, 331

dparam.mio_max_time, 332

dparam.mio_rel_gap_const, 332

dparam.mio_tol_abs_gap, 332

dparam.mio_tol_abs_relax_int, 332

dparam.mio_tol_feas, 332

dparam.mio_tol_rel_dual_bound_improvement,
332

dparam.mio_tol_rel_gap, 333

dparam.optimizer_max_time, 333

dparam.presolve_tol_abs_lindep, 333

dparam.presolve_tol_aij, 333

dparam.presolve_tol_rel_lindep, 333

dparam.presolve_tol_s, 333

dparam.presolve_tol_x, 334

dparam.qcqp_reformulate_rel_drop_tol, 334

dparam.semidefinite_tol_approx, 334

dparam.sim_lu_tol_rel_piv, 334

dparam.simplex_abs_tol_piv, 334

dparam.upper_obj_cut, 334

dparam.upper_obj_cut_finite_trh, 335

Integer parameters, 335

iparam.ana_sol_basis, 335

iparam.ana_sol_print_violated, 335

iparam.auto_sort_a_before_opt, 335

iparam.auto_update_sol_info, 335

iparam.basis_solve_use_plus_one, 336

iparam.bi_clean_optimizer, 336

iparam.bi_ignore_max_iter, 336

iparam.bi_ignore_num_error, 336

iparam.bi_max_iterations, 336

iparam.cache_license, 336

iparam.check_convexity, 337

iparam.compress_statfile, 337

iparam.infeas_generic_names, 337

iparam.infeas_prefer_primal, 337

iparam.infeas_report_auto, 337

iparam.infeas_report_level, 338

iparam.intpnt_basis, 338

iparam.intpnt_diff_step, 338

iparam.intpnt_hotstart, 338

iparam.intpnt_max_iterations, 338

iparam.intpnt_max_num_cor, 338

iparam.intpnt_max_num_refinement_steps, 339

iparam.intpnt_multi_thread, 339

iparam.intpnt_off_col_trh, 339

iparam.intpnt_order_gp_num_seeds, 339

iparam.intpnt_order_method, 339

iparam.intpnt_purify, 340

iparam.intpnt_regularization_use, 340

iparam.intpnt_scaling, 340

iparam.intpnt_solve_form, 340

iparam.intpnt_starting_point, 340

iparam.license_debug, 340

iparam.license_pause_time, 341

iparam.license_suppress_expire_wrns, 341

iparam.license_trh_expiry_wrn, 341

iparam.license_wait, 341

iparam.log, 341

iparam.log_ana_pro, 341

iparam.log_bi, 342

iparam.log_bi_freq, 342

iparam.log_check_convexity, 342

iparam.log_cut_second_opt, 342

iparam.log_expand, 342

iparam.log_feas_repair, 343

iparam.log_file, 343

iparam.log_include_summary, 343

iparam.log_infeas_ana, 343

iparam.log_intpnt, 343

iparam.log_local_info, 343

iparam.log_mio, 344

iparam.log_mio_freq, 344

iparam.log_order, 344

iparam.log_presolve, 344

iparam.log_response, 344
 iparam.log_sensitivity, 344
 iparam.log_sensitivity_opt, 345
 iparam.log_sim, 345
 iparam.log_sim_freq, 345
 iparam.log_sim_minor, 345
 iparam.log_storage, 345
 iparam.max_num_warnings, 345
 iparam.mio_branch_dir, 346
 iparam.mio_conic_outer_approximation, 346
 iparam.mio_cut_clique, 346
 iparam.mio_cut_cmir, 346
 iparam.mio_cut_gmi, 346
 iparam.mio_cut_implied_bound, 346
 iparam.mio_cut_knapsack_cover, 347
 iparam.mio_cut_selection_level, 347
 iparam.mio_feaspump_level, 347
 iparam.mio_heuristic_level, 347
 iparam.mio_max_num_branches, 347
 iparam.mio_max_num_relaxs, 348
 iparam.mio_max_num_root_cut_rounds, 348
 iparam.mio_max_num_solutions, 348
 iparam.mio_mode, 348
 iparam.mio_node_optimizer, 348
 iparam.mio_node_selection, 348
 iparam.mio_perspective_reformulate, 349
 iparam.mio_probing_level, 349
 iparam.mio_propagate_objective_constraint, 349
 iparam.mio_rins_max_nodes, 349
 iparam.mio_root_optimizer, 349
 iparam.mio_root_repeat_presolve_level, 350
 iparam.mio_seed, 350
 iparam.mio_vb_detection_level, 350
 iparam.mt_spincount, 350
 iparam.num_threads, 350
 iparam.opf_write_header, 351
 iparam.opf_write_hints, 351
 iparam.opf_write_line_length, 351
 iparam.opf_write_parameters, 351
 iparam.opf_write_problem, 351
 iparam.opf_write_sol_bas, 351
 iparam.opf_write_sol_itg, 352
 iparam.opf_write_sol_itr, 352
 iparam.opf_write_solutions, 352
 iparam.optimizer, 352
 iparam.param_read_case_name, 352
 iparam.param_read_ign_error, 352
 iparam.presolve_eliminator_max_fill, 353
 iparam.presolve_eliminator_max_num_tries, 353
 iparam.presolve_level, 353
 iparam.presolve_lindep_abs_work_trh, 353
 iparam.presolve_lindep_rel_work_trh, 353
 iparam.presolve_lindep_use, 353
 iparam.presolve_max_num_pass, 354
 iparam.presolve_max_num_reductions, 354
 iparam.presolve_use, 354
 iparam.primal_repair_optimizer, 354
 iparam.ptf_write_transform, 354
 iparam.read_debug, 354
 iparam.read_keep_free_con, 355
 iparam.read_lp_drop_new_vars_in_bou, 355
 iparam.read_lp_quoted_names, 355
 iparam.read_mps_format, 355
 iparam.read_mps_width, 355
 iparam.read_task_ignore_param, 355
 iparam.remove_unused_solutions, 356
 iparam.sensitivity_all, 356
 iparam.sensitivity_optimizer, 356
 iparam.sensitivity_type, 356
 iparam.sim_basis_factor_use, 356
 iparam.sim_degen, 357
 iparam.sim_dual_crash, 357
 iparam.sim_dual_phaseone_method, 357
 iparam.sim_dual_restrict_selection, 357
 iparam.sim_dual_selection, 357
 iparam.sim_exploit_dupvec, 357
 iparam.sim_hotstart, 358
 iparam.sim_hotstart_lu, 358
 iparam.sim_max_iterations, 358
 iparam.sim_max_num_setbacks, 358
 iparam.sim_non_singular, 358
 iparam.sim_primal_crash, 358
 iparam.sim_primal_phaseone_method, 359
 iparam.sim_primal_restrict_selection, 359
 iparam.sim_primal_selection, 359
 iparam.sim_refactor_freq, 359
 iparam.sim_reformulation, 359
 iparam.sim_save_lu, 360
 iparam.sim_scaling, 360
 iparam.sim_scaling_method, 360
 iparam.sim_seed, 360
 iparam.sim_solve_form, 360
 iparam.sim_stability_priority, 360
 iparam.sim_switch_optimizer, 360
 iparam.sol_filter_keep_basic, 361
 iparam.sol_filter_keep_ranged, 361
 iparam.sol_read_name_width, 361
 iparam.sol_read_width, 361
 iparam.solution_callback, 361
 iparam.timing_level, 362
 iparam.write_bas_constraints, 362
 iparam.write_bas_head, 362
 iparam.write_bas_variables, 362
 iparam.write_compression, 362
 iparam.write_data_param, 362
 iparam.write_free_con, 363
 iparam.write_generic_names, 363
 iparam.write_generic_names_io, 363
 iparam.write_ignore_incompatible_items, 363
 iparam.write_int_constraints, 363
 iparam.write_int_head, 363
 iparam.write_int_variables, 364
 iparam.write_lp_full_obj, 364
 iparam.write_lp_line_width, 364

- iparam.write_lp_quoted_names, 364
- iparam.write_lp_strict_format, 364
- iparam.write_lp_terms_per_line, 364
- iparam.write_mps_format, 365
- iparam.write_mps_int, 365
- iparam.write_precision, 365
- iparam.write_sol_barvariables, 365
- iparam.write_sol_constraints, 365
- iparam.write_sol_head, 365
- iparam.write_sol_ignore_invalid_names, 366
- iparam.write_sol_variables, 366
- iparam.write_task_inc_sol, 366
- iparam.write_xml_mode, 366
- String parameters, 366
- sparam.bas_sol_file_name, 366
- sparam.data_file_name, 366
- sparam.debug_file_name, 367
- sparam.int_sol_file_name, 367
- sparam.itr_sol_file_name, 367
- sparam.mio_debug_string, 367
- sparam.param_comment_sign, 367
- sparam.param_read_file_name, 367
- sparam.param_write_file_name, 367
- sparam.read_mps_bou_name, 368
- sparam.read_mps_obj_name, 368
- sparam.read_mps_ran_name, 368
- sparam.read_mps_rhs_name, 368
- sparam.remote_access_token, 368
- sparam.sensitivity_file_name, 368
- sparam.sensitivity_res_file_name, 368
- sparam.sol_filter_xc_low, 369
- sparam.sol_filter_xc_upr, 369
- sparam.sol_filter_xx_low, 369
- sparam.sol_filter_xx_upr, 369
- sparam.stat_file_name, 369
- sparam.stat_key, 369
- sparam.stat_name, 369
- sparam.write_lp_gen_var_name, 370

Response codes

- Termination, 370
- rescode.ok, 370
- rescode.trm_internal, 371
- rescode.trm_internal_stop, 371
- rescode.trm_max_iterations, 370
- rescode.trm_max_num_setbacks, 371
- rescode.trm_max_time, 370
- rescode.trm_mio_num_branches, 370
- rescode.trm_mio_num_relaxs, 370
- rescode.trm_num_max_num_int_solutions, 370
- rescode.trm_numerical_problem, 371
- rescode.trm_objective_range, 370
- rescode.trm_stall, 370
- rescode.trm_user_callback, 370
- Warnings, 371
- rescode.wrn_ana_almost_int_bounds, 373
- rescode.wrn_ana_c_zero, 373
- rescode.wrn_ana_close_bounds, 373

- rescode.wrn_ana_empty_cols, 373
- rescode.wrn_ana_large_bounds, 373
- rescode.wrn_dropped_nz_qobj, 371
- rescode.wrn_duplicate_barvariable_names, 372
- rescode.wrn_duplicate_cone_names, 372
- rescode.wrn_duplicate_constraint_names, 372
- rescode.wrn_duplicate_variable_names, 372
- rescode.wrn_eliminator_space, 372
- rescode.wrn_empty_name, 372
- rescode.wrn_exp_cones_with_variables_fixed_at_zero, 373
- rescode.wrn_ignore_integer, 371
- rescode.wrn_incomplete_linear_dependency_check, 372
- rescode.wrn_large_aij, 371
- rescode.wrn_large_bound, 371
- rescode.wrn_large_cj, 371
- rescode.wrn_large_con_fx, 371
- rescode.wrn_large_lo_bound, 371
- rescode.wrn_large_up_bound, 371
- rescode.wrn_license_expire, 372
- rescode.wrn_license_feature_expire, 372
- rescode.wrn_license_server, 372
- rescode.wrn_lp_drop_variable, 371
- rescode.wrn_lp_old_quad_format, 371
- rescode.wrn_mio_infeasible_final, 371
- rescode.wrn_mps_split_bou_vector, 371
- rescode.wrn_mps_split_ran_vector, 371
- rescode.wrn_mps_split_rhs_vector, 371
- rescode.wrn_name_max_len, 371
- rescode.wrn_no_dualizer, 373
- rescode.wrn_no_global_optimizer, 371
- rescode.wrn_nz_in_upr_tri, 371
- rescode.wrn_open_param_file, 371
- rescode.wrn_param_ignored_cmio, 372
- rescode.wrn_param_name_dou, 372
- rescode.wrn_param_name_int, 372
- rescode.wrn_param_name_str, 372
- rescode.wrn_param_str_value, 372
- rescode.wrn_pow_cones_with_root_fixed_at_zero, 373
- rescode.wrn_presolve_outofspace, 372
- rescode.wrn_quad_cones_with_root_fixed_at_zero, 373
- rescode.wrn_rquad_cones_with_root_fixed_at_zero, 373
- rescode.wrn_sol_file_ignored_con, 372
- rescode.wrn_sol_file_ignored_var, 372
- rescode.wrn_sol_filter, 371
- rescode.wrn_spar_max_len, 371
- rescode.wrn_sym_mat_large, 373
- rescode.wrn_too_few_basis_vars, 372
- rescode.wrn_too_many_basis_vars, 372
- rescode.wrn_undef_sol_file_name, 372
- rescode.wrn_using_generic_names, 372
- rescode.wrn_write_changed_names, 372
- rescode.wrn_write_discarded_cfix, 372

rescode.wrn_zero_aij, 371
 rescode.wrn_zeros_in_sparse_col, 372
 rescode.wrn_zeros_in_sparse_row, 372
 Errors, 373
 rescode.err_ad_invalid_codelist, 384
 rescode.err_api_array_too_small, 384
 rescode.err_api_cb_connect, 384
 rescode.err_api_fatal_error, 384
 rescode.err_api_internal, 384
 rescode.err_appending_too_big_cone, 381
 rescode.err_arg_is_too_large, 378
 rescode.err_arg_is_too_small, 378
 rescode.err_argument_dimension, 378
 rescode.err_argument_is_too_large, 385
 rescode.err_argument_lenneq, 377
 rescode.err_argument_perm_array, 380
 rescode.err_argument_type, 377
 rescode.err_bar_var_dim, 385
 rescode.err_basis, 379
 rescode.err_basis_factor, 383
 rescode.err_basis_singular, 383
 rescode.err_blank_name, 375
 rescode.err_cbf_duplicate_acoord, 386
 rescode.err_cbf_duplicate_bcoord, 386
 rescode.err_cbf_duplicate_con, 386
 rescode.err_cbf_duplicate_int, 386
 rescode.err_cbf_duplicate_obj, 386
 rescode.err_cbf_duplicate_objacoord, 386
 rescode.err_cbf_duplicate_pow_cones, 387
 rescode.err_cbf_duplicate_pow_star_cones, 387
 rescode.err_cbf_duplicate_psdvar, 387
 rescode.err_cbf_duplicate_var, 386
 rescode.err_cbf_invalid_con_type, 386
 rescode.err_cbf_invalid_dimension_of_cones, 387
 rescode.err_cbf_invalid_domain_dimension, 386
 rescode.err_cbf_invalid_exp_dimension, 387
 rescode.err_cbf_invalid_int_index, 387
 rescode.err_cbf_invalid_number_of_cones, 387
 rescode.err_cbf_invalid_power, 387
 rescode.err_cbf_invalid_power_cone_index, 387
 rescode.err_cbf_invalid_power_star_cone_index, 387
 rescode.err_cbf_invalid_psdvar_dimension, 387
 rescode.err_cbf_invalid_var_type, 386
 rescode.err_cbf_no_variables, 386
 rescode.err_cbf_no_version_specified, 386
 rescode.err_cbf_obj_sense, 386
 rescode.err_cbf_parse, 386
 rescode.err_cbf_power_cone_is_too_long, 387
 rescode.err_cbf_power_cone_mismatch, 387
 rescode.err_cbf_power_star_cone_mismatch, 387
 rescode.err_cbf_syntax, 386
 rescode.err_cbf_too_few_constraints, 387
 rescode.err_cbf_too_few_ints, 387
 rescode.err_cbf_too_few_psdvar, 387
 rescode.err_cbf_too_few_variables, 386
 rescode.err_cbf_too_many_constraints, 386
 rescode.err_cbf_too_many_ints, 387
 rescode.err_cbf_too_many_variables, 386
 rescode.err_cbf_unhandled_power_cone_type, 387
 rescode.err_cbf_unhandled_power_star_cone_type, 387
 rescode.err_cbf_unsupported, 387
 rescode.err_con_q_not_nsd, 380
 rescode.err_con_q_not_psd, 380
 rescode.err_cone_index, 380
 rescode.err_cone_overlap, 381
 rescode.err_cone_overlap_append, 381
 rescode.err_cone_parameter, 381
 rescode.err_cone_rep_var, 381
 rescode.err_cone_size, 380
 rescode.err_cone_type, 381
 rescode.err_cone_type_str, 381
 rescode.err_data_file_ext, 375
 rescode.err_dup_name, 375
 rescode.err_duplicate_aij, 381
 rescode.err_duplicate_barvariable_names, 385
 rescode.err_duplicate_cone_names, 385
 rescode.err_duplicate_constraint_names, 385
 rescode.err_duplicate_variable_names, 385
 rescode.err_end_of_file, 375
 rescode.err_factor, 383
 rescode.err_feasrepair_cannot_relax, 383
 rescode.err_feasrepair_inconsistent_bound, 383
 rescode.err_feasrepair_solving_relaxed, 383
 rescode.err_file_license, 373
 rescode.err_file_open, 375
 rescode.err_file_read, 375
 rescode.err_file_write, 375
 rescode.err_final_solution, 382
 rescode.err_first, 382
 rescode.err_firsti, 380
 rescode.err_firstj, 380
 rescode.err_fixed_bound_values, 382
 rescode.err_flexlm, 374
 rescode.err_format_string, 375
 rescode.err_global_inv_conic_problem, 382
 rescode.err_huge_aij, 381
 rescode.err_huge_c, 381
 rescode.err_identical_tasks, 384
 rescode.err_in_argument, 378
 rescode.err_index, 378
 rescode.err_index_arr_is_too_large, 378
 rescode.err_index_arr_is_too_small, 378
 rescode.err_index_is_too_large, 378
 rescode.err_index_is_too_small, 378

rescode.err_inf_dou_index, 378
 rescode.err_inf_dou_name, 378
 rescode.err_inf_int_index, 378
 rescode.err_inf_int_name, 378
 rescode.err_inf_lint_index, 378
 rescode.err_inf_lint_name, 378
 rescode.err_inf_type, 378
 rescode.err_infeas_undefined, 385
 rescode.err_infinite_bound, 381
 rescode.err_int64_to_int32_cast, 384
 rescode.err_internal, 384
 rescode.err_internal_test_failed, 384
 rescode.err_inv_aptre, 379
 rescode.err_inv_bk, 379
 rescode.err_inv_bkc, 379
 rescode.err_inv_bkx, 379
 rescode.err_inv_cone_type, 380
 rescode.err_inv_cone_type_str, 380
 rescode.err_inv_marki, 383
 rescode.err_inv_markj, 383
 rescode.err_inv_name_item, 380
 rescode.err_inv_numi, 383
 rescode.err_inv_numj, 383
 rescode.err_inv_optimizer, 382
 rescode.err_inv_problem, 382
 rescode.err_inv_qcon_subi, 381
 rescode.err_inv_qcon_subj, 381
 rescode.err_inv_qcon_subk, 381
 rescode.err_inv_qcon_val, 381
 rescode.err_inv_qobj_subi, 381
 rescode.err_inv_qobj_subj, 381
 rescode.err_inv_qobj_val, 381
 rescode.err_inv_sk, 380
 rescode.err_inv_sk_str, 380
 rescode.err_inv_skc, 379
 rescode.err_inv_skn, 380
 rescode.err_inv_skn, 380
 rescode.err_inv_skn, 380
 rescode.err_inv_var_type, 379
 rescode.err_invalid_aij, 382
 rescode.err_invalid_ampl_stub, 384
 rescode.err_invalid_barvar_name, 375
 rescode.err_invalid_cj, 382
 rescode.err_invalid_compression, 383
 rescode.err_invalid_con_name, 375
 rescode.err_invalid_cone_name, 375
 rescode.err_invalid_file_format_for_cfix, 385
 rescode.err_invalid_file_format_for_cones, 385
 rescode.err_invalid_file_format_for_free_constraints, 385
 rescode.err_invalid_file_format_for_nonlinear_constraints, 385
 rescode.err_invalid_file_format_for_ranged_constraints, 385
 rescode.err_invalid_file_format_for_sym_mat, 385
 rescode.err_invalid_file_name, 375
 rescode.err_invalid_format_type, 380
 rescode.err_invalid_idx, 379
 rescode.err_invalid_iomode, 383
 rescode.err_invalid_max_num, 379
 rescode.err_invalid_name_in_sol_file, 377
 rescode.err_invalid_obj_name, 375
 rescode.err_invalid_objective_sense, 382
 rescode.err_invalid_problem_type, 385
 rescode.err_invalid_sol_file_name, 375
 rescode.err_invalid_stream, 375
 rescode.err_invalid_surplus, 380
 rescode.err_invalid_sym_mat_dim, 385
 rescode.err_invalid_task, 375
 rescode.err_invalid_utf8, 384
 rescode.err_invalid_var_name, 375
 rescode.err_invalid_wchar, 384
 rescode.err_invalid_whichsol, 378
 rescode.err_json_data, 377
 rescode.err_json_format, 377
 rescode.err_json_missing_data, 377
 rescode.err_json_number_overflow, 377
 rescode.err_json_string, 377
 rescode.err_json_syntax, 377
 rescode.err_last, 382
 rescode.err_lasti, 380
 rescode.err_lastj, 380
 rescode.err_lau_arg_k, 386
 rescode.err_lau_arg_m, 386
 rescode.err_lau_arg_n, 386
 rescode.err_lau_arg_trans, 386
 rescode.err_lau_arg_transa, 386
 rescode.err_lau_arg_transb, 386
 rescode.err_lau_arg_uplo, 386
 rescode.err_lau_invalid_lower_triangular_matrix, 386
 rescode.err_lau_invalid_sparse_symmetric_matrix, 386
 rescode.err_lau_not_positive_definite, 385
 rescode.err_lau_singular_matrix, 385
 rescode.err_lau_unknown, 386
 rescode.err_license, 373
 rescode.err_license_cannot_allocate, 374
 rescode.err_license_cannot_connect, 374
 rescode.err_license_expired, 373
 rescode.err_license_feature, 374
 rescode.err_license_invalid_hostid, 374
 rescode.err_license_max, 374
 rescode.err_license_moseklm_daemon, 374
 rescode.err_license_no_server_line, 374
 rescode.err_license_no_server_support, 374
 rescode.err_license_server, 374
 rescode.err_license_server_version, 374
 rescode.err_license_version, 373
 rescode.err_link_file_dll, 374
 rescode.err_living_tasks, 375
 rescode.err_lower_bound_is_a_nan, 381
 rescode.err_lp_dup_slack_name, 376
 rescode.err_lp_empty, 376

rescode.err_lp_file_format, 377
 rescode.err_lp_format, 377
 rescode.err_lp_free_constraint, 377
 rescode.err_lp_incompatible, 376
 rescode.err_lp_invalid_con_name, 377
 rescode.err_lp_invalid_var_name, 377
 rescode.err_lp_write_conic_problem, 377
 rescode.err_lp_write_geco_problem, 377
 rescode.err_lu_max_num_tries, 383
 rescode.err_max_len_is_too_small, 380
 rescode.err_maxnumbarvar, 379
 rescode.err_maxnumcon, 379
 rescode.err_maxnumcone, 381
 rescode.err_maxnumqnz, 379
 rescode.err_maxnumvar, 379
 rescode.err_mio_internal, 385
 rescode.err_mio_invalid_node_optimizer, 387
 rescode.err_mio_invalid_root_optimizer, 387
 rescode.err_mio_no_optimizer, 382
 rescode.err_missing_license_file, 373
 rescode.err_mixed_conic_and_nl, 382
 rescode.err_mps_cone_overlap, 376
 rescode.err_mps_cone_repeat, 376
 rescode.err_mps_cone_type, 376
 rescode.err_mps_duplicate_q_element, 376
 rescode.err_mps_file, 375
 rescode.err_mps_inv_bound_key, 376
 rescode.err_mps_inv_con_key, 376
 rescode.err_mps_inv_field, 375
 rescode.err_mps_inv_marker, 376
 rescode.err_mps_inv_sec_name, 376
 rescode.err_mps_inv_sec_order, 376
 rescode.err_mps_invalid_obj_name, 376
 rescode.err_mps_invalid_objsense, 376
 rescode.err_mps_mul_con_name, 376
 rescode.err_mps_mul_csec, 376
 rescode.err_mps_mul_qobj, 376
 rescode.err_mps_mul_qsec, 376
 rescode.err_mps_no_objective, 376
 rescode.err_mps_non_symmetric_q, 376
 rescode.err_mps_null_con_name, 376
 rescode.err_mps_null_var_name, 376
 rescode.err_mps_splitting_var, 376
 rescode.err_mps_tab_in_field2, 376
 rescode.err_mps_tab_in_field3, 376
 rescode.err_mps_tab_in_field5, 376
 rescode.err_mps_undef_con_name, 376
 rescode.err_mps_undef_var_name, 376
 rescode.err_mul_a_element, 379
 rescode.err_name_is_null, 383
 rescode.err_name_max_len, 383
 rescode.err_nan_in_blc, 382
 rescode.err_nan_in_blx, 382
 rescode.err_nan_in_buc, 382
 rescode.err_nan_in_bux, 382
 rescode.err_nan_in_c, 382
 rescode.err_nan_in_double_data, 382
 rescode.err_negative_append, 383
 rescode.err_negative_surplus, 383
 rescode.err_newer_dll, 374
 rescode.err_noBars_for_solution, 385
 rescode.err_no_barx_for_solution, 385
 rescode.err_no_basis_sol, 383
 rescode.err_no_dual_for_itg_sol, 384
 rescode.err_no_dual_infeas_cer, 383
 rescode.err_no_init_env, 375
 rescode.err_no_optimizer_var_type, 382
 rescode.err_no_primal_infeas_cer, 383
 rescode.err_no_snx_for_bas_sol, 384
 rescode.err_no_solution_in_callback, 383
 rescode.err_non_unique_array, 385
 rescode.err_nonconvex, 380
 rescode.err_nonlinear_equality, 380
 rescode.err_nonlinear_ranged, 380
 rescode.err_null_env, 375
 rescode.err_null_pointer, 375
 rescode.err_null_task, 375
 rescode.err_num_arguments, 377
 rescode.err_numconlim, 379
 rescode.err_numvarlim, 379
 rescode.err_obj_q_not_nsd, 380
 rescode.err_obj_q_not_psd, 380
 rescode.err_objective_range, 379
 rescode.err_older_dll, 374
 rescode.err_opf_format, 377
 rescode.err_opf_new_variable, 377
 rescode.err_opf_premature_eof, 377
 rescode.err_optimizer_license, 374
 rescode.err_overflow, 383
 rescode.err_param_index, 378
 rescode.err_param_is_too_large, 378
 rescode.err_param_is_too_small, 378
 rescode.err_param_name, 378
 rescode.err_param_name_dou, 378
 rescode.err_param_name_int, 378
 rescode.err_param_name_str, 378
 rescode.err_param_type, 378
 rescode.err_param_value_str, 378
 rescode.err_platform_not_licensed, 374
 rescode.err_postsolve, 383
 rescode.err_pro_item, 380
 rescode.err_prob_license, 373
 rescode.err_ptf_format, 377
 rescode.err_qcon_subi_too_large, 381
 rescode.err_qcon_subi_too_small, 381
 rescode.err_qcon_upper_triangle, 382
 rescode.err_qobj_upper_triangle, 381
 rescode.err_read_format, 375
 rescode.err_read_lp_missing_end_tag, 377
 rescode.err_read_lp_nonexisting_name, 377
 rescode.err_remove_cone_variable, 381
 rescode.err_repair_invalid_problem, 383
 rescode.err_repair_optimization_failed, 383
 rescode.err_sen_bound_invalid_lo, 384
 rescode.err_sen_bound_invalid_up, 384
 rescode.err_sen_format, 384

```

rescode.err_sen_index_invalid, 384
rescode.err_sen_index_range, 384
rescode.err_sen_invalid_regexp, 384
rescode.err_sen_numerical, 384
rescode.err_sen_solution_status, 384
rescode.err_sen_undef_name, 384
rescode.err_sen_unhandled_problem_type, 384
rescode.err_server_connect, 387
rescode.err_server_problem_size, 388
rescode.err_server_protocol, 388
rescode.err_server_status, 388
rescode.err_server_token, 388
rescode.err_shape_is_too_large, 378
rescode.err_size_license, 373
rescode.err_size_license_con, 373
rescode.err_size_license_intvar, 374
rescode.err_size_license_numcores, 384
rescode.err_size_license_var, 373
rescode.err_slice_size, 382
rescode.err_sol_file_invalid_number, 381
rescode.err_solitem, 378
rescode.err_solver_probtype, 379
rescode.err_space, 375
rescode.err_space_leaking, 375
rescode.err_space_no_info, 375
rescode.err_sym_mat_duplicate, 385
rescode.err_sym_mat_huge, 382
rescode.err_sym_mat_invalid, 382
rescode.err_sym_mat_invalid_col_index, 385
rescode.err_sym_mat_invalid_row_index, 385
rescode.err_sym_mat_invalid_value, 385
rescode.err_sym_mat_not_lower_tringular,
    385
rescode.err_task_incompatible, 383
rescode.err_task_invalid, 383
rescode.err_task_write, 383
rescode.err_thread_cond_init, 374
rescode.err_thread_create, 374
rescode.err_thread_mutex_init, 374
rescode.err_thread_mutex_lock, 374
rescode.err_thread_mutex_unlock, 374
rescode.err_toconic_constr_not_conic, 387
rescode.err_toconic_constr_q_not_psd, 387
rescode.err_toconic_constraint_fx, 387
rescode.err_toconic_constraint_ra, 387
rescode.err_toconic_objective_not_psd, 387
rescode.err_too_small_a_truncation_value,
    382
rescode.err_too_small_max_num_nz, 379
rescode.err_too_small_maxnumanz, 379
rescode.err_unb_step_size, 384
rescode.err_undef_solution, 379
rescode.err_undefined_objective_sense, 382
rescode.err_unhandled_solution_status, 385
rescode.err_unknown, 374
rescode.err_upper_bound_is_a_nan, 381
rescode.err_upper_triangle, 385
rescode.err_whichitem_not_allowed, 379
rescode.err_whichsol, 378
rescode.err_write_lp_format, 377
rescode.err_write_lp_non_unique_name, 377
rescode.err_write_mps_invalid_name, 377
rescode.err_write_opf_invalid_var_name, 377
rescode.err_writing_file, 377
rescode.err_xml_invalid_problem_type, 384
rescode.err_y_is_undefined, 382

```

Index

A

- analysis
 - infeasibility, 183
- attaching
 - streams, 17

B

- basic
 - solution, 69
- basis identification, 100, 170
- basis type
 - sensitivity analysis, 189
- BLAS, 108
- bound
 - constraint, 13, 155, 158
 - linear optimization, 13
 - variable, 13, 155, 158

C

- callback, 79
- cardinality constraints, 61, 143
- CBF format, 442
- ceo1
 - example, 35
- certificate, 70
 - dual, 157, 161
 - primal, 157, 160
- Cholesky factorization, 109, 132
- CLASSPATH, 8
- column ordered
 - matrix format, 198
- complementarity, 156, 160
- concurrent optimizer, 150
- cone
 - dual, 159
 - dual exponential, 35
 - exponential, 35
 - power, 31
 - quadratic, 27
 - rotated quadratic, 27
 - semidefinite, 38
- conic exponential optimization, 35
- conic optimization, 27, 31, 35, 158
 - interior-point, 174
 - termination criteria, 175
- conic problem
 - example, 28, 32, 35
- conic quadratic optimization, 27
- Conic quadratic reformulation, 112

- constraint
 - bound, 13, 155, 158
 - linear optimization, 13
 - matrix, 13, 155, 158
 - quadratic, 163
- correlation matrix, 121
- covariance matrix, *see* correlation matrix
- cqo1
 - example, 28
- cut, 178

D

- defining
 - objective, 17
- determinism, 117
- dual
 - certificate, 157, 161
 - cone, 159
 - feasible, 156
 - infeasible, 156, 157, 161
 - problem, 156, 159, 162
 - solution, 71
 - variable, 156, 159
- duality
 - conic, 159
 - linear, 156
 - semidefinite, 162
- dualizer, 166

E

- efficient frontier, 129
- eliminator, 166
- entropy, 58
 - relative, 58
- environment variable
 - CLASSPATH, 8
- error
 - optimization, 69
- errors, 73
- example
 - ceo1, 35
 - conic problem, 28, 32, 35
 - cqo1, 28
 - lo1, 17
 - pow1, 32
 - qo1, 20
 - quadratic objective, 20
- exceptions, 73
- exponential, 58

exponential cone, 35

F

factor model, 132

feasible

dual, 156

primal, 155, 168, 174

problem, 155

format, 76

CBF, 442

json, 462

LP, 416

MPS, 421

OPF, 433

PTF, 457

sol, 470

task, 462

full

vector format, 197

G

geometric mean, 57

geometric programming, 50

GP, 50

H

hot-start, 172

I

I/O, 76

infeasibility, 70, 157, 160

analysis, 183

linear optimization, 157

repair, 183

semidefinite, 163

infeasible

dual, 156, 157, 161

primal, 155, 157, 160, 168, 175

problem, 155, 157, 163

information item, 78, 79

installation, 7

`nmake (command)`, 9, 10

requirements, 7

troubleshooting, 7

integer

optimizer, 178

solution, 69

variable, 45

integer feasible

solution, 179

integer optimization, 45, 178

cut, 178

initial solution, 48

objective bound, 178

optimality gap, 180

parameter, 45

relaxation, 178

termination criteria, 179

tolerance, 179

integer optimizer

logging, 180

interior-point

conic optimization, 174

linear optimization, 167

logging, 171, 177

optimizer, 167, 174

solution, 69

termination criteria, 169, 175

J

json format, 462

L

LAPACK, 108

license, 119

linear

objective, 17

linear constraint matrix, 13

linear dependency, 166

linear optimization, 13, 155

bound, 13

constraint, 13

infeasibility, 157

interior-point, 167

objective, 13

simplex, 172

termination criteria, 169, 172

variable, 13

linearity interval, 188

lo1

example, 17

log-sum-exp, 59, 147

logarithm, 57

logging, 75

integer optimizer, 180

interior-point, 171, 177

optimizer, 171, 173, 177

simplex, 173

logistic regression, 147

LP format, 416

M

machine learning

logistic regression, 147

market impact cost, 133

Markowitz

model, 120

Markowitz model, 121

portfolio optimization, 120

matrix

constraint, 13, 155, 158

semidefinite, 38

symmetric, 38

matrix format

column ordered, 198

row ordered, 198

- triplets, 198
- memory management, 116
- MIP, *see* integer optimization
- mixed-integer, *see* integer
- mixed-integer optimization, *see* integer optimization
- model
 - Markowitz, 120
 - portfolio optimization, 120
- modeling
 - design, 10
- monomial, 56
- MPS format, 421
 - free, 432
- N
- near-optimal
 - solution, 179
- norm
 - 1-norm, 55
 - 2-norm, 56
 - p-norm, 56
- numerical issues
 - presolve, 166
 - scaling, 166
 - simplex, 172
- O
- objective, 155, 158
 - defining, 17
 - linear, 17
 - linear optimization, 13
- objective bound, 178
- OPF format, 433
- optimal
 - solution, 70
- optimality gap, 180
- optimization
 - conic, 158
 - conic quadratic, 158
 - error, 69
 - linear, 13, 155
 - semidefinite, 162
- optimizer
 - concurrent, 150
 - determinism, 117
 - integer, 178
 - interior-point, 167, 174
 - interrupt, 79
 - logging, 171, 173, 177
 - parallel, 67
 - selection, 166, 167
 - simplex, 172
- P
- parallel optimization, 67, 150
- parallelization, 117
- parameter, 77
- integer optimization, 45
 - simplex, 172
- Pareto optimality, 121
- portfolio optimization
 - cardinality constraints, 61, 143
 - efficient frontier, 129
 - factor model, 132
 - market impact cost, 133
 - Markowitz model, 121
 - model, 120
 - Pareto optimality, 121
 - slippage cost, 133
 - transaction cost, 139
- positive semidefinite, 20
- pow1
 - example, 32
- power, 56
- power cone, 31
- power cone optimization, 31
- presolve, 165
 - eliminator, 166
 - linear dependency check, 166
 - numerical issues, 166
- primal
 - certificate, 157, 160
 - feasible, 155, 168, 174
 - infeasible, 155, 157, 160, 168, 175
 - problem, 156, 159, 162
 - solution, 71, 155
- primal-dual
 - problem, 168, 174
 - solution, 156
- problem
 - dual, 156, 159, 162
 - feasible, 155
 - infeasible, 155, 157, 163
 - load, 76
 - primal, 156, 159, 162
 - primal-dual, 168, 174
 - save, 76
 - status, 69
 - unbounded, 157, 161
- PTF format, 457
- Q
- qo1
 - example, 20
- quadratic
 - constraint, 163
- quadratic cone, 27
- quadratic objective
 - example, 20
- quadratic optimization, 163
- quality
 - solution, 180
- R
- regression

- logistic, 147
- relaxation, 178
- repair
 - infeasibility, 183
- response code, 73
- rotated quadratic cone, 27
- row ordered
 - matrix format, 198

S

- scaling, 166
- semicontinuous variable, 59
- semidefinite
 - cone, 38
 - infeasibility, 163
 - matrix, 38
 - variable, 38
- semidefinite optimization, 38, 162
- sensitivity analysis, 187
 - basis type, 189
- shadow price, 188
- simplex
 - linear optimization, 172
 - logging, 173
 - numerical issues, 172
 - optimizer, 172
 - parameter, 172
 - termination criteria, 172
- slippage cost, 133
- softplus, 59
- sol format, 470
- solution
 - basic, 69
 - dual, 71
 - file format, 470
 - integer, 69
 - integer feasible, 179
 - interior-point, 69
 - near-optimal, 179
 - optimal, 70
 - primal, 71, 155
 - primal-dual, 156
 - quality, 180
 - retrieve, 69
 - status, 16, 70
- solving linear system, 105
- sparse
 - vector format, 198
- sparse vector, 198
- status
 - problem, 69
 - solution, 16, 70
- streams
 - attaching, 17
- symmetric
 - matrix, 38

T

- task format, 462
- termination, 69
- termination criteria, 79
 - conic optimization, 175
 - integer optimization, 179
 - interior-point, 169, 175
 - linear optimization, 169, 172
 - simplex, 172
 - tolerance, 170, 176, 179
- thread, 117
- time limit, 79
- tolerance
 - integer optimization, 179
 - termination criteria, 170, 176, 179
- transaction cost, 139
- triplets
 - matrix format, 198
- troubleshooting
 - installation, 7

U

- unbounded
 - problem, 157, 161
- user callback, *see* callback

V

- variable, 155, 158
 - bound, 13, 155, 158
 - dual, 156, 159
 - integer, 45
 - linear optimization, 13
 - semicontinuous, 59
 - semidefinite, 38
- vector format
 - full, 197
 - sparse, 198