



MOSEK Optimizer API for C
Release 9.0.98

MOSEK ApS

15 July 2019

18.4	Constants	517
18.5	Response Codes	518
	Bibliography	520
	Symbol Index	521
	Index	537

Chapter 1

Introduction

The **MOSEK** Optimization Suite 9.0.98 is a powerful software package capable of solving large-scale optimization problems of the following kind:

- linear,
- conic:
 - conic quadratic (also known as second-order cone),
 - involving the exponential cone,
 - involving the power cone,
 - semidefinite,
- convex quadratic and quadratically constrained,
- integer.

In order to obtain an overview of features in the **MOSEK** Optimization Suite consult the [product introduction](#) guide.

The most widespread class of optimization problems is *linear optimization problems*, where all relations are linear. The tremendous success of both applications and theory of linear optimization can be ascribed to the following factors:

- The required data are simple, i.e. just matrices and vectors.
- Convexity is guaranteed since the problem is convex by construction.
- Linear functions are trivially differentiable.
- There exist very efficient algorithms and software for solving linear problems.
- Duality properties for linear optimization are nice and simple.

Even if the linear optimization model is only an approximation to the true problem at hand, the advantages of linear optimization may outweigh the disadvantages. In some cases, however, the problem formulation is inherently nonlinear and a linear approximation is either intractable or inadequate. *Conic optimization* has proved to be a very expressive and powerful way to introduce nonlinearities, while preserving all the nice properties of linear optimization listed above.

The fundamental expression in linear optimization is a linear expression of the form

$$Ax - b \geq 0.$$

In conic optimization this is replaced with a wider class of constraints

$$Ax - b \in \mathcal{K}$$

where \mathcal{K} is a *convex cone*. For example in 3 dimensions \mathcal{K} may correspond to an ice cream cone. The conic optimizer in **MOSEK** supports a number of different types of cones \mathcal{K} , which allows a surprisingly large number of nonlinear relations to be modeled, as described in the **MOSEK** [Modeling Cookbook](#), while preserving the nice algorithmic and theoretical properties of linear optimization.

1.1 Why the Optimizer API for C?

The Optimizer API for C provides low-level access to all functionalities of **MOSEK** from any C compatible language. It consists of a single header file and a set of library files which an application must link against when building. This interface has the smallest possible overhead, however other interfaces might be considered more convenient to use for the project at hand.

The Optimizer API for C 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 additional interfaces for:

- problem analysis,
- sensitivity analysis,
- infeasibility analysis,
- 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
Google+	https://plus.google.com/+Mosek/posts
Linkedin	https://www.linkedin.com/company/mosek-aps

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/9.0/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
```

fplib

MOSEK includes the floating point formatting library developed by David M. Gay obtained from the [netlib website](#). The license agreement for *fplib* is shown in [Listing 3.2](#).

Listing 3.2: *fplib* license.

```
/*****
 *
```

(continues on next page)

```
* The author of this software is David M. Gay.
*
* Copyright (c) 1991, 2000, 2001 by Lucent Technologies.
*
* Permission to use, copy, modify, and distribute this software for any
* purpose without fee is hereby granted, provided that this entire notice
* is included in all copies of any software which is or includes a copy
* or modification of this software and in all copies of the supporting
* documentation for such software.
*
* THIS SOFTWARE IS BEING PROVIDED "AS IS", WITHOUT ANY EXPRESS OR IMPLIED
* WARRANTY. IN PARTICULAR, NEITHER THE AUTHOR NOR LUCENT MAKES ANY
* REPRESENTATION OR WARRANTY OF ANY KIND CONCERNING THE MERCHANTABILITY
* OF THIS SOFTWARE OR ITS FITNESS FOR ANY PARTICULAR PURPOSE.
*
*****/
```

Zstandard

MOSEK includes the *Zstandard* library developed by Facebook obtained from [github/zstd](https://github.com/facebook/zstd). The license agreement for *Zstandard* is shown in [Listing 3.3](#).

Listing 3.3: *Zstandard* license.

```
BSD License

For Zstandard software

Copyright (c) 2016-present, Facebook, Inc. All rights reserved.

Redistribution and use in source and binary forms, with or without modification,
are permitted provided that the following conditions are met:

* Redistributions of source code must retain the above copyright notice, this
  list of conditions and the following disclaimer.

* Redistributions in binary form must reproduce the above copyright notice,
  this list of conditions and the following disclaimer in the documentation
  and/or other materials provided with the distribution.

* Neither the name Facebook nor the names of its contributors may be used to
  endorse or promote products derived from this software without specific
  prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND
ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED
WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE
DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS BE LIABLE FOR
ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES
(INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES;
LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON
ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
(INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS
SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
```


Chapter 5

Design Overview

5.1 Modeling

Optimizer API for C 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 C 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 C.

Creating an environment and task

Every interaction with **MOSEK** using Optimizer API for C 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.

Defining tasks

After a task is created, the input data can be specified. An optimization problem consists of several components; objective, objective sense, constraints, variable bounds etc. See [Sec. 6](#) for basic tutorials on how to specify and solve various types of optimization problems.

Retrieving the solutions

When the model is set up, the optimizer is invoked with the call to *MSK_optimize*. When the optimization is over, the user can check the results and retrieve numerical values. See further details in Sec. 7.

We refer also to Sec. 7 for information about more advanced mechanisms of interacting with the solver.

Source code example

Below is the most basic code sample that defines and solves a trivial optimization problem

$$\begin{array}{ll}\text{minimize} & x \\ \text{subject to} & 2.0 \leq x \leq 3.0.\end{array}$$

For simplicity the example does not contain any error or status checks.

Listing 5.1: “Hello World!” in MOSEK

```
#include "mosek.h"
#include <stdio.h>

/* Error checking not included */
int main() {
    MSKrescodee      r, trmcode;
    MSKenv_t         env = NULL;
    MSKtask_t        task = NULL;
    double           xx = 0.0;

    MSK_makeenv(&env, NULL);           // Create environment
    MSK_maketask(env, 0, 1, &task);    // Create task

    MSK_appendvars(task, 1);           // 1 variable x
    MSK_putcj(task, 0, 1.0);          // c_0 = 1.0
    MSK_putvarbound(task, 0, MSK_BK_RA, 2.0, 3.0); // 2.0 <= x <= 3.0
    MSK_putobjsense(task, MSK_OBJECTIVE_SENSE_MINIMIZE); // Minimize

    MSK_optimizetrm(task, &trmcode);  // Optimize

    MSK_getxx(task, MSK_SOL_ITR, &xx); // Get solution
    printf("Solution x = %f\n", xx);    // Print solution

    MSK_deletetask(&task); // Clean up task
    MSK_deleteenv(&env);   // Clean up environment
    return 0;
}
```

Chapter 6

Optimization Tutorials

In this section we demonstrate how to set up basic types of optimization problems. Each short tutorial contains a working example of formulating problems, defining variables and constraints and retrieving solutions.

6.1 Linear Optimization

The simplest optimization problem is a purely linear problem. A *linear optimization problem* is a problem of the following form:

Minimize or maximize the objective function

$$\sum_{j=0}^{n-1} c_j x_j + c^f$$

subject to the linear constraints

$$l_k^c \leq \sum_{j=0}^{n-1} a_{kj} x_j \leq u_k^c, \quad k = 0, \dots, m-1,$$

and the bounds

$$l_j^x \leq x_j \leq u_j^x, \quad j = 0, \dots, n-1.$$

The problem description consists of the following elements:

- m and n — the number of constraints and variables, respectively,
- x — the variable vector of length n ,
- c — the coefficient vector of length n

$$c = \begin{bmatrix} c_0 \\ \vdots \\ c_{n-1} \end{bmatrix},$$

- c^f — fixed term in the objective,
- A — an $m \times n$ matrix of coefficients

$$A = \begin{bmatrix} a_{0,0} & \cdots & a_{0,(n-1)} \\ \vdots & \cdots & \vdots \\ a_{(m-1),0} & \cdots & a_{(m-1),(n-1)} \end{bmatrix},$$


```

/* Directs the log task stream to the 'printstr' function. */
if (r == MSK_RES_OK)
    r = MSK_linkfunctotaskstream(task, MSK_STREAM_LOG, NULL, printstr);

/* Append 'numcon' empty constraints.
   The constraints will initially have no bounds. */
if (r == MSK_RES_OK)
    r = MSK_appendcons(task, numcon);

/* Append 'numvar' variables.
   The variables will initially be fixed at zero (x=0). */
if (r == MSK_RES_OK)
    r = MSK_appendvars(task, numvar);

for (j = 0; j < numvar && r == MSK_RES_OK; ++j)
{
    /* Set the linear term c_j in the objective.*/
    if (r == MSK_RES_OK)
        r = MSK_putcj(task, j, c[j]);

    /* Set the bounds on variable j.
       blx[j] <= x_j <= bux[j] */
    if (r == MSK_RES_OK)
        r = MSK_putvarbound(task,
                               j,           /* Index of variable.*/
                               bkc[j],      /* Bound key.*/
                               blx[j],       /* Numerical value of lower bound.*/
                               bux[j]);     /* Numerical value of upper bound.*/

    /* Input column j of A */
    if (r == MSK_RES_OK)
        r = MSK_putacol(task,
                          j,           /* Variable (column) index.*/
                          aptre[j] - aptrb[j], /* Number of non-zeros in column j.*/
                          asub + aptrb[j], /* Pointer to row indexes of column j.*/
                          aval + aptrb[j]); /* Pointer to Values of column j.*/
}

/* Set the bounds on constraints.
   for i=1, ..., numcon : blc[i] <= constraint i <= buc[i] */
for (i = 0; i < numcon && r == MSK_RES_OK; ++i)
    r = MSK_putconbound(task,
                          i,           /* Index of constraint.*/
                          bkc[i],      /* Bound key.*/
                          blc[i],       /* Numerical value of lower bound.*/
                          buc[i]);     /* Numerical value of upper bound.*/

/* Maximize objective function. */
if (r == MSK_RES_OK)
    r = MSK_putobjsense(task, MSK_OBJECTIVE_SENSE_MAXIMIZE);

if (r == MSK_RES_OK)
{
    MSKrescodeee trmcode;

    /* Run optimizer */
    r = MSK_optimizetrm(task, &trmcode);

    /* Print a summary containing information

```

```

    about the solution for debugging purposes. */
    MSK_solutionsummary(task, MSK_STREAM_LOG);

    if (r == MSK_RES_OK)
    {
        MSKsolstae solsta;

        if (r == MSK_RES_OK)
            r = MSK_getsolsta(task,
                             MSK_SOL_BAS,
                             &solsta);

        switch (solsta)
        {
            case MSK_SOL_STA_OPTIMAL:
            {
                double *xx = (double*) calloc(numvar, sizeof(double));
                if (xx)
                {
                    MSK_getxx(task,
                             MSK_SOL_BAS,    /* Request the basic solution. */
                             xx);

                    printf("Optimal primal solution\n");
                    for (j = 0; j < numvar; ++j)
                        printf("x[%d]: %e\n", j, xx[j]);

                    free(xx);
                }
                else
                    r = MSK_RES_ERR_SPACE;

                break;
            }
            case MSK_SOL_STA_DUAL_INFEAS_CER:
            case MSK_SOL_STA_PRIM_INFEAS_CER:
                printf("Primal or dual infeasibility certificate found.\n");
                break;
            case MSK_SOL_STA_UNKNOWN:
            {
                char symname[MSK_MAX_STR_LEN];
                char desc[MSK_MAX_STR_LEN];

                /* If the solutions status is unknown, print the termination code
                   indicating why the optimizer terminated prematurely. */

                MSK_getcodedesc(trmcode,
                               symname,
                               desc);

                printf("The solution status is unknown.\n");
                printf("The optimizer terminated with code: %s\n", symname);
                break;
            }
            default:
                printf("Other solution status.\n");
                break;
        }
    }
}

if (r != MSK_RES_OK)

```


Listing 6.2: Source code implementing problem (6.4).

```
#include <stdio.h>

#include "mosek.h" /* Include the MOSEK definition file. */

#define NUMCON 1 /* Number of constraints. */
#define NUMVAR 3 /* Number of variables. */
#define NUMANZ 3 /* Number of non-zeros in A. */
#define NUMQNZ 4 /* Number of non-zeros in Q. */

static void MSKAPI printstr(void *handle,
                           const char str[])
{
    printf("%s", str);
} /* printstr */

int main(int argc, const char *argv[])
{
    double          c[]    = {0.0, -1.0, 0.0};

    MSKboundkey     bkc[]  = {MSK_BK_LO};
    double          blc[]  = {1.0};
    double          buc[]  = { +MSK_INFINITY };

    MSKboundkey     bkc[]  = {MSK_BK_LO,
                              MSK_BK_LO,
                              MSK_BK_LO
                              };
    double          blx[]  = {0.0,
                              0.0,
                              0.0
                              };
    double          bux[]  = { +MSK_INFINITY,
                              +MSK_INFINITY,
                              +MSK_INFINITY
                              };

    MSKint32t       aptrb[] = {0, 1, 2},
    aptre[]         = {1, 2, 3},
    asub[]          = {0, 0, 0};
    double          aval[]  = {1.0, 1.0, 1.0};

    MSKint32t       qsubi[ NUMQNZ ];
    MSKint32t       qsubj[ NUMQNZ ];
    double          qval[ NUMQNZ ];

    MSKint32t       i, j;
    double          xx[ NUMVAR ];

    MSKenv_t        env = NULL;
    MSKtask_t       task = NULL;
    MSKrescodee     r;

    /* Create the mosek environment. */
    r = MSK_makeenv(&env, NULL);

    if (r == MSK_RES_OK)
    {
        /* Create the optimization task. */
        r = MSK_maketask(env, NUMCON, NUMVAR, &task);
    }
}
```

(continues on next page)

```

if (r == MSK_RES_OK)
{
    r = MSK_linkfunctotaskstream(task, MSK_STREAM_LOG, NULL, printstr);

    /* Append 'NUMCON' empty constraints.
       The constraints will initially have no bounds. */
    if (r == MSK_RES_OK)
        r = MSK_appendcons(task, NUMCON);

    /* Append 'NUMVAR' variables.
       The variables will initially be fixed at zero (x=0). */
    if (r == MSK_RES_OK)
        r = MSK_appendvars(task, NUMVAR);

    /* Optionally add a constant term to the objective. */
    if (r == MSK_RES_OK)
        r = MSK_putcfix(task, 0.0);
    for (j = 0; j < NUMVAR && r == MSK_RES_OK; ++j)
    {
        /* Set the linear term c_j in the objective.*/
        if (r == MSK_RES_OK)
            r = MSK_putcj(task, j, c[j]);

        /* Set the bounds on variable j.
           blx[j] <= x_j <= bux[j] */
        if (r == MSK_RES_OK)
            r = MSK_putvarbound(task,
                                j,           /* Index of variable.*/
                                bkc[j],     /* Bound key.*/
                                blx[j],     /* Numerical value of lower bound.*/
                                bux[j]);    /* Numerical value of upper bound.*/

        /* Input column j of A */
        if (r == MSK_RES_OK)
            r = MSK_putacol(task,
                                j,           /* Variable (column) index.*/
                                aptre[j] - aptrb[j], /* Number of non-zeros in column j.*/
                                asub + aptrb[j], /* Pointer to row indexes of column j.*/
                                aval + aptrb[j]); /* Pointer to Values of column j.*/
    }

    /* Set the bounds on constraints.
       for i=1, ..., NUMCON : blc[i] <= constraint i <= buc[i] */
    for (i = 0; i < NUMCON && r == MSK_RES_OK; ++i)
        r = MSK_putconbound(task,
                                i,           /* Index of constraint.*/
                                bkc[i],     /* Bound key.*/
                                blc[i],     /* Numerical value of lower bound.*/
                                buc[i]);    /* Numerical value of upper bound.*/

    if (r == MSK_RES_OK)
    {
        /*
         * The lower triangular part of the Q
         * matrix in the objective is specified.
         */

        qsubi[0] = 0;   qsubj[0] = 0;   qval[0] = 2.0;
        qsubi[1] = 1;   qsubj[1] = 1;   qval[1] = 0.2;
    }
}

```

```

qsubi[2] = 2;   qsubj[2] = 0;   qval[2] = -1.0;
qsubi[3] = 2;   qsubj[3] = 2;   qval[3] = 2.0;

/* Input the Q for the objective. */

r = MSK_putqobj(task, NUMQNZ, qsubi, qsubj, qval);
}

if (r == MSK_RES_OK)
{
    MSKrescodee trmcode;

    /* Run optimizer */
    r = MSK_optimizetrm(task, &trmcode);

    /* Print a summary containing information
       about the solution for debugging purposes*/
    MSK_solutionsummary(task, MSK_STREAM_MSG);

    if (r == MSK_RES_OK)
    {
        MSKsolsta solsta;
        int j;

        MSK_getsolsta(task, MSK_SOL_ITR, &solsta);

        switch (solsta)
        {
            case MSK_SOL_STA_OPTIMAL:
                MSK_getxx(task,
                           MSK_SOL_ITR, /* Request the interior solution. */
                           xx);

                printf("Optimal primal solution\n");
                for (j = 0; j < NUMVAR; ++j)
                    printf("x[%d]: %e\n", j, xx[j]);

                break;

            case MSK_SOL_STA_DUAL_INFEAS_CER:
            case MSK_SOL_STA_PRIM_INFEAS_CER:
                printf("Primal or dual infeasibility certificate found.\n");
                break;

            case MSK_SOL_STA_UNKNOWN:
                printf("The status of the solution could not be determined. Termination code: %d.\n", trmcode);
                break;

            default:
                printf("Other solution status.");
                break;
        }
    }
    else
    {
        printf("Error while optimizing.\n");
    }
}

if (r != MSK_RES_OK)

```



```

4,
qsubi,
qsubj,
qval);

```

While *MSK_putqconk* adds quadratic terms to a specific constraint, it is also possible to input all quadratic terms in one chunk using the *MSK_putqcon* function.

Source code

Listing 6.3: Implementation of the quadratically constrained problem (6.5).

```

#include <stdio.h>

#include "mosek.h" /* Include the MOSEK definition file. */

#define NUMCON 1 /* Number of constraints. */
#define NUMVAR 3 /* Number of variables. */
#define NUMANZ 3 /* Number of non-zeros in A. */
#define NUMQNZ 4 /* Number of non-zeros in Q. */

static void MSKAPI printstr(void *handle,
                             const char str[])
{
    printf("%s", str);
} /* printstr */

int main(int argc, const char *argv[])
{
    MSKrescodee r;

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

    MSKboundkeye bkc[] = {MSK_BK_LO};
    double blc[] = {1.0};
    double buc[] = {+MSK_INFINITY};

    MSKboundkeye bkc[] = {MSK_BK_LO,
                          MSK_BK_LO,
                          MSK_BK_LO};
    double blx[] = {0.0,
                   0.0,
                   0.0};
    double bux[] = {+MSK_INFINITY,
                   +MSK_INFINITY,
                   +MSK_INFINITY};

    MSKint32t aptrb[] = {0, 1, 2},
               aptre[] = {1, 2, 3},
               asub[] = {0, 0, 0};
    double aval[] = {1.0, 1.0, 1.0};
    MSKint32t qsubi[NUMQNZ],
               qsubj[NUMQNZ];
    double qval[NUMQNZ];

```

(continues on next page)

```

MSKint32t    j, i;
double       xx[NUMVAR];
MSKenv_t     env;
MSKtask_t    task;

/* Create the mosek environment. */
r = MSK_makeenv(&env, NULL);

if (r == MSK_RES_OK)
{
    /* Create the optimization task. */
    r = MSK_maketask(env, NUMCON, NUMVAR, &task);

    if (r == MSK_RES_OK)
    {
        r = MSK_linkfunctotaskstream(task, MSK_STREAM_LOG, NULL, printstr);

        /* Append 'NUMCON' empty constraints.
           The constraints will initially have no bounds. */
        if (r == MSK_RES_OK)
            r = MSK_appendcons(task, NUMCON);

        /* Append 'NUMVAR' variables.
           The variables will initially be fixed at zero (x=0). */
        if (r == MSK_RES_OK)
            r = MSK_appendvars(task, NUMVAR);

        /* Optionally add a constant term to the objective. */
        if (r == MSK_RES_OK)
            r = MSK_putcfix(task, 0.0);
        for (j = 0; j < NUMVAR && r == MSK_RES_OK; ++j)
        {
            /* Set the linear term c_j in the objective.*/
            if (r == MSK_RES_OK)
                r = MSK_putcj(task, j, c[j]);

            /* Set the bounds on variable j.
               blx[j] <= x_j <= bux[j] */
            if (r == MSK_RES_OK)
                r = MSK_putvarbound(task,
                                     j,           /* Index of variable.*/
                                     bkc[j],      /* Bound key.*/
                                     blx[j],       /* Numerical value of lower bound.*/
                                     bux[j]);      /* Numerical value of upper bound.*/

            /* Input column j of A */
            if (r == MSK_RES_OK)
                r = MSK_putacol(task,
                                   j,           /* Variable (column) index.*/
                                   aptre[j] - aptrb[j], /* Number of non-zeros in column j.*/
                                   asub + aptrb[j], /* Pointer to row indexes of column j.*/
                                   aval + aptrb[j]); /* Pointer to Values of column j.*/
        }

        /* Set the bounds on constraints.
           for i=1, ..., NUMCON : blc[i] <= constraint i <= buc[i] */
        for (i = 0; i < NUMCON && r == MSK_RES_OK; ++i)
            r = MSK_putconbound(task,
                                   i,           /* Index of constraint.*/
                                   bkc[i],      /* Bound key.*/

```

```

        blc[i],      /* Numerical value of lower bound.*/
        buc[i]);    /* Numerical value of upper bound.*/

if (r == MSK_RES_OK)
{
    /*
     * The lower triangular part of the  $Q^0$ 
     * matrix in the objective is specified.
     */

    qsubi[0] = 0;  qsubj[0] = 0;  qval[0] = 2.0;
    qsubi[1] = 1;  qsubj[1] = 1;  qval[1] = 0.2;
    qsubi[2] = 2;  qsubj[2] = 0;  qval[2] = -1.0;
    qsubi[3] = 2;  qsubj[3] = 2;  qval[3] = 2.0;

    /* Input the  $Q^0$  for the objective. */

    r = MSK_putqobj(task, NUMQNZ, qsubi, qsubj, qval);
}

if (r == MSK_RES_OK)
{
    /*
     * The lower triangular part of the  $Q^0$ 
     * matrix in the first constraint is specified.
     * This corresponds to adding the term
     * -  $x_1^2$  -  $x_2^2$  -  $0.1 x_3^2$  +  $0.2 x_1 x_3$ 
     */

    qsubi[0] = 0;  qsubj[0] = 0;  qval[0] = -2.0;
    qsubi[1] = 1;  qsubj[1] = 1;  qval[1] = -2.0;
    qsubi[2] = 2;  qsubj[2] = 2;  qval[2] = -0.2;
    qsubi[3] = 2;  qsubj[3] = 0;  qval[3] = 0.2;

    /* Put  $Q^0$  in constraint with index 0. */

    r = MSK_putqconk(task,
                     0,
                     4,
                     qsubi,
                     qsubj,
                     qval);
}

if (r == MSK_RES_OK)
    r = MSK_putobjsense(task, MSK_OBJECTIVE_SENSE_MINIMIZE);

if (r == MSK_RES_OK)
{
    MSKrescodee trmcode;

    /* Run optimizer */
    r = MSK_optimizetrm(task, &trmcode);

    /* Print a summary containing information
     * about the solution for debugging purposes*/
    MSK_solutionsummary(task, MSK_STREAM_LOG);

    if (r == MSK_RES_OK)
    {
        MSKsolstae solsta;

```

```

    int j;

    MSK_getsolsta(task, MSK_SOL_ITR, &solsta);

    switch (solsta)
    {
        case MSK_SOL_STA_OPTIMAL:
            MSK_getxx(task,
                      MSK_SOL_ITR,    /* Request the interior solution. */
                      xx);

            printf("Optimal primal solution\n");
            for (j = 0; j < NUMVAR; ++j)
                printf("x[%d]: %e\n", j, xx[j]);

            break;

        case MSK_SOL_STA_DUAL_INFEAS_CER:
        case MSK_SOL_STA_PRIM_INFEAS_CER:
            printf("Primal or dual infeasibility certificate found.\n");
            break;

        case MSK_SOL_STA_UNKNOWN:
            printf("The status of the solution could not be determined. Termination code: %d.
↪\n", trmcode);
            break;

        default:
            printf("Other solution status.");
            break;
    }
}
else
{
    printf("Error while optimizing.\n");
}

if (r != MSK_RES_OK)
{
    /* In case of an error print error code and description. */
    char symname[MSK_MAX_STR_LEN];
    char desc[MSK_MAX_STR_LEN];

    printf("An error occurred while optimizing.\n");
    MSK_getcodedesc(r,
                    symname,
                    desc);
    printf("Error %s - '%s'\n", symname, desc);
}

MSK_deletetask(&task);
}
MSK_deleteenv(&env);

return (r);
} /* main */

```


Setting up the linear part

The linear parts (constraints, variables, objective) are set up using exactly the same methods as for linear problems, and we refer to [Sec. 6.1](#) for all the details. The same applies to technical aspects such as defining an optimization task, retrieving the solution and so on.

Setting up the conic constraints

A cone is defined using the function `MSK_appendcone`:

```
csub[0] = 3;
csub[1] = 0;
csub[2] = 1;

r = MSK_appendcone(task,
                    MSK_CT_QUAD,
                    0.0, /* Can be set to 0.0 */
                    3,
                    csub);
```

The first argument selects the type of quadratic cone, in this case either `MSK_CT_QUAD` for a *quadratic cone* or `MSK_CT_RQUAD` for a *rotated quadratic cone*. The second parameter is currently ignored and passing 0.0 will work.

The next argument denotes the number of variables in the cone, in this case 3, and the last argument is a list of indexes of the variables appearing in the cone.

Variants of this method are available to append multiple cones at a time.

Source code

Listing 6.4: Source code solving problem (6.6).

```
#include <stdio.h>
#include "mosek.h" /* Include the MOSEK definition file. */

static void MSKAPI printstr(void *handle,
                             const char str[])
{
    printf("%s", str);
} /* printstr */

int main(int argc, const char *argv[])
{
    MSKrescodee r;

    const MSKint32t numvar = 6,
                    numcon = 1;

    MSKboundkeye bkc[] = { MSK_BK_FX };
    double       blc[] = { 1.0 };
    double       buc[] = { 1.0 };

    MSKboundkeye bkc[] = {MSK_BK_LO,
                          MSK_BK_LO,
                          MSK_BK_LO,
                          MSK_BK_FR,
                          MSK_BK_FR,
                          MSK_BK_FR
                          };
    double       blx[] = {0.0,
                          0.0,
                          0.0,
```

(continues on next page)

```

        -MSK_INFINITY,
        -MSK_INFINITY,
        -MSK_INFINITY
    };

    double    bux[] = { +MSK_INFINITY,
                        +MSK_INFINITY,
                        +MSK_INFINITY,
                        +MSK_INFINITY,
                        +MSK_INFINITY,
                        +MSK_INFINITY
    };

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

    MSKint32t  aptrb[] = {0, 1, 2, 3, 3, 3},
               aptre[] = {1, 2, 3, 3, 3, 3},
               asub[]  = {0, 0, 0, 0};
    double     aval[]  = {1.0, 1.0, 2.0};

    MSKint32t  i, j, csub[3];

    MSKenv_t    env  = NULL;
    MSKtask_t   task = NULL;

    /* Create the mosek environment. */
    r = MSK_makeenv(&env, NULL);

    if (r == MSK_RES_OK)
    {
        /* Create the optimization task. */
        r = MSK_maketask(env, numcon, numvar, &task);

        if (r == MSK_RES_OK)
        {
            MSK_linkfunctotaskstream(task, MSK_STREAM_LOG, NULL, printstr);

            /* Append 'numcon' empty constraints.
             The constraints will initially have no bounds. */
            if (r == MSK_RES_OK)
                r = MSK_appendcons(task, numcon);

            /* Append 'numvar' variables.
             The variables will initially be fixed at zero (x=0). */
            if (r == MSK_RES_OK)
                r = MSK_appendvars(task, numvar);

            for (j = 0; j < numvar && r == MSK_RES_OK; ++j)
            {
                /* Set the linear term c_j in the objective.*/
                if (r == MSK_RES_OK)
                    r = MSK_putcj(task, j, c[j]);

                /* Set the bounds on variable j.
                 blx[j] <= x_j <= bux[j] */
            }
        }
    }

```

(continues on next page)

```

if (r == MSK_RES_OK)
    r = MSK_putvarbound(task,
                        j,           /* Index of variable.*/
                        bkc[j],     /* Bound key.*/
                        blc[j],     /* Numerical value of lower bound.*/
                        buc[j]);    /* Numerical value of upper bound.*/

/* Input column j of A */
if (r == MSK_RES_OK)
    r = MSK_putacol(task,
                    j,           /* Variable (column) index.*/
                    aptre[j] - aptrb[j], /* Number of non-zeros in column j.*/
                    asub + aptrb[j], /* Pointer to row indexes of column j.*/
                    aval + aptrb[j]); /* Pointer to Values of column j.*/

}

/* Set the bounds on constraints.
for i=1, ..., numcon : blc[i] <= constraint i <= buc[i] */
for (i = 0; i < numcon && r == MSK_RES_OK; ++i)
    r = MSK_putconbound(task,
                        i,           /* Index of constraint.*/
                        bkc[i],     /* Bound key.*/
                        blc[i],     /* Numerical value of lower bound.*/
                        buc[i]);    /* Numerical value of upper bound.*/

if (r == MSK_RES_OK)
{
    /* Append the first cone. */
    csub[0] = 3;
    csub[1] = 0;
    csub[2] = 1;

    r = MSK_appendcone(task,
                      MSK_CT_QUAD,
                      0.0, /* Can be set to 0.0 */
                      3,
                      csub);
}

if (r == MSK_RES_OK)
{
    /* Append the second cone. */
    csub[0] = 4;
    csub[1] = 5;
    csub[2] = 2;

    r = MSK_appendcone(task,
                      MSK_CT_RQUAD,
                      0.0,
                      3,
                      csub);
}

if (r == MSK_RES_OK)
{
    MSKrescodeee trmcode;

    /* Run optimizer */
    r = MSK_optimizetrm(task, &trmcode);
}

```

```

/* Print a summary containing information
   about the solution for debugging purposes*/
MSK_solutionsummary(task, MSK_STREAM_MSG);

if (r == MSK_RES_OK)
{
    MSKsolstae solsta;

    MSK_getsolsta(task, MSK_SOL_ITR, &solsta);

    switch (solsta)
    {
        case MSK_SOL_STA_OPTIMAL:
        {
            double *xx = NULL;

            xx = calloc(numvar, sizeof(double));
            if (xx)
            {
                MSK_getxx(task,
                           MSK_SOL_ITR,    /* Request the interior solution. */
                           xx);

                printf("Optimal primal solution\n");
                for (j = 0; j < numvar; ++j)
                    printf("x[%d]: %e\n", j, xx[j]);
            }
            else
            {
                r = MSK_RES_ERR_SPACE;
            }
            free(xx);
        }
        break;
        case MSK_SOL_STA_DUAL_INFEAS_CER:
        case MSK_SOL_STA_PRIM_INFEAS_CER:
            printf("Primal or dual infeasibility certificate found.\n");
            break;
        case MSK_SOL_STA_UNKNOWN:
            printf("The status of the solution could not be determined. Termination code: %d.
↪\n", trmcode);
            break;
        default:
            printf("Other solution status.");
            break;
    }
}
else
{
    printf("Error while optimizing.\n");
}

if (r != MSK_RES_OK)
{
    /* In case of an error print error code and description. */
    char symname[MSK_MAX_STR_LEN];
    char desc[MSK_MAX_STR_LEN];

    printf("An error occurred while optimizing.\n");
}

```



```

double      blx[6], bux[6];

double      val[] = { 1.0, 1.0, -1.0 };
MSKint32t   sub[] = { 3, 4, 0 };

double      aval[] = { 1.0, 1.0, 0.5 };
MSKint32t   asub[] = { 0, 1, 2 };

MSKint32t   i, j;

MSKenv_t     env = NULL;
MSKtask_t    task = NULL;

/* Create the mosek environment. */
r = MSK_makeenv(&env, NULL);

if (r == MSK_RES_OK)
{
    /* Create the optimization task. */
    r = MSK_maketask(env, numcon, numvar, &task);

    if (r == MSK_RES_OK)
    {
        MSK_linkfunctotaskstream(task, MSK_STREAM_LOG, NULL, printstr);

        /* Append 'numcon' empty constraints.
         The constraints will initially have no bounds. */
        if (r == MSK_RES_OK)
            r = MSK_appendcons(task, numcon);

        /* Append 'numvar' variables.
         The variables will initially be fixed at zero (x=0). */
        if (r == MSK_RES_OK)
            r = MSK_appendvars(task, numvar);

        /* Set up the linear part */
        MSK_putclist(task, 3, sub, val);
        MSK_putarow(task, 0, 3, asub, aval);
        MSK_putconbound(task, 0, MSK_BK_FX, 2.0, 2.0);
        for(i=0; i<5; i++)
            bkg[i] = MSK_BK_FR, blx[i] = -MSK_INFINITY, bux[i] = MSK_INFINITY;
        bkg[5] = MSK_BK_FX, blx[5] = bux[5] = 1.0;
        MSK_putvarboundslice(task, 0, numvar, bkg, blx, bux);

        if (r == MSK_RES_OK)
        {
            /* Append two power cones. */
            MSKint32t csub[2][3] = { {0, 1, 3}, {2, 5, 4} };
            r = MSK_appendcone(task, MSK_CT_PPOW, 0.2, 3, csub[0]);
            r = MSK_appendcone(task, MSK_CT_PPOW, 0.4, 3, csub[1]);
        }

        MSK_putobjsense(task, MSK_OBJECTIVE_SENSE_MAXIMIZE);

        if (r == MSK_RES_OK)
        {
            MSKrescodee trmcode;

            /* Run optimizer */
            r = MSK_optimizetrm(task, &trmcode);

```

```

/* Print a summary containing information
   about the solution for debugging purposes*/
MSK_solutionsummary(task, MSK_STREAM_MSG);

if (r == MSK_RES_OK)
{
    MSKsolstae solsta;

    MSK_getsolsta(task, MSK_SOL_ITR, &solsta);

    switch (solsta)
    {
        case MSK_SOL_STA_OPTIMAL:
        {
            double *xx = NULL;

            xx = calloc(numvar, sizeof(double));
            if (xx)
            {
                MSK_getxx(task,
                          MSK_SOL_ITR,    /* Request the interior solution. */
                          xx);

                printf("Optimal primal solution\n");
                for (j = 0; j < 3; ++j)
                    printf("x[%d]: %e\n", j, xx[j]);
            }
            else
            {
                r = MSK_RES_ERR_SPACE;
            }
            free(xx);
        }
        break;
        case MSK_SOL_STA_DUAL_INFEAS_CER:
        case MSK_SOL_STA_PRIM_INFEAS_CER:
            printf("Primal or dual infeasibility certificate found.\n");
            break;
        case MSK_SOL_STA_UNKNOWN:
            printf("The status of the solution could not be determined. Termination code: %d.
↪\n", trmcode);
            break;
        default:
            printf("Other solution status.");
            break;
    }
}
else
{
    printf("Error while optimizing.\n");
}
}

if (r != MSK_RES_OK)
{
    /* In case of an error print error code and description. */
    char symname[MSK_MAX_STR_LEN];
    char desc[MSK_MAX_STR_LEN];

    printf("An error occurred while optimizing.\n");
    MSK_getcodedesc(r,

```



```

const MSKint32t numvar = 3,
              numcon = 1;

MSKboundkeye bkc = MSK_BK_FX;
double      blc = 1.0;
double      buc = 1.0;

MSKboundkeye bkc[] = {MSK_BK_FR,
                      MSK_BK_FR,
                      MSK_BK_FR
                      };
double      blx[] = {-MSK_INFINITY,
                    -MSK_INFINITY,
                    -MSK_INFINITY
                    };
double      bux[] = {+MSK_INFINITY,
                    +MSK_INFINITY,
                    +MSK_INFINITY
                    };

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

double      a[] = {1.0, 1.0, 1.0};
MSKint32t asub[] = {0, 1, 2};

MSKint32t i, j, csub[3];

MSKenv_t env = NULL;
MSKtask_t task = NULL;

/* Create the mosek environment. */
r = MSK_makeenv(&env, NULL);

if (r == MSK_RES_OK)
{
    /* Create the optimization task. */
    r = MSK_maketask(env, numcon, numvar, &task);

    if (r == MSK_RES_OK)
    {
        MSK_linkfunctotaskstream(task, MSK_STREAM_LOG, NULL, printstr);

        /* Append 'numcon' empty constraints.
         The constraints will initially have no bounds. */
        if (r == MSK_RES_OK)
            r = MSK_appendcons(task, numcon);

        /* Append 'numvar' variables.
         The variables will initially be fixed at zero (x=0). */
        if (r == MSK_RES_OK)
            r = MSK_appendvars(task, numvar);

        /* Set up the linear part */
        if (r == MSK_RES_OK)
            r = MSK_putcslice(task, 0, numvar, c);
        if (r == MSK_RES_OK)
            r = MSK_putarow(task, 0, numvar, asub, a);
    }
}

```

```

if (r == MSK_RES_OK)
    r = MSK_putconbound(task, 0, bkc, blc, buc);
if (r == MSK_RES_OK)
    r = MSK_putvarboundslice(task, 0, numvar, bkx, blx, bux);

if (r == MSK_RES_OK)
{
    /* Append an exponential cone. */
    csub[0] = 0;
    csub[1] = 1;
    csub[2] = 2;

    r = MSK_appendcone(task,
                        MSK_CT_PEXP,
                        0.0, /* Can be set to 0.0 */
                        3,
                        csub);
}

if (r == MSK_RES_OK)
{
    MSKrescodee trmcode;

    /* Run optimizer */
    r = MSK_optimizetrm(task, &trmcode);

    /* Print a summary containing information
       about the solution for debugging purposes*/
    MSK_solutionsummary(task, MSK_STREAM_MSG);

    if (r == MSK_RES_OK)
    {
        MSKsolstae solsta;

        MSK_getsolsta(task, MSK_SOL_ITR, &solsta);

        switch (solsta)
        {
            case MSK_SOL_STA_OPTIMAL:
            {
                double *xx = NULL;

                xx = calloc(numvar, sizeof(double));
                if (xx)
                {
                    MSK_getxx(task,
                             MSK_SOL_ITR, /* Request the interior solution. */
                             xx);

                    printf("Optimal primal solution\n");
                    for (j = 0; j < numvar; ++j)
                        printf("x[%d]: %e\n", j, xx[j]);
                }
                else
                {
                    r = MSK_RES_ERR_SPACE;
                }
                free(xx);
            }
            break;
            case MSK_SOL_STA_DUAL_INFEAS_CER:

```

(continues on next page)


```

        aptre[] = {1, 3},
        asub[] = {0, 1, 2}; /* column subscripts of A */
double    aval[] = {1.0, 1.0, 1.0};

MSKint32t  bara_i[] = {0, 1, 2, 0, 1, 2, 1, 2, 2},
        bara_j[] = {0, 1, 2, 0, 0, 0, 1, 1, 2};
double    bara_v[] = {1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0};
MSKint32t  conesub[] = {0, 1, 2};

MSKint32t  i, j;
MSKint64t  idx;
double    falpha = 1.0;

MSKrealt   *xx;
MSKrealt   *barx;
MSKenv_t   env = NULL;
MSKtask_t  task = NULL;

/* Create the mosek environment. */
r = MSK_makeenv(&env, NULL);

if (r == MSK_RES_OK)
{
    /* Create the optimization task. */
    r = MSK_maketask(env, NUMCON, 0, &task);

    if (r == MSK_RES_OK)
    {
        MSK_linkfunctotaskstream(task, MSK_STREAM_LOG, NULL, printstr);

        /* Append 'NUMCON' empty constraints.
           The constraints will initially have no bounds. */
        if (r == MSK_RES_OK)
            r = MSK_appendcons(task, NUMCON);

        /* Append 'NUMVAR' variables.
           The variables will initially be fixed at zero (x=0). */
        if (r == MSK_RES_OK)
            r = MSK_appendvars(task, NUMVAR);

        /* Append 'NUMBARVAR' semidefinite variables. */
        if (r == MSK_RES_OK) {
            r = MSK_appendbarvars(task, NUMBARVAR, DIMBARVAR);
        }

        /* Optionally add a constant term to the objective. */
        if (r == MSK_RES_OK)
            r = MSK_putcfix(task, 0.0);

        /* Set the linear term c_j in the objective.*/
        if (r == MSK_RES_OK)
            r = MSK_putcj(task, 0, 1.0);

        for (j = 0; j < NUMVAR && r == MSK_RES_OK; ++j)
            r = MSK_putvarbound(task,
                                j,
                                MSK_BK_FR,
                                -MSK_INFINITY,
                                MSK_INFINITY);

        /* Set the linear term barc_j in the objective.*/

```

```

if (r == MSK_RES_OK)
    r = MSK_appendsparsesymmat(task,
                                DIMBARVAR[0],
                                5,
                                barc_i,
                                barc_j,
                                barc_v,
                                &idx);

if (r == MSK_RES_OK)
    r = MSK_putbarcj(task, 0, 1, &idx, &falpha);

/* Set the bounds on constraints.
   for i=1, ..., NUMCON : blc[i] <= constraint i <= buc[i] */
for (i = 0; i < NUMCON && r == MSK_RES_OK; ++i)
    r = MSK_putconbound(task,
                        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 (i = 0; i < NUMCON && r == MSK_RES_OK; ++i)
    r = MSK_putarow(task,
                    i,
                    aptre[i] - aptrb[i],
                    asub    + aptrb[i],
                    aval    + aptrb[i]);

/* Append the conic quadratic cone */
if (r == MSK_RES_OK)
    r = MSK_appendcone(task,
                      MSK_CT_QUAD,
                      0.0,
                      3,
                      conesub);

/* Add the first row of barA */
if (r == MSK_RES_OK)
    r = MSK_appendsparsesymmat(task,
                                DIMBARVAR[0],
                                3,
                                bara_i,
                                bara_j,
                                bara_v,
                                &idx);

if (r == MSK_RES_OK)
    r = MSK_putbaraij(task, 0, 0, 1, &idx, &falpha);

/* Add the second row of barA */
if (r == MSK_RES_OK)
    r = MSK_appendsparsesymmat(task,
                                DIMBARVAR[0],
                                6,
                                bara_i + 3,
                                bara_j + 3,
                                bara_v + 3,
                                &idx);

if (r == MSK_RES_OK)

```

```

r = MSK_putbaraij(task, 1, 0, 1, &idx, &falpha);

if (r == MSK_RES_OK)
{
    MSKrescodee trmcode;

    /* Run optimizer */
    r = MSK_optimizetrm(task, &trmcode);

    /* Print a summary containing information
       about the solution for debugging purposes*/
    MSK_solutionsummary(task, MSK_STREAM_MSG);

    if (r == MSK_RES_OK)
    {
        MSKsolstae solsta;

        MSK_getsolsta(task, MSK_SOL_ITR, &solsta);

        switch (solsta)
        {
            case MSK_SOL_STA_OPTIMAL:
                xx = (MSKrealt *) MSK_calloc(task, NUMVAR, sizeof(MSKrealt));
                barx = (MSKrealt *) MSK_calloc(task, LENBARVAR[0], sizeof(MSKrealt));

                MSK_getxx(task,
                           MSK_SOL_ITR,
                           xx);
                MSK_getbarxj(task,
                             MSK_SOL_ITR, /* Request the interior solution. */
                             0,
                             barx);

                printf("Optimal primal solution\n");
                for (i = 0; i < NUMVAR; ++i)
                    printf("x[%d] : % e\n", i, xx[i]);

                for (i = 0; i < LENBARVAR[0]; ++i)
                    printf("barx[%d]: % e\n", i, barx[i]);

                MSK_freetask(task, xx);
                MSK_freetask(task, barx);

                break;

            case MSK_SOL_STA_DUAL_INFEAS_CER:
            case MSK_SOL_STA_PRIM_INFEAS_CER:
                printf("Primal or dual infeasibility certificate found.\n");
                break;

            case MSK_SOL_STA_UNKNOWN:
                printf("The status of the solution could not be determined. Termination code: %d.\n", trmcode);
                break;

            default:
                printf("Other solution status.");
                break;
        }
    }
    else

```


The complete source for the example is listed Listing 6.8. Please note that when *MSK_getsolutionslice* is called, the integer solution is requested by using *MSK_SOL_ITG*. No dual solution is defined for integer optimization problems.

Listing 6.8: Source code implementing problem (6.12).

```
#include <stdio.h>
#include "mosek.h" /* Include the MOSEK definition file. */

static void MSKAPI printstr(void *handle,
                           const char str[])
{
    printf("%s", str);
} /* printstr */

int main(int argc, char *argv[])
{
    const MSKint32t numvar = 2,
                  numcon = 2;

    double        c[] = { 1.0, 0.64 };
    MSKboundkeye bkc[] = { MSK_BK_UP,    MSK_BK_LO };
    double        blc[] = { -MSK_INFINITY, -4.0 };
    double        buc[] = { 250.0,      MSK_INFINITY };

    MSKboundkeye bkc[] = { MSK_BK_LO,    MSK_BK_LO };
    double        blx[] = { 0.0,        0.0 };
    double        bux[] = { MSK_INFINITY, MSK_INFINITY };

    MSKint32t     aptrb[] = { 0, 2 },
                aptre[] = { 2, 4 },
                asub[] = { 0, 1, 0, 1 };
    double        aval[] = { 50.0, 3.0, 31.0, -2.0 };
    MSKint32t     i, j;

    MSKenv_t      env = NULL;
    MSKtask_t     task = NULL;
    MSKrescodee   r;

    /* Create the mosek environment. */
    r = MSK_makeenv(&env, NULL);

    /* Check if return code is ok. */
    if (r == MSK_RES_OK)
    {
        /* Create the optimization task. */
        r = MSK_maketask(env, 0, 0, &task);

        if (r == MSK_RES_OK)
            r = MSK_linkfunctotaskstream(task, MSK_STREAM_LOG, NULL, printstr);

        /* Append 'numcon' empty constraints.
           The constraints will initially have no bounds. */
        if (r == MSK_RES_OK)
            r = MSK_appendcons(task, numcon);

        /* Append 'numvar' variables.
           The variables will initially be fixed at zero (x=0). */
        if (r == MSK_RES_OK)
            r = MSK_appendvars(task, numvar);
    }
}
```

(continues on next page)


```

if (r == MSK_RES_OK)
{
    MSKint32t j;
    MSKsolstae solsta;
    double *xx = NULL;

    MSK_getsolsta(task, MSK_SOL_ITG, &solsta);

    xx = calloc(numvar, sizeof(double));
    if (xx)
    {
        switch (solsta)
        {
            case MSK_SOL_STA_INTEGER_OPTIMAL:
                MSK_getxx(task,
                        MSK_SOL_ITG, /* Request the integer solution. */
                        xx);

                printf("Optimal solution.\n");
                for (j = 0; j < numvar; ++j)
                    printf("x[%d]: %e\n", j, xx[j]);
                break;
            case MSK_SOL_STA_PRIM_FEAS:
                /* A feasible but not necessarily optimal solution was located. */
                MSK_getxx(task, MSK_SOL_ITG, xx);

                printf("Feasible solution.\n");
                for (j = 0; j < numvar; ++j)
                    printf("x[%d]: %e\n", j, xx[j]);
                break;
            case MSK_SOL_STA_UNKNOWN:
                {
                    MSKprosta prostae;
                    MSK_getprosta(task, MSK_SOL_ITG, &prostae);
                    switch (prostae)
                    {
                        case MSK_PRO_STA_PRIM_INFEAS_OR_UNBOUNDED:
                            printf("Problem status Infeasible or unbounded\n");
                            break;
                        case MSK_PRO_STA_PRIM_INFEAS:
                            printf("Problem status Infeasible.\n");
                            break;
                        case MSK_PRO_STA_UNKNOWN:
                            printf("Problem status unknown. Termination code %d.\n", trmcode);
                            break;
                        default:
                            printf("Other problem status.");
                            break;
                    }
                }
                break;
            default:
                printf("Other solution status.");
                break;
        }
    }
    else
    {
        r = MSK_RES_ERR_SPACE;
    }
}

```


(continued from previous page)

```
printf("Task %d res %d trm %d obj_val %.5f time %.5f\n",
      i,
      res[i],
      trm[i],
      obj,
      tm);
}

for(int i = 0; i < n; i++)
    MSK_deletetask(&(tasks[i]));
delete[] tasks;
delete[] res;
delete[] trm;
MSK_deleteenv(&env);
return 0;
}
```

Another, slightly more advanced application of the parallel optimizer is presented in [Sec. 11.3](#). For a more in-depth treatment see the following sections:

- *Case studies* for more advanced and complicated optimization examples.
- *Problem Formulation and Solutions* for formal mathematical formulations of problems **MOSEK** can solve, dual problems and infeasibility certificates.


```

if (res == MSK_RES_OK)
    res = MSK_readdata(task, filename);

if (res == MSK_RES_OK)
    res = MSK_linkfunctotaskstream(task, MSK_STREAM_LOG, NULL, printstr);

for (i = 0; i < numpolls && res == MSK_RES_OK ; i++)
{
    #if __linux__
        sleep(1);
    #elif defined(_WIN32)
        Sleep(1000);
    #endif

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

    res = MSK_asyncpoll(task,
                        host,
                        port,
                        token,
                        &respavailable,
                        &resp,
                        &trm);

    puts("polling done");

    if (respavailable)
    {
        puts("solution available!");

        res = MSK_asyncgetresult(task,
                                host,
                                port,
                                token,
                                &respavailable,
                                &resp,
                                &trm);

        MSK_solutionsummary(task, MSK_STREAM_LOG);
        break;
    }
}

if (i == numpolls)
{
    printf("max num polls reached, stopping %s", host);
    MSK_asyncstop(task, host, port, token);
}

MSK_deletetask(&task);
MSK_deleteenv(&env);

printf("%s:%d: Result = %d\n", __FILE__, __LINE__, res); fflush(stdout);

return res;
}

```



```

    On return w contains the solution x and sub
    the index of the non-zeros in x.
*/
if (r == MSK_RES_OK)
    r = MSK_solvewithbasis(task, 0, &nz, sub, w);

if (r == MSK_RES_OK)
{
    printf("\nSolution to Bx = w:\n\n");

    /* Print solution and b. */

    for (i = 0; i < nz; ++i)
    {
        if (basis[sub[i]] < numcon)
            printf("xc%d = %e\n", basis[sub[i]] , w[sub[i]]);
        else
            printf("x%d = %e\n", basis[sub[i]] - numcon , w[sub[i]]);
    }
}

/* Solve B^T y = w */
nz      = 1;    /* Only one element in sub is nonzero. */
sub[0] = 1;     /* Only w[1] is nonzero. */
w[0]    = 0.0;
w[1]    = 1.0;

if (r == MSK_RES_OK)
    r = MSK_solvewithbasis(task, 1, &nz, sub, w);

if (r == MSK_RES_OK)
{
    printf("\nSolution to B^T y = w:\n\n");
    /* Print solution and y. */
    for (i = 0; i < nz; ++i)
        printf("y%d = %e\n", sub[i], w[sub[i]]);
}

return (r);
}/* main */

```

In the example above the linear system is solved using the optimal basis for (9.4) and the original right-hand side of the problem. Thus the solution to the linear system is the optimal solution to the problem. When running the example program the following output is produced.

```

basis[0] = 1
Basis variable no 0 is xc1.
basis[1] = 2
Basis variable no 1 is x0.

Solution to Bx = b:

x0 = 2.000000e+00
xc1 = -4.000000e+00

Solution to B^Tx = c:

x1 = -1.000000e+00
x0 = 1.000000e+00

```


(continued from previous page)

```
if (skx == NULL && numvar)
    r = MSK_RES_ERR_SPACE;

skc = (MSKstakeye *) calloc(numvar, sizeof(MSKstakeye));
if (skc == NULL && numvar)
    r = MSK_RES_ERR_SPACE;

for (i = 0; i < numvar && r == MSK_RES_OK; ++i)
{
    skx[i] = MSK_SK_BAS;
    skc[i] = MSK_SK_FIX;
}

/* Create a coefficient matrix and right hand
   side with the data from the linear system */
if (r == MSK_RES_OK)
    r = MSK_appendvars(task, numvar);

if (r == MSK_RES_OK)
    r = MSK_appendcons(task, numvar);

for (i = 0; i < numvar && r == MSK_RES_OK; ++i)
    r = MSK_putacol(task, i, ptre[i] - ptrb[i], asub + ptrb[i], aval + ptrb[i]);

for (i = 0; i < numvar && r == MSK_RES_OK; ++i)
    r = MSK_putconbound(task, i, MSK_BK_FX, 0, 0);

for (i = 0; i < numvar && r == MSK_RES_OK; ++i)
    r = MSK_putvarbound(task, i, MSK_BK_FR, -MSK_INFINITY, MSK_INFINITY);

/* Allocate space for the solution and set status to unknown */
if (r == MSK_RES_OK)
    r = MSK_deletesolution(task, MSK_SOL_BAS);

/* Setup status keys. That is all that is needed. */
if (r == MSK_RES_OK)
    r = MSK_putskcslice(task, MSK_SOL_BAS, 0, numvar, skc);

if (r == MSK_RES_OK)
    r = MSK_putskxslice(task, MSK_SOL_BAS, 0, numvar, skx);

if (r == MSK_RES_OK)
    r = MSK_initbasissolve(task, basis);

free(skx);
free(skc);

return (r);
}

#define NUMCON 2
#define NUMVAR 2

int main(int argc, const char *argv[])
{
    const MSKint32t numvar = NUMCON,
                  numcon = NUMVAR;    /* we must have numvar == numcon */
    MSKenv_t      env;
    MSKtask_t     task;
```

(continues on next page)

```

MSKrescodee      r = MSK_RES_OK;
MSKint32t        i, nz;
double           aval[] = { -1.0, 1.0, 1.0};
MSKint32t        asub[] = {1, 0, 1};
MSKint32t        ptrb[] = {0, 1};
MSKint32t        ptre[] = {1, 3};
MSKint32t        bsub[NUMCON];
double           b[NUMCON];
MSKint32t        *basis = NULL;

if (r == MSK_RES_OK)
    r = MSK_makeenv(&env, NULL);

if (r == MSK_RES_OK)
    r = MSK_makeemptytask(env, &task);

if (r == MSK_RES_OK)
    MSK_linkfunctotaskstream(task, MSK_STREAM_LOG, NULL, printstr);

basis = (MSKint32t *) calloc(numcon, sizeof(MSKint32t));
if (basis == NULL && numvar)
    r = MSK_RES_ERR_SPACE;

/* setup: Put A matrix and factor A.
   Call this function only once for a given task. */
if (r == MSK_RES_OK)
    r = setup(task,
              aval,
              asub,
              ptrb,
              ptre,
              numvar,
              basis
              );

/* now solve rhs */
b[0] = 1;
b[1] = -2;
bsub[0] = 0;
bsub[1] = 1;
nz = 2;

if (r == MSK_RES_OK)
    r = MSK_solvewithbasis(task, 0, &nz, bsub, b);

if (r == MSK_RES_OK)
{
    printf("\nSolution to Bx = b:\n\n");
    /* Print solution and show correspondents
       to original variables in the problem */
    for (i = 0; i < nz; ++i)
    {
        if (basis[bsub[i]] < numcon)
            printf("This should never happen\n");
        else
            printf("x%d = %e\n", basis[bsub[i]] - numcon, b[bsub[i]]);
    }
}

b[0] = 7;
bsub[0] = 0;

```

```

nz = 1;

if (r == MSK_RES_OK)
    r = MSK_solvewithbasis(task, 0, &nz, bsub, b);

if (r == MSK_RES_OK)
{
    printf("\nSolution to Bx = b:\n\n");
    /* Print solution and show correspondents
       to original variables in the problem */
    for (i = 0; i < nz; ++i)
    {
        if (basis[bsub[i]] < numcon)
            printf("This should never happen\n");
        else
            printf("x%d = %e\n", basis[bsub[i]] - numcon, b[bsub[i]]);
    }
}

free(basis);
return r;
}

```

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

```

9.2 Calling BLAS/LAPACK Routines from MOSEK

Sometimes users need to perform linear algebra operations that involve dense matrices and vectors. Also **MOSEK** extensively uses high-performance linear algebra routines from the BLAS and LAPACK packages and some of these routines are included in the package shipped to the users.

The **MOSEK** versions of BLAS/LAPACK routines:

- use **MOSEK** data types and return value conventions,
- preserve the BLAS/LAPACK naming convention.

Therefore the user can leverage on efficient linear algebra routines, with a simplified interface, with no need for additional packages.


```

/* BLAS routines*/
r = MSK_makeenv(&env, NULL);
printf("n=%d m=%d k=%d\n", m, n, k);
printf("alpha=%f\n", alpha);
printf("beta=%f\n", beta);

r = MSK_dot(env, n, x, y, &xy);
printf("dot results= %f r=%d\n", xy, r);

print_matrix(x, 1, n);
print_matrix(y, 1, n);

r = MSK_axpy(env, n, alpha, x, y);
puts("axpy results is");
print_matrix(y, 1, n);

r = MSK_gemv(env, MSK_TRANSPOSE_NO, m, n, alpha, A, x, beta, z);
printf("gemv results is (r=%d) \n", r);
print_matrix(z, 1, m);

r = MSK_gemm(env, MSK_TRANSPOSE_NO, MSK_TRANSPOSE_NO, m, n, k, alpha, A, B, beta, C);
printf("gemm results is (r=%d) \n", r);
print_matrix(C, m, n);

r = MSK_syrk(env, MSK_UPLO_LO, MSK_TRANSPOSE_NO, m, k, 1., A, beta, D);
printf("syrk results is (r=%d) \n", r);
print_matrix(D, m, m);

/* LAPACK routines*/

r = MSK_potrf(env, MSK_UPLO_LO, m, Q);
printf("potrf results is (r=%d) \n", r);
print_matrix(Q, m, m);

r = MSK_syeig(env, MSK_UPLO_LO, m, Q, v);
printf("sy eig results is (r=%d) \n", r);
print_matrix(v, 1, m);

r = MSK_syevd(env, MSK_UPLO_LO, m, Q, v);
printf("sy evd results is (r=%d) \n", r);
print_matrix(v, 1, m);
print_matrix(Q, m, m);

/* Delete the environment and the associated data. */
MSK_deleteenv(&env);

return r;
}

```

9.3 Computing a Sparse Cholesky Factorization

Given a positive semidefinite symmetric (PSD) matrix

$$A \in \mathbb{R}^{n \times n}$$

it is well known there exists a matrix L such that

$$A = LL^T.$$

The model generated by *MSK_toconic* is

```
[comment]
  Written by MOSEK version 8.1.0.19
  Date 21-08-17
  Time 10:53:36
[/comment]

[hints]
  [hint NUMVAR] 9 [/hint]
  [hint NUMCON] 4 [/hint]
  [hint NUMANZ] 11 [/hint]
  [hint NUMQNZ] 0 [/hint]
  [hint NUMCONE] 1 [/hint]
[/hints]

[variables disallow_new_variables]
  x0000_x0 x0001_x1 x0002_x2 x0003 x0004
  x0005 x0006 x0007 x0008
[/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.

10.6 Deployment

When redistributing a C application using the **MOSEK** Optimizer API for C 9.0.98, the following libraries must be included:

64-bit Linux	64-bit Windows	32-bit Windows	64-bit Mac OS
libmosek64.so.9.0	mosek64_9_0.dll	mosek9_0.dll	libmosek64.9.0.dylib
libcilkrts.so.5	cilkrts20.dll	cilkrts20.dll	libcilkrts.5.dylib

10.7 Strings

MOSEK supports Unicode strings. All arguments of type `char *` are allowed to be UTF8 encoded strings (<http://en.wikipedia.org/wiki/UTF-8>). Please note that

- an ASCII string is always a valid UTF8 string, and
- an UTF8 string is stored in a `char` array.

It is possible to convert between `wchar_t` strings and UTF8 strings using the functions *MSK_wchartoutf8* and *MSK_utf8towchar*. A working example is provided in the example file `unicode.c`.

Chapter 11

Case Studies

In this section we present some case studies in which the Optimizer API for C 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*


```

for (k = 0; k < n; ++k)
    if ( GT[k][j] != 0.0 )
        MOSEKCALL(res, MSK_putaij(task, 1 + k, offsetx + j, GT[k][j]));
/* Diagonal -1 entries in a block of A */
MOSEKCALL(res, MSK_putaij(task, 1 + j, offsett + j, -1.0));
/* Free - no bounds */
MOSEKCALL(res, MSK_putvarbound(task, offsett + j, MSK_BK_FR, -MSK_INFINITY, MSK_INFINITY));
sprintf(buf, "t[%d]", 1 + j);
MOSEKCALL(res, MSK_putvarname(task, offsett + j, buf));
}

if ( res == MSK_RES_OK )
{
    /* Define the cone spanned by variables (s, t), i.e. dimension = n + 1 */
    MSKint32t *sub = (MSKint32t *) MSK_calloc(task, n + 1, sizeof(MSKint32t));

    if ( sub )
    {
        /* Copy indices of variables involved in the conic constraint */
        sub[0] = offsets + 0;
        for (j = 0; j < n; ++j)
            sub[j + 1] = offsett + j;

        MOSEKCALL(res, MSK_appendcone(task, MSK_CT_QUAD, 0.0, n + 1, sub));
        MOSEKCALL(res, MSK_putconename(task, 0, "stddev"));

        MSK_freetask(task, sub);
    }
    else
        res = MSK_RES_ERR_SPACE;
}

MOSEKCALL(res, MSK_putobjsense(task, MSK_OBJECTIVE_SENSE_MAXIMIZE));

#if 0
/* No log output */
MOSEKCALL(res, MSK_putintparam(task, MSK_IPAR_LOG, 0));
#endif

#if 0
/* Dump the problem to a human readable OPF file. */
MOSEKCALL(res, MSK_writedata(task, "dump.opf"));
#endif

MOSEKCALL(res, MSK_optimizetrm(task, &trmcode));

#if 1
/* Display the solution summary for quick inspection of results. */
MSK_solutionsummary(task, MSK_STREAM_MSG);
#endif

if ( res == MSK_RES_OK )
{
    expret = 0.0;
    stddev = 0.0;

    /* Read the x variables one by one and compute expected return. */
    /* Can also be obtained as value of the objective. */
    for (j = 0; j < n; ++j)
    {
        MOSEKCALL(res, MSK_getxxslice(task, MSK_SOL_ITR, offsetx + j, offsetx + j + 1, &xj));
    }
}

```


Listing 11.5: Code implementing model (11.9).

```
#include <math.h>
#include <stdio.h>

#include "mosek.h"

#define MOSEKCALL(_r,_call)  if ( (_r)==MSK_RES_OK ) (_r) = (_call)
#define LOGLEVEL             0

static void MSKAPI printstr(void      *handle,
                             const char str[])
{
    printf("%s", str);
} /* printstr */

int main(int argc, const char **argv)
{
    char          buf[128];
    const MSKint32t n      = 3,
                  numalpha = 12;
    const double  mu[]     = {0.1073, 0.0737, 0.0627},
                  x0[3]     = {0.0, 0.0, 0.0},
                  w         = 1.0,
                  alphas[12] = {0.0, 0.25, 0.5, 0.75, 1.0, 1.5, 2.0, 2.5, 3.0, 3.5, 4.0, 4.5},
                  GT[][3]    = {{0.1667, 0.0232, 0.0013},
                                {0.0000, 0.1033, -0.0022},
                                {0.0000, 0.0000, 0.0338}
                                };

    MSKenv_t      env;
    MSKint32t     k, i, j, offsetx, offsets, offsett, offsetu;
    MSKrescodeee  res = MSK_RES_OK, lres;
    MSKtask_t     task;
    MSKrealt      xj;
    MSKsolstae    solsta;

    /* Initial setup. */
    env = NULL;
    task = NULL;

    /* Replace "" with NULL in production. */
    MOSEKCALL(res, MSK_makeenv(&env, ""));

    MOSEKCALL(res, MSK_maketask(env, 0, 0, &task));
    MOSEKCALL(res, MSK_linkfunctotaskstream(task, MSK_STREAM_LOG, NULL, printstr));

    /* Constraints. */
    MOSEKCALL(res, MSK_appendcons(task, 1 + n));
    MOSEKCALL(res, MSK_putconbound(task, 0, MSK_BK_FX, 1.0, 1.0));
    sprintf(buf, "%s", "budget");
    MOSEKCALL(res, MSK_putconname(task, 0, buf));

    for (i = 0; i < n; ++i)
    {
        MOSEKCALL(res, MSK_putconbound(task, 1 + i, MSK_BK_FX, 0.0, 0.0));
        sprintf(buf, "GT[%d]", 1 + i);
        MOSEKCALL(res, MSK_putconname(task, 1 + i, buf));
    }

    /* Variables. */
    MOSEKCALL(res, MSK_appendvars(task, 2 + 2 * n));
```

(continues on next page)

```

offsetx = 0;  /* Offset of variable x into the API variable. */
offsets = n;  /* Offset of variable s into the API variable. */
offsett = n + 1; /* Offset of variable t into the API variable. */
offsetu = 2*n + 1; /* Offset of variable u into the API variable. */

/* x variables. */
for (j = 0; j < n; ++j)
{
    MOSEKCALL(res, MSK_putcj(task, offsetx + j, mu[j]));
    MOSEKCALL(res, MSK_putaij(task, 0, offsetx + j, 1.0));
    for (k = 0; k < n; ++k)
        if (GT[k][j] != 0.0)
            MOSEKCALL(res, MSK_putaij(task, 1 + k, offsetx + j, GT[k][j]));

    MOSEKCALL(res, MSK_putvarbound(task, offsetx + j, MSK_BK_LO, 0.0, MSK_INFINITY));
    sprintf(buf, "x[%d]", 1 + j);
    MOSEKCALL(res, MSK_putvarname(task, offsetx + j, buf));
}

/* s variable. */
MOSEKCALL(res, MSK_putvarbound(task, offsets + 0, MSK_BK_FR, -MSK_INFINITY, MSK_INFINITY));
sprintf(buf, "s");
MOSEKCALL(res, MSK_putvarname(task, offsets + 0, buf));

/* t variables. */
for (j = 0; j < n; ++j)
{
    MOSEKCALL(res, MSK_putaij(task, 1 + j, offsett + j, -1.0));
    MOSEKCALL(res, MSK_putvarbound(task, offsett + j, MSK_BK_FR, -MSK_INFINITY, MSK_INFINITY));
    sprintf(buf, "t[%d]", 1 + j);
    MOSEKCALL(res, MSK_putvarname(task, offsett + j, buf));
}

/* u variable. */
MOSEKCALL(res, MSK_putvarbound(task, offsetu + 0, MSK_BK_FX, 0.5, 0.5));
sprintf(buf, "u");
MOSEKCALL(res, MSK_putvarname(task, offsetu + 0, buf));

if (res == MSK_RES_OK)
{
    /* sub should be n+2 long i.e. the dimension of the cone. */
    MSKint32t *sub = (MSKint32t *) MSK_calloctask(task, n + 2, sizeof(MSKint32t));

    if (sub)
    {
        sub[0] = offsets + 0;
        sub[1] = offsetu + 0;
        for (j = 0; j < n; ++j)
            sub[j + 2] = offsett + j;

        MOSEKCALL(res, MSK_appendcone(task, MSK_CT_RQUAD, 0.0, n + 2, sub));
        MOSEKCALL(res, MSK_putconename(task, 0, "variance"));

        MSK_freetask(task, sub);
    }
    else
        res = MSK_RES_ERR_SPACE;
}

MOSEKCALL(res, MSK_putobjsense(task, MSK_OBJECTIVE_SENSE_MAXIMIZE));

```

```

/* Set the log level */
MOSEKCALL(res, MSK_putintparam(task, MSK_IPAR_LOG, LOGLEVEL));

printf("%-12s  %-12s  %-12s\n", "alpha", "exp ret", "variance");

for (k = 0; k < numalpha && res==MSK_RES_OK; ++k)
{
    const double alpha = alphas[k];
    MSKrescodee trmcode;

    /* Sets the objective function coefficient for s. */
    MOSEKCALL(res, MSK_putcj(task, offsets + 0, -alpha));

    MOSEKCALL(res, MSK_optimizetrm(task, &trmcode));

    MOSEKCALL(res, MSK_getsolsta(task, MSK_SOL_ITR, &solsta));

    if (solsta == MSK_SOL_STA_OPTIMAL)
    {
        double expret = 0.0,
            stddev;

        for (j = 0; j < n; ++j)
        {
            MOSEKCALL(res, MSK_getxxslice(task, MSK_SOL_ITR, offsetx + j, offsetx + j + 1, &xj));
            expret += mu[j] * xj;
        }

        MOSEKCALL(res, MSK_getxxslice(task, MSK_SOL_ITR, offsets + 0, offsets + 1, &stddev));

        printf("%-12.3e  %-12.3e  %-12.3e\n", alpha, expret, stddev);
    }
    else
    {
        printf("An error occurred when solving for alpha=%e\n", alpha);
    }
}
lres = MSK_deletetask(&task);
res = res==MSK_RES_OK ? lres : res;

lres = MSK_deleteenv(&env);
res = res==MSK_RES_OK ? lres : res;

return ( res );
}

```

11.1.3 Factor model and efficiency

In practice it is often important to solve the portfolio problem very quickly. Therefore, in this section we discuss how to improve computational efficiency at the modeling stage.

The computational cost is of course to some extent dependent on the number of constraints and variables in the optimization problem. However, in practice a more important factor is the sparsity: the number of nonzeros used to represent the problem. Indeed it is often better to focus on the number of nonzeros in G see (11.2) and try to reduce that number by for instance changing the choice of G .

In other words if the computational efficiency should be improved then it is always good idea to start with focusing at the covariance matrix. As an example assume that

$$\Sigma = D + VV^T$$

where D is a positive definite diagonal matrix. Moreover, V is a matrix with n rows and p columns. Such a model for the covariance matrix is called a factor model and usually p is much smaller than n .


```

char          buf[128];
const MSKint32t n      = 3;
const double   w       = 1.0,
              x0[]     = {0.0, 0.0, 0.0},
              gamma    = 0.05,
              mu[]     = {0.1073, 0.0737, 0.0627},
              m[]      = {0.01, 0.01, 0.01},
              GT[][3]  = {{0.1667, 0.0232, 0.0013},
                          {0.0000, 0.1033, -0.0022},
                          {0.0000, 0.0000, 0.0338}
                          };

double        expret,
              stddev,
              xj;
MSKenv_t      env;
MSKint32t     k, i, j,
              offsetx, offsets, offsett, offsetc,
              offsetz, offsetf;
MSKrescodee   res = MSK_RES_OK, trmcode;
MSKtask_t     task;

/* Initial setup. */
env = NULL;
task = NULL;
MOSEKCALL(res, MSK_makeenv(&env, NULL));
MOSEKCALL(res, MSK_maketask(env, 0, 0, &task));
MOSEKCALL(res, MSK_linkfunctotaskstream(task, MSK_STREAM_LOG, NULL, printstr));

/* Constraints. */
MOSEKCALL(res, MSK_appendcons(task, 1 + 3 * n));
MOSEKCALL(res, MSK_putconbound(task, 0, MSK_BK_FX, w, w));
sprintf(buf, "%s", "budget");
MOSEKCALL(res, MSK_putconname(task, 0, buf));

for (i = 0; i < n; ++i)
{
    MOSEKCALL(res, MSK_putconbound(task, 1 + i, MSK_BK_FX, 0.0, 0.0));
    sprintf(buf, "GT[%d]", 1 + i);
    MOSEKCALL(res, MSK_putconname(task, 1 + i, buf));
}

for (i = 0; i < n; ++i)
{
    MOSEKCALL(res, MSK_putconbound(task, 1 + n + i, MSK_BK_LO, -x0[i], MSK_INFINITY));
    sprintf(buf, "zabs1[%d]", 1 + i);
    MOSEKCALL(res, MSK_putconname(task, 1 + n + i, buf));
}

for (i = 0; i < n; ++i)
{
    MOSEKCALL(res, MSK_putconbound(task, 1 + 2 * n + i, MSK_BK_LO, x0[i], MSK_INFINITY));
    sprintf(buf, "zabs2[%d]", 1 + i);
    MOSEKCALL(res, MSK_putconname(task, 1 + 2 * n + i, buf));
}

/* Offsets of variables into the (serialized) API variable. */
offsetx = 0;
offsets = n;
offsett = n + 1;
offsetc = 2 * n + 1;
offsetz = 3 * n + 1;

```

```

offsetf = 4 * n + 1;

/* Variables. */
MOSEKCALL(res, MSK_appendvars(task, 5 * n + 1));

/* x variables. */
for (j = 0; j < n; ++j)
{
    MOSEKCALL(res, MSK_putcj(task, offsetx + j, mu[j]));
    MOSEKCALL(res, MSK_putaij(task, 0, offsetx + j, 1.0));
    for (k = 0; k < n; ++k)
        if (GT[k][j] != 0.0)
            MOSEKCALL(res, MSK_putaij(task, 1 + k, offsetx + j, GT[k][j]));
    MOSEKCALL(res, MSK_putaij(task, 1 + n + j, offsetx + j, -1.0));
    MOSEKCALL(res, MSK_putaij(task, 1 + 2 * n + j, offsetx + j, 1.0));

    MOSEKCALL(res, MSK_putvarbound(task, offsetx + j, MSK_BK_LO, 0.0, MSK_INFINITY));
    sprintf(buf, "x[%d]", 1 + j);
    MOSEKCALL(res, MSK_putvarname(task, offsetx + j, buf));
}

/* s variable. */
MOSEKCALL(res, MSK_putvarbound(task, offsets + 0, MSK_BK_FX, gamma, gamma));
sprintf(buf, "s");
MOSEKCALL(res, MSK_putvarname(task, offsets + 0, buf));

/* t variables. */
for (j = 0; j < n; ++j)
{
    MOSEKCALL(res, MSK_putaij(task, 1 + j, offsett + j, -1.0));
    MOSEKCALL(res, MSK_putvarbound(task, offsett + j, MSK_BK_FR, -MSK_INFINITY, MSK_INFINITY));
    sprintf(buf, "t[%d]", 1 + j);
    MOSEKCALL(res, MSK_putvarname(task, offsett + j, buf));
}

/* c variables. */
for (j = 0; j < n; ++j)
{
    MOSEKCALL(res, MSK_putaij(task, 0, offsetc + j, m[j]));
    MOSEKCALL(res, MSK_putvarbound(task, offsetc + j, MSK_BK_FR, -MSK_INFINITY, MSK_INFINITY));
    sprintf(buf, "c[%d]", 1 + j);
    MOSEKCALL(res, MSK_putvarname(task, offsetc + j, buf));
}

/* z variables. */
for (j = 0; j < n; ++j)
{
    MOSEKCALL(res, MSK_putaij(task, 1 + 1 * n + j, offsetz + j, 1.0));
    MOSEKCALL(res, MSK_putaij(task, 1 + 2 * n + j, offsetz + j, 1.0));
    MOSEKCALL(res, MSK_putvarbound(task, offsetz + j, MSK_BK_FR, -MSK_INFINITY, MSK_INFINITY));
    sprintf(buf, "z[%d]", 1 + j);
    MOSEKCALL(res, MSK_putvarname(task, offsetz + j, buf));
}

/* f variables. */
for (j = 0; j < n; ++j)
{
    MOSEKCALL(res, MSK_putvarbound(task, offsetf + j, MSK_BK_FX, 1.0, 1.0));
    sprintf(buf, "f[%d]", 1 + j);
    MOSEKCALL(res, MSK_putvarname(task, offsetf + j, buf));
}

```

```

if (res == MSK_RES_OK)
{
    /* sub should be n+1 long i.e. the dimension of the cone. */
    MSKint32t *sub = (MSKint32t *) MSK_calloc(task, 3 >= n + 1 ? 3 : n + 1,
↪sizeof(MSKint32t));

    if (sub)
    {
        /* Add quadratic cone */
        sub[0] = offsets + 0;
        for (j = 0; j < n; ++j)
            sub[j + 1] = offsett + j;

        MOSEKCALL(res, MSK_appendcone(task, MSK_CT_QUAD, 0.0, n + 1, sub));
        MOSEKCALL(res, MSK_putconename(task, 0, "stddev"));

        /* Add power cones */
        for (k = 0; k < n; ++k)
        {
            sub[0] = offsetc + k, sub[1] = offsetf + k, sub[2] = offsetz + k;
            MOSEKCALL(res, MSK_appendcone(task, MSK_CT_PPOW, 2.0/3.0, 3, sub));
            sprintf(buf, "trans[%d]", 1 + k);
            MOSEKCALL(res, MSK_putconename(task, 1 + k, buf));
        }

        MSK_freetask(task, sub);
    }
    else
        res = MSK_RES_ERR_SPACE;
}

MOSEKCALL(res, MSK_putobjsense(task, MSK_OBJECTIVE_SENSE_MAXIMIZE));

#if 0
    /* no log output. */
    MOSEKCALL(res, MSK_putintparam(task, MSK_IPAR_LOG, 0));
#endif

#if 0
    /* Dump the problem to a human readable OPF file. */
    MOSEKCALL(res, MSK_writedata(task, "dump.opf"));
#endif

MOSEKCALL(res, MSK_optimizetrm(task, &trmcode));

/* Display the solution summary for quick inspection of results. */
#if 1
    MSK_solutionsummary(task, MSK_STREAM_MSG);
#endif

if (res == MSK_RES_OK)
{
    expret = 0.0;
    stddev = 0.0;

    for (j = 0; j < n; ++j)
    {
        MOSEKCALL(res, MSK_getxxslice(task, MSK_SOL_ITR, offsetx + j, offsetx + j + 1, &xj));
        expret += mu[j] * xj;
    }
}

```



```

const double    w        = 1.0,
                x0[]      = {0.0, 0.0, 0.0},
                gamma     = 0.05,
                mu[]       = {0.1073, 0.0737, 0.0627},
                f[]        = {0.01, 0.01, 0.01},
                g[]        = {0.001, 0.001, 0.001},
                GT[][3]    = {{0.1667, 0.0232, 0.0013},
                              {0.0000, 0.1033, -0.0022},
                              {0.0000, 0.0000, 0.0338}
                              };

double          U,
                expret,
                stddev,
                xj;
MSKenv_t        env;
MSKint32t       k, i, j,
                offsetx, offsets, offsett, offsetz, offsety;
MSKrescodee     res = MSK_RES_OK, trmcode;
MSKtask_t       task;

/* Initial setup. */
env = NULL;
task = NULL;
MOSEKCALL(res, MSK_makeenv(&env, NULL));
MOSEKCALL(res, MSK_maketask(env, 0, 0, &task));
MOSEKCALL(res, MSK_linkfunctotaskstream(task, MSK_STREAM_LOG, NULL, printstr));

/* Compute total wealth */
U = w;
for (k = 0; k < n; ++k) U += x0[k];

/* Constraints. */
MOSEKCALL(res, MSK_appendcons(task, 1 + 4 * n));
MOSEKCALL(res, MSK_putconbound(task, 0, MSK_BK_FX, w, w));
sprintf(buf, "%s", "budget");
MOSEKCALL(res, MSK_putconname(task, 0, buf));

for (i = 0; i < n; ++i)
{
    MOSEKCALL(res, MSK_putconbound(task, 1 + i, MSK_BK_FX, 0.0, 0.0));
    sprintf(buf, "GT[%d]", 1 + i);
    MOSEKCALL(res, MSK_putconname(task, 1 + i, buf));
}

for (i = 0; i < n; ++i)
{
    MOSEKCALL(res, MSK_putconbound(task, 1 + n + i, MSK_BK_LO, -x0[i], MSK_INFINITY));
    sprintf(buf, "zabs1[%d]", 1 + i);
    MOSEKCALL(res, MSK_putconname(task, 1 + n + i, buf));
}

for (i = 0; i < n; ++i)
{
    MOSEKCALL(res, MSK_putconbound(task, 1 + 2 * n + i, MSK_BK_LO, x0[i], MSK_INFINITY));
    sprintf(buf, "zabs2[%d]", 1 + i);
    MOSEKCALL(res, MSK_putconname(task, 1 + 2 * n + i, buf));
}

for (i = 0; i < n; ++i)
{
    MOSEKCALL(res, MSK_putconbound(task, 1 + 3 * n + i, MSK_BK_UP, -MSK_INFINITY, 0.0));

```

(continues on next page)

(continued from previous page)

```
MOSEKCALL(res, MSK_putvarbound(task, offsety + j, MSK_BK_RA, 0.0, 1.0));
MOSEKCALL(res, MSK_putvartype(task, offsety + j, MSK_VAR_TYPE_INT));
sprintf(buf, "y[%d]", 1 + j);
MOSEKCALL(res, MSK_putvarname(task, offsety + j, buf));
}

if (res == MSK_RES_OK)
{
    /* sub should be n+1 long i.e. the dimension of the cone. */
    MSKint32t *sub = (MSKint32t *) MSK_calloc(task, 3 * (n + 1), sizeof(MSKint32t));
    if (sub)
    {
        /* Add quadratic cone */
        sub[0] = offsety + 0;
        for (j = 0; j < n; ++j)
            sub[j + 1] = offsety + j;

        MOSEKCALL(res, MSK_appendcone(task, MSK_CT_QUAD, 0.0, n + 1, sub));
        MOSEKCALL(res, MSK_putconename(task, 0, "stddev"));

        MSK_freetask(task, sub);
    }
    else
        res = MSK_RES_ERR_SPACE;
}

MOSEKCALL(res, MSK_putobjsense(task, MSK_OBJECTIVE_SENSE_MAXIMIZE));

#ifdef NO_LOG
    /* no log output. */
    MOSEKCALL(res, MSK_putintparam(task, MSK_IPAR_LOG, 0));
#endif

#ifdef DUMP_OPF
    /* Dump the problem to a human readable OPF file. */
    MOSEKCALL(res, MSK_writedata(task, "dump.opf"));
#endif

MOSEKCALL(res, MSK_optimizetrm(task, &trmcode));

#ifdef DISPLAY_RESULTS
    /* Display the solution summary for quick inspection of results. */
    MSK_solutionsummary(task, MSK_STREAM_MSG);
#endif

if (res == MSK_RES_OK)
{
    expret = 0.0;
    stddev = 0.0;

    for (j = 0; j < n; ++j)
    {
        MOSEKCALL(res, MSK_getxxslice(task, MSK_SOL_ITG, offsetx + j, offsetx + j + 1, &xj));
        expret += mu[j] * xj;
    }

    MOSEKCALL(res, MSK_getxxslice(task, MSK_SOL_ITG, offsety + 0, offsety + 1, &stddev));

    printf("\nExpected return %e for gamma %e\n", expret, stddev);
}
```

(continues on next page)

The next step is to consider how the linear constraint matrix A and the remaining data vectors are laid out. Reusing the idea in [Sec. 11.1.1](#) we can write the data in block matrix form and read off all the required coordinates. This extension of the code setting up the constraint $G^T x - t = 0$ from [Sec. 11.1.1](#) is shown below.

Example code

The following example code demonstrates how to compute an optimal portfolio with cardinality bounds. Note that we are now solving a problem with integer variables, and therefore the solution must be retrieved from `MSK_SOL_ITG`.

Listing 11.9: Code solving problem (11.20).

```
MSKrescodee markowitz_with_card(const int      n,
                                const double   x0[],
                                const double   w,
                                const double   gamma,
                                const double   mu[],
                                const double   GT[],
                                const int      k,
                                double         *xx)
{
    char          buf[128];
    double        U;
    MSKenv_t      env;
    MSKint32t     i, j,
                offsetx, offsety, offsetz, offsetw, offsetv;
    MSKrescodee   res = MSK_RES_OK, trmcode;
    MSKtask_t     task;

    /* Initial setup. */
    env = NULL;
    task = NULL;
    MOSEKCALL(res, MSK_makeenv(&env, NULL));
    MOSEKCALL(res, MSK_maketask(env, 0, 0, &task));
    MOSEKCALL(res, MSK_linkfunctotaskstream(task, MSK_STREAM_LOG, NULL, printstr));

    /* Compute total wealth */
    U = w;
    for (i = 0; i < n; ++i) U += x0[i];

    /* Constraints. */
    MOSEKCALL(res, MSK_appendcons(task, 2 + 4 * n));

    MOSEKCALL(res, MSK_putconbound(task, 0, MSK_BK_FX, w, w));
    sprintf(buf, "%s", "budget");
    MOSEKCALL(res, MSK_putconname(task, 0, buf));

    MOSEKCALL(res, MSK_putconbound(task, 1 + 4 * n, MSK_BK_UP, -MSK_INFINITY, (double)k));
    sprintf(buf, "%s", "cardinality");
    MOSEKCALL(res, MSK_putconname(task, 1 + 4 * n, buf));

    for (i = 0; i < n; ++i)
    {
        MOSEKCALL(res, MSK_putconbound(task, 1 + i, MSK_BK_FX, 0.0, 0.0));
        sprintf(buf, "GT[%d]", 1 + i);
        MOSEKCALL(res, MSK_putconname(task, 1 + i, buf));
    }

    for (i = 0; i < n; ++i)
    {
        MOSEKCALL(res, MSK_putconbound(task, 1 + n + i, MSK_BK_LO, -x0[i], MSK_INFINITY));
    }
}
```

(continues on next page)

(continued from previous page)

```
bestObj = (sense == MSK_OBJECTIVE_SENSE_MINIMIZE) ? 1.0e+10 : -1.0e+10;

for (int i = 0; i < n; ++i) {
    double priObj;
    MSK_getprimalobj(tasks[i], MSK_SOL_ITG, &priObj);
    printf("%d\t%f\n", i, priObj);
}

for (int i = 0; i < n; ++i) {
    double priObj;
    MSKsolstae solsta;
    MSK_getprimalobj(tasks[i], MSK_SOL_ITG, &priObj);
    MSK_getsolsta(tasks[i], MSK_SOL_ITG, &solsta);

    if ((res[i] == MSK_RES_OK) &&
        (solsta == MSK_SOL_STA_PRIM_FEAS ||
         solsta == MSK_SOL_STA_INTEGER_OPTIMAL) &&
        ((sense == MSK_OBJECTIVE_SENSE_MINIMIZE) ?
         (priObj < bestObj) : (priObj > bestObj)) ) )
    {
        bestObj = priObj;
        bestPos = i;
    }
}

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

// Cleanup
for (int i = 0; i < n; ++i)
    if (i != bestPos) MSK_deletetask(&(tasks[i]));

delete[] tasks; delete[] res; delete[] trm;
return bestPos;
}
```

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

Listing 11.17: Calling concurrent integer optimization (C++).

```
int seeds[3] = { 42, 13, 71749373 };

idx = optimizeconcurrentMIO(task, 3, seeds, &t, &trm, &res);
```


(continued from previous page)

Number of branches	: 4425
Number of relaxations solved	: 4410
Number of interior point iterations	: 25
Number of simplex iterations	: 221131

The first lines contain a summary of the problem as seen by the optimizer. This is followed by the iteration log. The columns have the following meaning:

- BRANCHES: Number of branches generated.
- RELAXS: Number of relaxations solved.
- ACT_NDS: Number of active branch bound nodes.
- DEPTH: Depth of the recently solved node.
- BEST_INT_OBJ: The best integer objective value, \bar{z} .
- BEST_RELAX_OBJ: The best objective bound, \underline{z} .
- REL_GAP(%): Relative optimality gap, $100\% \cdot \epsilon_{\text{rel}}$
- TIME: Time (in seconds) from the start of optimization.

Following that a summary of the optimization process is printed.

(continued from previous page)

```
if (r == MSK_RES_OK)
    r = MSK_makeemptytask(env, &task);

if (r == MSK_RES_OK)
    MSK_linkfunctotaskstream(task, MSK_STREAM_LOG, NULL, printstr);

if (r == MSK_RES_OK)
    r = MSK_readlpstring(task,feasrepair_lp); /* Read problem from string */

if (r == MSK_RES_OK)
    r = MSK_putintparam(task, MSK_IPAR_LOG_FEAS_REPAIR, 3);

if (r == MSK_RES_OK)
{
    /* Weights are NULL implying all weights are 1. */
    r = MSK_primalrepair(task, NULL, NULL, NULL, NULL);
}

if (r == MSK_RES_OK)
{
    double sum_viol;

    r = MSK_getdouinf(task, MSK_DINF_PRIMAL_REPAIR_PENALTY_OBJ, &sum_viol);

    if (r == MSK_RES_OK)
    {
        printf("Minimized sum of violations = %e\n", sum_viol);

        r = MSK_optimizetrm(task, &trmcode); /* Optimize the repaired task. */

        MSK_solutionsummary(task, MSK_STREAM_MSG);
    }
}

printf("Return code: %d\n", r);

return (r);
}
```

The above code will produce the following log report:

```
MOSEK Version 9.0.0.25(ALPHA) (Build date: 2017-11-7 16:11:50)
Copyright (c) MOSEK ApS, Denmark. WWW: mosek.com
Platform: Linux/64-X86

Open file 'feasrepair.lp'
Reading started.
Reading terminated. Time: 0.00

Read summary
  Type           : LO (linear optimization problem)
  Objective sense : min
  Scalar variables : 2
  Matrix variables : 0
  Constraints      : 4
  Cones           : 0
  Time            : 0.0

Problem
  Name           :
  Objective sense : min
  Type           : LO (linear optimization problem)
```

(continues on next page)


```

    r = MSK_appendvars(task, numvar);

    /* Put C. */
    if (r == MSK_RES_OK)
        r = MSK_putcfix(task, 0.0);

    if (r == MSK_RES_OK)
        r = MSK_putcslice(task, 0, numvar, c);

    /* Put constraint bounds. */
    if (r == MSK_RES_OK)
        r = MSK_putconboundslice(task, 0, numcon, bkc, blc, buc);

    /* Put variable bounds. */
    if (r == MSK_RES_OK)
        r = MSK_putvarboundslice(task, 0, numvar, bkx, blx, bux);

    /* Put A. */
    if (r == MSK_RES_OK)
        r = MSK_putacolslice(task, 0, numvar, ptrb, ptre, sub, val);

    if (r == MSK_RES_OK)
        r = MSK_putobjsense(task, MSK_OBJECTIVE_SENSE_MINIMIZE);

    if (r == MSK_RES_OK)
        r = MSK_optimizetrm(task, NULL);

    if (r == MSK_RES_OK)
    {
        /* Analyze upper bound on c1 and the equality constraint on c4 */
        MSKint32t subi[] = {0, 3};
        MSKmarke marki[] = {MSK_MARK_UP, MSK_MARK_UP};

        /* Analyze lower bound on the variables x12 and x31 */
        MSKint32t subj[] = {1, 4};
        MSKmarke markj[] = {MSK_MARK_LO, MSK_MARK_LO};

        MSKrealt leftpricei[2];
        MSKrealt rightpricei[2];
        MSKrealt leftrangei[2];
        MSKrealt rightrangei[2];
        MSKrealt leftpricej[2];
        MSKrealt rightpricej[2];
        MSKrealt leftrangej[2];
        MSKrealt rightrangej[2];

        r = MSK_primalsensitivity(task,
                                2,
                                subi,
                                marki,
                                2,
                                subj,
                                markj,
                                leftpricei,
                                rightpricei,
                                leftrangei,
                                rightrangei,
                                leftpricej,
                                rightpricej,
                                leftrangej,
                                rightrangej);
    }

```

```

printf("Results from sensitivity analysis on bounds:\n");

printf("For constraints:\n");
for (i = 0; i < 2; ++i)
    printf("leftprice = %e, rightprice = %e, leftrange = %e, rightrange = %e\n",
           leftpricei[i], rightpricei[i], leftrangei[i], rightrangei[i]);

printf("For variables:\n");
for (i = 0; i < 2; ++i)
    printf("leftprice = %e, rightprice = %e, leftrange = %e, rightrange = %e\n",
           leftpricej[i], rightpricej[i], leftrangej[i], rightrangej[i]);
}

if (r == MSK_RES_OK)
{
    MSKint32t subj[] = {2, 5};
    MSKrealt leftprice[2];
    MSKrealt rightprice[2];
    MSKrealt leftrange[2];
    MSKrealt rightrange[2];

    r = MSK_dualsensitivity(task,
                           2,
                           subj,
                           leftprice,
                           rightprice,
                           leftrange,
                           rightrange
                           );

    printf("Results from sensitivity analysis on objective coefficients:\n");

    for (i = 0; i < 2; ++i)
        printf("leftprice = %e, rightprice = %e, leftrange = %e, rightrange = %e\n",
               leftprice[i], rightprice[i], leftrange[i], rightrange[i]);
}

MSK_deletetask(&task);
}

MSK_deleteenv(&env);

printf("Return code: %d (0 means no error occurred.)\n", r);
return (r);
} /* main */

```

Chapter 15

API Reference

This section contains the complete reference of the **MOSEK** Optimizer API for C. It is organized as follows:

- *General API conventions.*
- **Functions:**
 - *Full list*
 - *Browse by topic*
- **Optimizer parameters:**
 - *Double, Integer, String*
 - *Full list*
 - *Browse by topic*
- **Optimizer information items:**
 - *Double, Integer, Long*
- *Optimizer response codes*
- *Constants*
- *User-defined function types*
- *Simple data types*
- *Nonlinear API (SCopt, DGopt, EXPopt)*

15.1 API Conventions

15.1.1 Function arguments

Naming Convention

In the definition of the **MOSEK** Optimizer API for C a consistent naming convention has been used. This implies that whenever for example `numcon` is an argument in a function definition it indicates the number of constraints. In Table 15.1 the variable names used to specify the problem parameters are listed.

15.2 Functions grouped by topic

Callback

- *MSK_linkfunctotaskstream* – Connects a user-defined function to a task stream.
- *MSK_putcallbackfunc* – Input the progress callback function.
- Infrequent: *MSK_getcallbackfunc*, *MSK_linkfunctoenvstream*, *MSK_putexitfunc*, *MSK_putresponsefunc*, *MSK_unlinkfuncfromenvstream*, *MSK_unlinkfuncfromtaskstream*

Environment and task management

- *MSK_clonetask* – Creates a clone of an existing task.
- *MSK_deleteenv* – Delete a MOSEK environment.
- *MSK_deletetask* – Deletes a task.
- *MSK_makeenv* – Creates a new MOSEK environment.
- *MSK_maketask* – Creates a new task.
- *MSK_puttaskname* – Assigns a new name to the task.
- Infrequent: *MSK_commitchanges*, *MSK_deletesolution*, *MSK_getenv*, *MSK_makeemptytask*, *MSK_makeenvalloc*, *MSK_putmaxnumanz*, *MSK_putmaxnumbarvar*, *MSK_putmaxnumcon*, *MSK_putmaxnumcone*, *MSK_putmaxnumqnz*, *MSK_putmaxnumvar*, *MSK_resizetask*

Infeasibility diagnostic

- *MSK_getinfeasiblesubproblem* – Obtains an infeasible subproblem.
- *MSK_primalrepair* – Repairs a primal infeasible optimization problem by adjusting the bounds on the constraints and variables.

Information items and statistics

- *MSK_getdouinfo* – Obtains a double information item.
- *MSK_getintinfo* – Obtains an integer information item.
- *MSK_getlintinfo* – Obtains a long integer information item.
- *MSK_updatesolutioninfo* – Update the information items related to the solution.
- Infrequent: *MSK_getinfindex*, *MSK_getinfmax*, *MSK_getinfname*, *MSK_getnadouinfo*, *MSK_getnaintinfo*

Input/Output

- *MSK_readdata* – Reads problem data from a file.
- *MSK_writedata* – Writes problem data to a file.
- *MSK_writesolution* – Write a solution to a file.
- Infrequent: *MSK_echotask*, *MSK_readdataautoformat*, *MSK_readdataformat*, *MSK_readjsonstring*, *MSK_readlpstring*, *MSK_readopfstring*, *MSK_readparamfile*, *MSK_readptfstring*, *MSK_readsolution*, *MSK_readsummary*, *MSK_readtask*, *MSK_writejsonsol*, *MSK_writeparamfile*, *MSK_writetask*

Problem data - linear part

- *MSK_appendcons* – Appends a number of constraints to the optimization task.
- *MSK_appendvars* – Appends a number of variables to the optimization task.
- *MSK_getnumcon* – Obtains the number of constraints.
- *MSK_putacol* – Replaces all elements in one column of the linear constraint matrix.
- *MSK_putacolslice* – Replaces all elements in a sequence of columns the linear constraint matrix.
- *MSK_putacolslice64* – Replaces all elements in a sequence of columns the linear constraint matrix.
- *MSK_putaij* – Changes a single value in the linear coefficient matrix.
- *MSK_putaijlist* – Changes one or more coefficients in the linear constraint matrix.
- *MSK_putaijlist64* – Changes one or more coefficients in the linear constraint matrix.
- *MSK_putarow* – Replaces all elements in one row of the linear constraint matrix.
- *MSK_putarowslice* – Replaces all elements in several rows the linear constraint matrix.
- *MSK_putarowslice64* – Replaces all elements in several rows the linear constraint matrix.
- *MSK_putcfix* – Replaces the fixed term in the objective.
- *MSK_putcj* – Modifies one linear coefficient in the objective.
- *MSK_putconbound* – Changes the bound for one constraint.
- *MSK_putconboundslice* – Changes the bounds for a slice of the constraints.
- *MSK_putconname* – Sets the name of a constraint.
- *MSK_putcslice* – Modifies a slice of the linear objective coefficients.
- *MSK_putobjname* – Assigns a new name to the objective.
- *MSK_putobjsense* – Sets the objective sense.
- *MSK_putvarbound* – Changes the bounds for one variable.
- *MSK_putvarboundslice* – Changes the bounds for a slice of the variables.
- *MSK_putvarname* – Sets the name of a variable.
- *MSK_removecons* – Removes a number of constraints.
- *MSK_removevars* – Removes a number of variables.
- *Infrequent:* *MSK_chgconbound*, *MSK_chgvarbound*, *MSK_generateconnames*, *MSK_generatevarnames*, *MSK_getacol*, *MSK_getacolnumnz*, *MSK_getacolslice*, *MSK_getacolslice64*, *MSK_getacolslicenumnz*, *MSK_getacolslicenumnz64*, *MSK_getacolslicetrip*, *MSK_getaij*, *MSK_getapiecenumnz*, *MSK_getarow*, *MSK_getarownumnz*, *MSK_getarowslice*, *MSK_getarowslice64*, *MSK_getarowslicenumnz*, *MSK_getarowslicenumnz64*, *MSK_getarowslicetrip*, *MSK_getatruncatetol*, *MSK_getc*, *MSK_getcfix*, *MSK_getcj*, *MSK_getclist*, *MSK_getconbound*, *MSK_getconboundslice*, *MSK_getconname*, *MSK_getconnameindex*, *MSK_getconnamelen*, *MSK_getcslice*, *MSK_getmaxnumanz*, *MSK_getmaxnumanz64*, *MSK_getmaxnumcon*, *MSK_getmaxnumvar*, *MSK_getnumanz*, *MSK_getnumanz64*, *MSK_getobjsense*, *MSK_getvarbound*, *MSK_getvarboundslice*, *MSK_getvarname*, *MSK_getvarnameindex*, *MSK_getvarnamelen*, *MSK_inputdata*, *MSK_inputdata64*, *MSK_putacollist*, *MSK_putacollist64*, *MSK_putarowlist*, *MSK_putarowlist64*, *MSK_putatruncatetol*, *MSK_putclist*, *MSK_putconboundlist*, *MSK_putconboundlistconst*, *MSK_putconboundsliceconst*, *MSK_putvarboundlist*, *MSK_putvarboundlistconst*, *MSK_putvarboundsliceconst*

Problem data - objective

- *MSK_putbarcj* – Changes one element in barc.
- *MSK_putcfix* – Replaces the fixed term in the objective.
- *MSK_putcj* – Modifies one linear coefficient in the objective.
- *MSK_putcslice* – Modifies a slice of the linear objective coefficients.
- *MSK_putobjname* – Assigns a new name to the objective.
- *MSK_putobjsense* – Sets the objective sense.
- *MSK_putqobj* – Replaces all quadratic terms in the objective.
- *MSK_putqobjij* – Replaces one coefficient in the quadratic term in the objective.
- Infrequent: *MSK_putclist*

Problem data - quadratic part

- *MSK_putqcon* – Replaces all quadratic terms in constraints.
- *MSK_putqconk* – Replaces all quadratic terms in a single constraint.
- *MSK_putqobj* – Replaces all quadratic terms in the objective.
- *MSK_putqobjij* – Replaces one coefficient in the quadratic term in the objective.
- Infrequent: *MSK_getmaxnumqnz*, *MSK_getmaxnumqnz64*, *MSK_getnumqconknz*, *MSK_getnumqconknz64*, *MSK_getnumqobjnz*, *MSK_getnumqobjnz64*, *MSK_getqconk*, *MSK_getqconk64*, *MSK_getqobj*, *MSK_getqobj64*, *MSK_getqobjij*, *MSK_putmaxnumqnz*, *MSK_toconic*

Problem data - semidefinite

- *MSK_appendbarvars* – Appends semidefinite variables to the problem.
- *MSK_appendsparsesymmat* – Appends a general sparse symmetric matrix to the storage of symmetric matrices.
- *MSK_appendsparsesymmatlist* – Appends a general sparse symmetric matrix to the storage of symmetric matrices.
- *MSK_putbaraij* – Inputs an element of barA.
- *MSK_putbaraijlist* – Inputs list of elements of barA.
- *MSK_putbararowlist* – Replace a set of rows of barA.
- *MSK_putbarcj* – Changes one element in barc.
- *MSK_putbarvarname* – Sets the name of a semidefinite variable.
- Infrequent: *MSK_getbarablocktriplet*, *MSK_getbaraidx*, *MSK_getbaraidxij*, *MSK_getbaraidxinfo*, *MSK_getbarasparsity*, *MSK_getbarcblocktriplet*, *MSK_getbarcidx*, *MSK_getbarcidxinfo*, *MSK_getbarcidxj*, *MSK_getbarcsparsity*, *MSK_getdimbarvarj*, *MSK_getlenbarvarj*, *MSK_getmaxnumbarvar*, *MSK_getnumbarablocktriplets*, *MSK_getnumbaranz*, *MSK_getnumbarcblocktriplets*, *MSK_getnumbarcnz*, *MSK_getnumbarvar*, *MSK_getnumsymmat*, *MSK_getsparsesymmat*, *MSK_getsymmatinfo*, *MSK_putbarablocktriplet*, *MSK_putbarcblocktriplet*, *MSK_putmaxnumanz*, *MSK_putmaxnumbarvar*, *MSK_removebarvars*

Problem data - variables

- *MSK_appendvars* – Appends a number of variables to the optimization task.
- *MSK_getnumvar* – Obtains the number of variables.
- *MSK_putvarbound* – Changes the bounds for one variable.
- *MSK_putvarboundslice* – Changes the bounds for a slice of the variables.
- *MSK_putvarname* – Sets the name of a variable.
- *MSK_putvartype* – Sets the variable type of one variable.
- *MSK_removevars* – Removes a number of variables.
- Infrequent: *MSK_chgvarbound*, *MSK_generatevarnames*, *MSK_getc*, *MSK_getcj*, *MSK_getmaxnumvar*, *MSK_getnumintvar*, *MSK_getvarbound*, *MSK_getvarboundslice*, *MSK_getvarname*, *MSK_getvarnameindex*, *MSK_getvarnamelen*, *MSK_getvartype*, *MSK_getvartypelist*, *MSK_putclist*, *MSK_putmaxnumvar*, *MSK_putvarboundlist*, *MSK_putvarboundlistconst*, *MSK_putvarboundsliceconst*, *MSK_putvartypelist*

Remote optimization

- *MSK_asyncgetresult* – Request a response from a remote job.
- *MSK_asyncoptimize* – Offload the optimization task to a solver server.
- *MSK_asyncpoll* – Requests information about the status of the remote job.
- *MSK_asyncstop* – Request that the job identified by the token is terminated.
- *MSK_optimizermt* – Offload the optimization task to a solver server.

Responses, errors and warnings

- *MSK_getcodedesc* – Obtains a short description of a response code.
- Infrequent: *MSK_getlasterror*, *MSK_getlasterror64*, *MSK_getresponseclass*

Sensitivity analysis

- *MSK_dualsensitivity* – Performs sensitivity analysis on objective coefficients.
- *MSK_primalsensitivity* – Perform sensitivity analysis on bounds.
- *MSK_sensitivityreport* – Creates a sensitivity report.

Solution - dual

- *MSK_getdualobj* – Computes the dual objective value associated with the solution.
- *MSK_gety* – Obtains the y vector for a solution.
- *MSK_getyslice* – Obtains a slice of the y vector for a solution.
- Infrequent: *MSK_getreducedcosts*, *MSK_getslc*, *MSK_getslcslice*, *MSK_getslx*, *MSK_getslxslice*, *MSK_getsnx*, *MSK_getsnxslice*, *MSK_getsolution*, *MSK_getsolutionslice*, *MSK_getsuc*, *MSK_getsucslice*, *MSK_getsux*, *MSK_getsuxslice*, *MSK_putconsolutioni*, *MSK_putslc*, *MSK_putslcslice*, *MSK_putslx*, *MSK_putslxslice*, *MSK_putsnx*, *MSK_putsnxslice*, *MSK_putsolution*, *MSK_putsolutionyi*, *MSK_putsuc*, *MSK_putsucslice*, *MSK_putsux*, *MSK_putsuxslice*, *MSK_putvarsolutionj*, *MSK_putyslice*

Solution - primal

- *MSK_getprimalobj* – Computes the primal objective value for the desired solution.
- *MSK_getxx* – Obtains the xx vector for a solution.
- *MSK_getxxslice* – Obtains a slice of the xx vector for a solution.
- *MSK_putxx* – Sets the xx vector for a solution.
- *MSK_putxxslice* – Sets a slice of the xx vector for a solution.
- Infrequent: *MSK_getsolution*, *MSK_getsolutionslice*, *MSK_getxc*, *MSK_getxcslice*, *MSK_putconsolutioni*, *MSK_putsolution*, *MSK_putvarsolutionj*, *MSK_putxc*, *MSK_putxcslice*, *MSK_puty*

Solution - semidefinite

- *MSK_getbarsj* – Obtains the dual solution for a semidefinite variable.
- *MSK_getbarsslice* – Obtains the dual solution for a sequence of semidefinite variables.
- *MSK_getbarxj* – Obtains the primal solution for a semidefinite variable.
- *MSK_getbarxslice* – Obtains the primal solution for a sequence of semidefinite variables.
- Infrequent: *MSK_putbarsj*, *MSK_putbarxj*

Solution information

- *MSK_getdualobj* – Computes the dual objective value associated with the solution.
- *MSK_getprimalobj* – Computes the primal objective value for the desired solution.
- *MSK_getprosta* – Obtains the problem status.
- *MSK_getpviolcon* – Computes the violation of a primal solution associated to a constraint.
- *MSK_getpviolvar* – Computes the violation of a primal solution for a list of scalar variables.
- *MSK_getsolsta* – Obtains the solution status.
- *MSK_getsolutioninfo* – Obtains information about of a solution.
- *MSK_onesolutionsummary* – Prints a short summary of a specified solution.
- *MSK_solutiondef* – Checks whether a solution is defined.
- *MSK_solutionsummary* – Prints a short summary of the current solutions.
- Infrequent: *MSK_analyzesolution*, *MSK_deletesolution*, *MSK_getdualsolutionnorms*, *MSK_getdviolbarvar*, *MSK_getdviolcon*, *MSK_getdviolcones*, *MSK_getdviolvar*, *MSK_getprimalsolutionnorms*, *MSK_getpviolbarvar*, *MSK_getpviolcones*, *MSK_getskc*, *MSK_getskcslice*, *MSK_getskn*, *MSK_getskx*, *MSK_getskxslice*, *MSK_getsolution*, *MSK_getsolutionslice*, *MSK_prostatostr*, *MSK_putconsolutioni*, *MSK_putskc*, *MSK_putskcslice*, *MSK_putskx*, *MSK_putskxslice*, *MSK_putsolution*, *MSK_putsolutionyi*, *MSK_putvarsolutionj*

Solving systems with basis matrix

- Infrequent: *MSK_basiscond*, *MSK_initbasissolve*, *MSK_solvewithbasis*

System, memory and debugging

- *Infrequent:* `MSK_callocdbgenv`, `MSK_callocdbgtask`, `MSK_callocenv`, `MSK_calloctask`, `MSK_checkmemenv`, `MSK_checkmentask`, `MSK_freedbgenv`, `MSK_freedbgtask`, `MSK_freeenv`, `MSK_freetask`, `MSK_getmemusagetask`, `MSK_setupthreads`, `MSK_strdupdbgtask`, `MSK_strduptask`, `MSK_utf8towchar`, `MSK_wchartoutf8`

Versions

- `MSK_getversion` – Obtains MOSEK version information.
- *Infrequent:* `MSK_checkversion`, `MSK_getbuildinfo`

Other

- *Infrequent:* `MSK_isinfinity`

15.3 Functions in alphabetical order

MSK_analyzenames

```
MSKrescodee (MSKAPI MSK_analyzenames) (  
    MSKtask_t task,  
    MSKstreamtypee whichstream,  
    MSKnametypee nametype)
```

The function analyzes the names and issues an error if a name is invalid.

Parameters

- `task` (`MSKtask_t`) – An optimization task. (input)
- `whichstream` (`MSKstreamtypee`) – Index of the stream. (input)
- `nametype` (`MSKnametypee`) – The type of names e.g. valid in MPS or LP files. (input)

Return (`MSKrescodee`) – The function response code.

Groups *Names*

MSK_analyzeproblem

```
MSKrescodee (MSKAPI MSK_analyzeproblem) (  
    MSKtask_t task,  
    MSKstreamtypee whichstream)
```

The function analyzes the data of a task and writes out a report.

Parameters

- `task` (`MSKtask_t`) – An optimization task. (input)
- `whichstream` (`MSKstreamtypee`) – Index of the stream. (input)

Return (`MSKrescodee`) – The function response code.

Groups *Inspecting the task*

MSK_analyzesolution

```
MSKrescodee (MSKAPI MSK_analyzesolution) (  
    MSKtask_t task,  
    MSKstreamtypee whichstream,  
    MSKsoltypee whichsol)
```


MSK_appendsparsesymmat

```
MSKrescodee (MSKAPI MSK_appendsparsesymmat) (  
    MSKtask_t task,  
    MSKint32t dim,  
    MSKint64t nz,  
    const MSKint32t * subi,  
    const MSKint32t * subj,  
    const MSKrealt * valij,  
    MSKint64t * idx)
```

MOSEK maintains a storage of symmetric data matrices that is used to build \bar{C} and \bar{A} . The storage can be thought of as a vector of symmetric matrices denoted E . Hence, E_i is a symmetric matrix of certain dimension.

This function appends a general sparse symmetric matrix on triplet form to the vector E of symmetric matrices. The vectors `subi`, `subj`, and `valij` contains the row subscripts, column subscripts and values of each element in the symmetric matrix to be appended. Since the matrix that is appended is symmetric, only the lower triangular part should be specified. Moreover, duplicates are not allowed.

Observe the function reports the index (position) of the appended matrix in E . This index should be used for later references to the appended matrix.

Parameters

- `task` (*MSKtask_t*) – An optimization task. (input)
- `dim` (*MSKint32t*) – Dimension of the symmetric matrix that is appended. (input)
- `nz` (*MSKint64t*) – Number of triplets. (input)
- `subi` (*MSKint32t**) – Row subscript in the triplets. (input)
- `subj` (*MSKint32t**) – Column subscripts in the triplets. (input)
- `valij` (*MSKrealt**) – Values of each triplet. (input)
- `idx` (*MSKint64t by reference*) – Unique index assigned to the inputted matrix that can be used for later reference. (output)

Return (*MSKrescodee*) – The function response code.

Groups *Problem data - semidefinite*

MSK_appendsparsesymmatlist

```
MSKrescodee (MSKAPI MSK_appendsparsesymmatlist) (  
    MSKtask_t task,  
    MSKint32t num,  
    const MSKint32t * dims,  
    const MSKint64t * nz,  
    const MSKint32t * subi,  
    const MSKint32t * subj,  
    const MSKrealt * valij,  
    MSKint64t * idx)
```

MOSEK maintains a storage of symmetric data matrices that is used to build \bar{C} and \bar{A} . The storage can be thought of as a vector of symmetric matrices denoted E . Hence, E_i is a symmetric matrix of certain dimension.

This function appends general sparse symmetric matrixes on triplet form to the vector E of symmetric matrices. The vectors `subi`, `subj`, and `valij` contains the row subscripts, column subscripts and values of each element in the symmetric matrix to be appended. Since the matrix that is appended is symmetric, only the lower triangular part should be specified. Moreover, duplicates are not allowed.

Observe the function reports the index (position) of the appended matrix in E . This index should be used for later references to the appended matrix.


```
void * (MSKAPI MSK_allocdbgtask) (
    MSKtask_t task,
    const size_t number,
    const size_t size,
    const char * file,
    const unsigned line)
```

Debug version of *MSK_allocctask*.

Parameters

- task (*MSKtask_t*) – An optimization task. (input)
- number (*size_t*) – Number of elements. (input)
- size (*size_t*) – Size of each individual element. (input)
- file (*MSKstring_t*) – File from which the function is called. (input)
- line (*unsigned*) – Line in the file from which the function is called. (input)

Return (*void**) – A pointer to the memory allocated through the task.

Groups *System, memory and debugging*

MSK_allocenv

```
void * (MSKAPI MSK_allocenv) (
    MSKenv_t env,
    const size_t number,
    const size_t size)
```

Equivalent to `calloc` i.e. allocate space for an array of length **number** where each element is of size **size**.

Parameters

- env (*MSKenv_t*) – The MOSEK environment. (input)
- number (*size_t*) – Number of elements. (input)
- size (*size_t*) – Size of each individual element. (input)

Return (*void**) – A pointer to the memory allocated through the environment.

Groups *System, memory and debugging*

MSK_allocctask

```
void * (MSKAPI MSK_allocctask) (
    MSKtask_t task,
    const size_t number,
    const size_t size)
```

Equivalent to `calloc` i.e. allocate space for an array of length **number** where each element is of size **size**.

Parameters

- task (*MSKtask_t*) – An optimization task. (input)
- number (*size_t*) – Number of elements. (input)
- size (*size_t*) – Size of each individual element. (input)

Return (*void**) – A pointer to the memory allocated through the task.

Groups *System, memory and debugging*

MSK_checkinall

```
MSKrescodee (MSKAPI MSK_checkinall) (
    MSKenv_t env)
```


Parameters

- **task** (*MSKtask_t*) – An optimization task. (input)
- **i** (*MSKint32t*) – Index of the constraint for which the bounds should be changed. (input)
- **lower** (*MSKint32t*) – If non-zero, then the lower bound is changed, otherwise the upper bound is changed. (input)
- **finite** (*MSKint32t*) – If non-zero, then **value** is assumed to be finite. (input)
- **value** (*MSKrealt*) – New value for the bound. (input)

Return (*MSKrescodee*) – The function response code.

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

MSK_chgvarbound

```
MSKrescodee (MSKAPI MSK_chgvarbound) (  
    MSKtask_t task,  
    MSKint32t j,  
    MSKint32t lower,  
    MSKint32t finite,  
    MSKrealt value)
```

Changes a bound for one variable.

If **lower** is non-zero, then the lower bound is changed as follows:

$$\text{new lower bound} = \begin{cases} -\infty, & \text{finite} = 0, \\ \text{value} & \text{otherwise.} \end{cases}$$

Otherwise if **lower** is zero, then

$$\text{new upper bound} = \begin{cases} \infty, & \text{finite} = 0, \\ \text{value} & \text{otherwise.} \end{cases}$$

Please note that this function automatically updates the bound key for the bound, in particular, if the lower and upper bounds are identical, the bound key is changed to **fixed**.

Parameters

- **task** (*MSKtask_t*) – An optimization task. (input)
- **j** (*MSKint32t*) – Index of the variable for which the bounds should be changed. (input)
- **lower** (*MSKint32t*) – If non-zero, then the lower bound is changed, otherwise the upper bound is changed. (input)
- **finite** (*MSKint32t*) – If non-zero, then **value** is assumed to be finite. (input)
- **value** (*MSKrealt*) – New value for the bound. (input)

Return (*MSKrescodee*) – The function response code.

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

MSK_clonetask

```
MSKrescodee (MSKAPI MSK_clonetask) (  
    MSKtask_t task,  
    MSKtask_t * clonedtask)
```

Creates a clone of an existing task copying all problem data and parameter settings to a new task. Callback functions are not copied.

Parameters

- **task** (*MSKtask_t*) – An optimization task. (input)
- **clonedtask** (*MSKtask_t by reference*) – The cloned task. (output)

Return (*MSKrescodee*) – The function response code.

Groups *Environment and task management*

MSK_commitchanges

```
MSKrescodee (MSKAPI MSK_commitchanges) (  
    MSKtask_t task)
```

Commits all cached problem changes to the task. It is usually not necessary to call this function explicitly since changes will be committed automatically when required.

Parameters *task* (*MSKtask_t*) – An optimization task. (input)

Return (*MSKrescodee*) – The function response code.

Groups *Environment and task management*

MSK_computesparscholesky

```
MSKrescodee (MSKAPI MSK_computesparscholesky) (  
    MSKenv_t env,  
    MSKint32t multithread,  
    MSKint32t ordermethod,  
    MSKrealt tolsingular,  
    MSKint32t n,  
    const MSKint32t * anzc,  
    const MSKint64t * aptrc,  
    const MSKint32t * asubc,  
    const MSKrealt * avalc,  
    MSKint32t ** perm,  
    MSKrealt ** diag,  
    MSKint32t ** lnzc,  
    MSKint64t ** lptrc,  
    MSKint64t * lensubnval,  
    MSKint32t ** lsubc,  
    MSKrealt ** lvalc)
```

The function computes a Cholesky factorization of a sparse positive semidefinite matrix. Sparsity is exploited during the computations to reduce the amount of space and work required. Both the input and output matrices are represented using the sparse format.

To be precise, given a symmetric matrix $A \in \mathbb{R}^{n \times n}$ the function computes a nonsingular lower triangular matrix L , a diagonal matrix D and a permutation matrix P such that

$$LL^T - D = PAP^T.$$

If *ordermethod* is zero then reordering heuristics are not employed and P is the identity.

If a pivot during the computation of the Cholesky factorization is less than

$$-\rho \cdot \max((PAP^T)_{jj}, 1.0)$$

then the matrix is declared negative semidefinite. On the hand if a pivot is smaller than

$$\rho \cdot \max((PAP^T)_{jj}, 1.0),$$

then D_{jj} is increased from zero to

$$\rho \cdot \max((PAP^T)_{jj}, 1.0).$$

Therefore, if A is sufficiently positive definite then D will be the zero matrix. Here ρ is set equal to value of *tolsingular*.

The function allocates memory for the output arrays. It must be freed by the user with *MSK_freeenv*.

MSK_echointro

```
MSKrescodee (MSKAPI MSK_echointro) (  
    MSKenv_t env,  
    MSKint32t longver)
```

Prints an intro to message stream.

Parameters

- `env` (*MSKenv_t*) – The MOSEK environment. (input)
- `longver` (*MSKint32t*) – If non-zero, then the intro is slightly longer. (input)

Return (*MSKrescodee*) – The function response code.

Groups *Logging*

MSK_echotask

```
MSKrescodee (MSKAPIVA MSK_echotask) (  
    MSKtask_t task,  
    MSKstreamtypee whichstream,  
    const char * format,  
    ...)
```

Prints a formatted string to a task stream.

Parameters

- `task` (*MSKtask_t*) – An optimization task. (input)
- `whichstream` (*MSKstreamtypee*) – Index of the stream. (input)
- `format` (*MSKstring_t*) – Is a valid C format string which matches the arguments in ... (input)
- `varnumarg` (...) – Additional arguments (input)

Return (*MSKrescodee*) – The function response code.

Groups *Input/Output*

MSK_freedbgenv

```
void (MSKAPI MSK_freedbgenv) (  
    MSKenv_t env,  
    void * buffer,  
    const char * file,  
    const unsigned line)
```

Frees space allocated by **MOSEK**. Debug version of *MSK_freeenv*.

Parameters

- `env` (*MSKenv_t*) – The MOSEK environment. (input)
- `buffer` (void*) – A pointer. (input/output)
- `file` (*MSKstring_t*) – File from which the function is called. (input)
- `line` (unsigned) – Line in the file from which the function is called. (input)

Return (void)

Groups *System, memory and debugging*

MSK_freedbgtask

```
void (MSKAPI MSK_freedbgtask) (  
    MSKtask_t task,  
    void * buffer,  
    const char * file,  
    const unsigned line)
```

Frees space allocated by **MOSEK**. Debug version of *MSK_freetask*.

Parameters

- **task** (*MSKtask_t*) – An optimization task. (input)
- **buffer** (*void**) – A pointer. (input/output)
- **file** (*MSKstring_t*) – File from which the function is called. (input)
- **line** (*unsigned*) – Line in the file from which the function is called. (input)

Return (*void*)

Groups *System, memory and debugging*

MSK_freeenv

```
void (MSKAPI MSK_freeenv) (  
    MSKenv_t env,  
    void * buffer)
```

Frees space allocated by a **MOSEK** function. Must not be applied to the **MOSEK** environment and task.

Parameters

- **env** (*MSKenv_t*) – The MOSEK environment. (input)
- **buffer** (*void**) – A pointer. (input/output)

Return (*void*)

Groups *System, memory and debugging*

MSK_freetask

```
void (MSKAPI MSK_freetask) (  
    MSKtask_t task,  
    void * buffer)
```

Frees space allocated by a **MOSEK** function. Must not be applied to the **MOSEK** environment and task.

Parameters

- **task** (*MSKtask_t*) – An optimization task. (input)
- **buffer** (*void**) – A pointer. (input/output)

Return (*void*)

Groups *System, memory and debugging*

MSK_gemm

```
MSKrescodee (MSKAPI MSK_gemm) (  
    MSKenv_t env,  
    MSKtransposee transa,  
    MSKtransposee transb,  
    MSKint32t m,  
    MSKint32t n,  
    MSKint32t k,  
    MSKrealt alpha,  
    const MSKrealt * a,  
    const MSKrealt * b,  
    MSKrealt beta,  
    MSKrealt * c)
```


Parameters

- `env` (*MSKenv_t*) – The MOSEK environment. (input)
- `transa` (*MSKtransposee*) – Indicates whether the matrix A must be transposed. (input)
- `m` (*MSKint32t*) – Specifies the number of rows of the matrix A . (input)
- `n` (*MSKint32t*) – Specifies the number of columns of the matrix A . (input)
- `alpha` (*MSKrealt*) – A scalar value multiplying the matrix A . (input)
- `a` (*MSKrealt**) – A pointer to the array storing matrix A in a column-major format. (input)
- `x` (*MSKrealt**) – A pointer to the array storing the vector x . (input)
- `beta` (*MSKrealt*) – A scalar value multiplying the vector y . (input)
- `y` (*MSKrealt**) – A pointer to the array storing the vector y . (input/output)

Return (*MSKrescodee*) – The function response code.

Groups *Linear algebra*

MSK_generateconenames

```
MSKrescodee (MSKAPI MSK_generateconenames) (  
    MSKtask_t task,  
    MSKint32t num,  
    const MSKint32t * subk,  
    const char * fmt,  
    MSKint32t ndims,  
    const MSKint32t * dims,  
    const MSKint64t * sp)
```

Generates systematic names for cone.

Parameters

- `task` (*MSKtask_t*) – An optimization task. (input)
- `num` (*MSKint32t*) – Number of cone indexes. (input)
- `subk` (*MSKint32t**) – Indexes of the cone. (input)
- `fmt` (*MSKstring_t*) – The cone name formatting string. (input)
- `ndims` (*MSKint32t*) – Number of dimensions in the shape. (input)
- `dims` (*MSKint32t**) – Dimensions in the shape. (input)
- `sp` (*MSKint64t**) – Items that should be named. (input)

Return (*MSKrescodee*) – The function response code.

Groups *Names, Problem data - cones*

MSK_generateconnames

```
MSKrescodee (MSKAPI MSK_generateconnames) (  
    MSKtask_t task,  
    MSKint32t num,  
    const MSKint32t * subi,  
    const char * fmt,  
    MSKint32t ndims,  
    const MSKint32t * dims,  
    const MSKint64t * sp)
```

Generates systematic names for constraints.

Parameters

- `task` (*MSKtask_t*) – An optimization task. (input)
- `num` (*MSKint32t*) – Number of constraint indexes. (input)
- `subi` (*MSKint32t**) – Indexes of the constraints. (input)

- `last` (*MSKint32t*) – Index of the last column **plus one** in the sequence. (input)
 - `numnz` (*MSKint64t by reference*) – Number of non-zeros in the slice. (output)
- Return** (*MSKrescodee*) – The function response code.
- Groups** *Problem data - linear part, Inspecting the task*

MSK_getacolslicetrip

```
MSKrescodee (MSKAPI MSK_getacolslicetrip) (
    MSKtask_t task,
    MSKint32t first,
    MSKint32t last,
    MSKint64t maxnumnz,
    MSKint64t * surp,
    MSKint32t * subi,
    MSKint32t * subj,
    MSKrealt * val)
```

Obtains a sequence of columns from A in sparse triplet format. The function returns the content of all columns whose index j satisfies $\text{first} \leq j < \text{last}$. The triplets corresponding to nonzero entries are stored in the arrays `subi`, `subj` and `val`.

Parameters

- `task` (*MSKtask_t*) – An optimization task. (input)
- `first` (*MSKint32t*) – Index of the first column in the sequence. (input)
- `last` (*MSKint32t*) – Index of the last column in the sequence **plus one**. (input)
- `maxnumnz` (*MSKint64t*) – Denotes the length of the arrays `subi`, `subj`, and `val`. (input)
- `surp` (*MSKint64t by reference*) – Surplus of subscript and coefficient arrays. The required entries are stored sequentially in `subi`, `subj` and `val` starting from position `surp` away from the end of the arrays. On return `surp` will be decremented by the total number of non-zeros written. (input/output)
- `subi` (*MSKint32t**) – Constraint subscripts. (output)
- `subj` (*MSKint32t**) – Column subscripts. (output)
- `val` (*MSKrealt**) – Values. (output)

Return (*MSKrescodee*) – The function response code.

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

MSK_getaij

```
MSKrescodee (MSKAPI MSK_getaij) (
    MSKtask_t task,
    MSKint32t i,
    MSKint32t j,
    MSKrealt * aij)
```

Obtains a single coefficient in A .

Parameters

- `task` (*MSKtask_t*) – An optimization task. (input)
- `i` (*MSKint32t*) – Row index of the coefficient to be returned. (input)
- `j` (*MSKint32t*) – Column index of the coefficient to be returned. (input)
- `aij` (*MSKrealt by reference*) – The required coefficient $a_{i,j}$. (output)

Return (*MSKrescodee*) – The function response code.

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

MSK_getapiecenumnz

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

Parameters

- `task` (*MSKtask_t*) – An optimization task. (input)
- `i` (*MSKint32t*) – Index of the row. (input)
- `nzi` (*MSKint32t by reference*) – Number of non-zeros in the i -th row of A . (output)

Return (*MSKrescodee*) – The function response code.

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

MSK_getarowslice

```
MSKrescodee (MSKAPI MSK_getarowslice) (
    MSKtask_t task,
    MSKint32t first,
    MSKint32t last,
    MSKint32t maxnumnz,
    MSKint32t * surp,
    MSKint32t * ptrb,
    MSKint32t * ptre,
    MSKint32t * sub,
    MSKrealt * val)
```

Obtains a sequence of rows from A in sparse format.

Parameters

- `task` (*MSKtask_t*) – An optimization task. (input)
- `first` (*MSKint32t*) – Index of the first row in the sequence. (input)
- `last` (*MSKint32t*) – Index of the last row in the sequence **plus one**. (input)
- `maxnumnz` (*MSKint32t*) – Denotes the length of the arrays `sub` and `val`. (input)
- `surp` (*MSKint32t by reference*) – Surplus of subscript and coefficient arrays. The required entries are stored sequentially in `sub` and `val` starting from position `surp` away from the end of the arrays. Upon return `surp` will be decremented by the total number of non-zeros written. (input/output)
- `ptrb` (*MSKint32t**) – `ptrb[t]` is an index pointing to the first element in the t -th row obtained. (output)
- `ptre` (*MSKint32t**) – `ptre[t]` is an index pointing to the last element plus one in the t -th row obtained. (output)
- `sub` (*MSKint32t**) – Contains the column subscripts. (output)
- `val` (*MSKrealt**) – Contains the coefficient values. (output)

Return (*MSKrescodee*) – The function response code.

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

MSK_getarowslice64

```
MSKrescodee (MSKAPI MSK_getarowslice64) (
    MSKtask_t task,
    MSKint32t first,
    MSKint32t last,
    MSKint64t maxnumnz,
    MSKint64t * surp,
    MSKint64t * ptrb,
    MSKint64t * ptre,
    MSKint32t * sub,
    MSKrealt * val)
```

Obtains a sequence of rows from A in sparse format.


```

MSKrescodee (MSKAPI MSK_getarowslicetrip) (
    MSKtask_t task,
    MSKint32t first,
    MSKint32t last,
    MSKint64t maxnumnz,
    MSKint64t * surp,
    MSKint32t * subi,
    MSKint32t * subj,
    MSKrealt * val)

```

Obtains a sequence of rows from A in sparse triplet format. The function returns the content of all rows whose index i satisfies $\text{first} \leq i < \text{last}$. The triplets corresponding to nonzero entries are stored in the arrays `subi`, `subj` and `val`.

Parameters

- `task` (*MSKtask_t*) – An optimization task. (input)
- `first` (*MSKint32t*) – Index of the first row in the sequence. (input)
- `last` (*MSKint32t*) – Index of the last row in the sequence **plus one**. (input)
- `maxnumnz` (*MSKint64t*) – Denotes the length of the arrays `subi`, `subj`, and `val`. (input)
- `surp` (*MSKint64t by reference*) – Surplus of subscript and coefficient arrays. The required entries are stored sequentially in `subi`, `subj` and `val` starting from position `surp` away from the end of the arrays. On return `surp` will be decremented by the total number of non-zeros written. (input/output)
- `subi` (*MSKint32t**) – Constraint subscripts. (output)
- `subj` (*MSKint32t**) – Column subscripts. (output)
- `val` (*MSKrealt**) – Values. (output)

Return (*MSKrescodee*) – The function response code.

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

MSK_getatruncatetol

```

MSKrescodee (MSKAPI MSK_getatruncatetol) (
    MSKtask_t task,
    MSKrealt * tolzero)

```

Obtains the tolerance value set with *MSK_putatruncatetol*.

Parameters

- `task` (*MSKtask_t*) – An optimization task. (input)
- `tolzero` (*MSKrealt**) – All elements $|a_{i,j}|$ less than this tolerance is truncated to zero. (output)

Return (*MSKrescodee*) – The function response code.

Groups *Parameters, Problem data - linear part*

MSK_getbarablocktriplet

```

MSKrescodee (MSKAPI MSK_getbarablocktriplet) (
    MSKtask_t task,
    MSKint64t maxnum,
    MSKint64t * num,
    MSKint32t * subi,
    MSKint32t * subj,
    MSKint32t * subk,
    MSKint32t * subl,
    MSKrealt * valijkl)

```

Obtains \overline{A} in block triplet form.

Parameters

- **task** (*MSKtask_t*) – An optimization task. (input)
- **maxnum** (*MSKint64t*) – **subi**, **subj**, **subk**, **subl** and **valijkl** must be **maxnum** long. (input)
- **num** (*MSKint64t by reference*) – Number of elements in the block triplet form. (output)
- **subi** (*MSKint32t**) – Constraint index. (output)
- **subj** (*MSKint32t**) – Symmetric matrix variable index. (output)
- **subk** (*MSKint32t**) – Block row index. (output)
- **subl** (*MSKint32t**) – Block column index. (output)
- **valijkl** (*MSKrealt**) – The numerical value associated with each block triplet. (output)

Return (*MSKrescodee*) – The function response code.

Groups *Problem data - semidefinite, Inspecting the task*

MSK_getbaraidx

```
MSKrescodee (MSKAPI MSK_getbaraidx) (  
    MSKtask_t task,  
    MSKint64t idx,  
    MSKint64t maxnum,  
    MSKint32t * i,  
    MSKint32t * j,  
    MSKint64t * num,  
    MSKint64t * sub,  
    MSKrealt * weights)
```

Obtains information about an element in \bar{A} . Since \bar{A} is a sparse matrix of symmetric matrices, only the nonzero elements in \bar{A} are stored in order to save space. Now \bar{A} is stored vectorized i.e. as one long vector. This function makes it possible to obtain information such as the row index and the column index of a particular element of the vectorized form of \bar{A} .

Please observe if one element of \bar{A} is inputted multiple times then it may be stored several times in vectorized form. In that case the element with the highest index is the one that is used.

Parameters

- **task** (*MSKtask_t*) – An optimization task. (input)
- **idx** (*MSKint64t*) – Position of the element in the vectorized form. (input)
- **maxnum** (*MSKint64t*) – **sub** and **weights** must be at least **maxnum** long. (input)
- **i** (*MSKint32t by reference*) – Row index of the element at position **idx**. (output)
- **j** (*MSKint32t by reference*) – Column index of the element at position **idx**. (output)
- **num** (*MSKint64t by reference*) – Number of terms in weighted sum that forms the element. (output)
- **sub** (*MSKint64t**) – A list indexes of the elements from symmetric matrix storage that appear in the weighted sum. (output)
- **weights** (*MSKrealt**) – The weights associated with each term in the weighted sum. (output)

Return (*MSKrescodee*) – The function response code.

Groups *Problem data - semidefinite, Inspecting the task*

MSK_getbaraidxij

```
MSKrescodee (MSKAPI MSK_getbaraidxij) (  
    MSKtask_t task,  
    MSKint64t idx,  
    MSKint32t * i,  
    MSKint32t * j)
```


- `idxij` (*MSKint64t**) – Position of each nonzero element in the vectorized form of \bar{A} . (output)

Return (*MSKrescodee*) – The function response code.

Groups *Problem data - semidefinite, Inspecting the task*

MSK_getbarcblocktriplet

```
MSKrescodee (MSKAPI MSK_getbarcblocktriplet) (
    MSKtask_t task,
    MSKint64t maxnum,
    MSKint64t * num,
    MSKint32t * subj,
    MSKint32t * subk,
    MSKint32t * subl,
    MSKrealt * valjkl)
```

Obtains \bar{C} in block triplet form.

Parameters

- `task` (*MSKtask_t*) – An optimization task. (input)
- `maxnum` (*MSKint64t*) – `subj`, `subk`, `subl` and `valjkl` must be `maxnum` long. (input)
- `num` (*MSKint64t by reference*) – Number of elements in the block triplet form. (output)
- `subj` (*MSKint32t**) – Symmetric matrix variable index. (output)
- `subk` (*MSKint32t**) – Block row index. (output)
- `subl` (*MSKint32t**) – Block column index. (output)
- `valjkl` (*MSKrealt**) – The numerical value associated with each block triplet. (output)

Return (*MSKrescodee*) – The function response code.

Groups *Problem data - semidefinite, Inspecting the task*

MSK_getbarcidx

```
MSKrescodee (MSKAPI MSK_getbarcidx) (
    MSKtask_t task,
    MSKint64t idx,
    MSKint64t maxnum,
    MSKint32t * j,
    MSKint64t * num,
    MSKint64t * sub,
    MSKrealt * weights)
```

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

Parameters

- `task` (*MSKtask_t*) – An optimization task. (input)
- `idx` (*MSKint64t*) – Index of the element for which information should be obtained. (input)
- `maxnum` (*MSKint64t*) – `sub` and `weights` must be at least `maxnum` long. (input)
- `j` (*MSKint32t by reference*) – Row index in \bar{C} . (output)
- `num` (*MSKint64t by reference*) – Number of terms in the weighted sum. (output)
- `sub` (*MSKint64t**) – Elements appearing the weighted sum. (output)
- `weights` (*MSKrealt**) – Weights of terms in the weighted sum. (output)

Return (*MSKrescodee*) – The function response code.

Groups *Problem data - semidefinite, Inspecting the task*

MSK_getbarcidxinfo

```
MSKrescodee (MSKAPI MSK_getbarcidxinfo) (  
    MSKtask_t task,  
    MSKint64t idx,  
    MSKint64t * num)
```

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

Parameters

- **task** (*MSKtask_t*) – An optimization task. (input)
- **idx** (*MSKint64t*) – Index of the element for which information should be obtained. The value is an index of a symmetric sparse variable. (input)
- **num** (*MSKint64t by reference*) – Number of terms that appear in the weighted sum that forms the requested element. (output)

Return (*MSKrescodee*) – The function response code.

Groups *Problem data - semidefinite, Inspecting the task*

MSK_getbarcidxj

```
MSKrescodee (MSKAPI MSK_getbarcidxj) (  
    MSKtask_t task,  
    MSKint64t idx,  
    MSKint32t * j)
```

Obtains the row index of an element in \overline{C} .

Parameters

- **task** (*MSKtask_t*) – An optimization task. (input)
- **idx** (*MSKint64t*) – Index of the element for which information should be obtained. (input)
- **j** (*MSKint32t by reference*) – Row index in \overline{C} . (output)

Return (*MSKrescodee*) – The function response code.

Groups *Problem data - semidefinite, Inspecting the task*

MSK_getbarcsparsity

```
MSKrescodee (MSKAPI MSK_getbarcsparsity) (  
    MSKtask_t task,  
    MSKint64t maxnumnz,  
    MSKint64t * numnz,  
    MSKint64t * idxj)
```

Internally only the nonzero elements of \overline{C} are stored in a vector. This function is used to obtain the nonzero elements of \overline{C} and their indexes in the internal vector representation (in **idxj**). From the index detailed information about each nonzero \overline{C}_j can be obtained using *MSK_getbarcidxinfo* and *MSK_getbarcidx*.

Parameters

- **task** (*MSKtask_t*) – An optimization task. (input)
- **maxnumnz** (*MSKint64t*) – **idxj** must be at least **maxnumnz** long. (input)
- **numnz** (*MSKint64t by reference*) – Number of nonzero elements in \overline{C} . (output)
- **idxj** (*MSKint64t**) – Internal positions of the nonzero elements in \overline{C} . (output)

Return (*MSKrescodee*) – The function response code.

Groups *Problem data - semidefinite, Inspecting the task*

MSK_getbarsj

```
MSKrescodee (MSKAPI MSK_getbarsj) (  
    MSKtask_t task,  
    MSKsoltypee whichsol,  
    MSKint32t j,  
    MSKrealt * barsj)
```

Obtains the dual solution for a semidefinite variable. Only the lower triangular part of \bar{S}_j is returned because the matrix by construction is symmetric. The format is that the columns are stored sequentially in the natural order.

Parameters

- **task** (*MSKtask_t*) – An optimization task. (input)
- **whichsol** (*MSKsoltypee*) – Selects a solution. (input)
- **j** (*MSKint32t*) – Index of the semidefinite variable. (input)
- **barsj** (*MSKrealt**) – Value of \bar{S}_j . (output)

Return (*MSKrescodee*) – The function response code.

Groups *Solution - semidefinite*

MSK_getbarsslice

```
MSKrescodee (MSKAPI MSK_getbarsslice) (  
    MSKtask_t task,  
    MSKsoltypee whichsol,  
    MSKint32t first,  
    MSKint32t last,  
    MSKint64t slicesize,  
    MSKrealt * barsslice)
```

Obtains the dual solution for a sequence of semidefinite variables. The format is that matrices are stored sequentially, and in each matrix the columns are stored as in *MSK_getbarsj*.

Parameters

- **task** (*MSKtask_t*) – An optimization task. (input)
- **whichsol** (*MSKsoltypee*) – Selects a solution. (input)
- **first** (*MSKint32t*) – Index of the first semidefinite variable in the slice. (input)
- **last** (*MSKint32t*) – Index of the last semidefinite variable in the slice plus one. (input)
- **slicesize** (*MSKint64t*) – Denotes the length of the array **barsslice**. (input)
- **barsslice** (*MSKrealt**) – Dual solution values of symmetric matrix variables in the slice, stored sequentially. (output)

Return (*MSKrescodee*) – The function response code.

Groups *Solution - semidefinite*

MSK_getbarvarname

```
MSKrescodee (MSKAPI MSK_getbarvarname) (  
    MSKtask_t task,  
    MSKint32t i,  
    MSKint32t sizename,  
    char * name)
```

Obtains the name of a semidefinite variable.

Parameters

- **task** (*MSKtask_t*) – An optimization task. (input)

- `i` (*MSKint32t*) – Index of the variable. (input)
- `size` (*MSKint32t*) – Length of the name buffer. (input)
- `name` (*MSKstring_t*) – The requested name is copied to this buffer. (output)

Return (*MSKrescodee*) – The function response code.

Groups *Names, Inspecting the task*

MSK_getbarvarnameindex

```
MSKrescodee (MSKAPI MSK_getbarvarnameindex) (
    MSKtask_t task,
    const char * somename,
    MSKint32t * asgn,
    MSKint32t * index)
```

Obtains the index of semidefinite variable from its name.

Parameters

- `task` (*MSKtask_t*) – An optimization task. (input)
- `somename` (*MSKstring_t*) – The name of the variable. (input)
- `asgn` (*MSKint32t by reference*) – Non-zero if the name `somename` is assigned to some semidefinite variable. (output)
- `index` (*MSKint32t by reference*) – The index of a semidefinite variable with the name `somename` (if one exists). (output)

Return (*MSKrescodee*) – The function response code.

Groups *Names, Inspecting the task*

MSK_getbarvarnamelen

```
MSKrescodee (MSKAPI MSK_getbarvarnamelen) (
    MSKtask_t task,
    MSKint32t i,
    MSKint32t * len)
```

Obtains the length of the name of a semidefinite variable.

Parameters

- `task` (*MSKtask_t*) – An optimization task. (input)
- `i` (*MSKint32t*) – Index of the variable. (input)
- `len` (*MSKint32t by reference*) – Returns the length of the indicated name. (output)

Return (*MSKrescodee*) – The function response code.

Groups *Names, Inspecting the task*

MSK_getbarxj

```
MSKrescodee (MSKAPI MSK_getbarxj) (
    MSKtask_t task,
    MSKsoltypee whichsol,
    MSKint32t j,
    MSKrealt * barxj)
```

Obtains the primal solution for a semidefinite variable. Only the lower triangular part of \bar{X}_j is returned because the matrix by construction is symmetric. The format is that the columns are stored sequentially in the natural order.

Parameters

- `task` (*MSKtask_t*) – An optimization task. (input)

- `whichsol` (*MSKsoltypee*) – Selects a solution. (input)
- `j` (*MSKint32t*) – Index of the semidefinite variable. (input)
- `barxj` (*MSKrealt**) – Value of \bar{X}_j . (output)

Return (*MSKrescodee*) – The function response code.

Groups *Solution - semidefinite*

MSK_getbarxslice

```
MSKrescodee (MSKAPI MSK_getbarxslice) (
    MSKtask_t task,
    MSKsoltypee whichsol,
    MSKint32t first,
    MSKint32t last,
    MSKint64t slicesize,
    MSKrealt * barxslice)
```

Obtains the primal solution for a sequence of semidefinite variables. The format is that matrices are stored sequentially, and in each matrix the columns are stored as in *MSK_getbarxj*.

Parameters

- `task` (*MSKtask_t*) – An optimization task. (input)
- `whichsol` (*MSKsoltypee*) – Selects a solution. (input)
- `first` (*MSKint32t*) – Index of the first semidefinite variable in the slice. (input)
- `last` (*MSKint32t*) – Index of the last semidefinite variable in the slice plus one. (input)
- `slicesize` (*MSKint64t*) – Denotes the length of the array `barxslice`. (input)
- `barxslice` (*MSKrealt**) – Solution values of symmetric matrix variables in the slice, stored sequentially. (output)

Return (*MSKrescodee*) – The function response code.

Groups *Solution - semidefinite*

MSK_getbuildinfo

```
MSKrescodee (MSKAPI MSK_getbuildinfo) (
    char * buildstate,
    char * builddate)
```

Obtains build information.

Parameters

- `buildstate` (*MSKstring_t*) – State of binaries, i.e. a debug, release candidate or final release. (output)
- `builddate` (*MSKstring_t*) – Date when the binaries were built. (output)

Return (*MSKrescodee*) – The function response code.

Groups *Versions*

MSK_getc

```
MSKrescodee (MSKAPI MSK_getc) (
    MSKtask_t task,
    MSKrealt * c)
```

Obtains all objective coefficients *c*.

Parameters

- `task` (*MSKtask_t*) – An optimization task. (input)


```
MSKrescodee (MSKAPI MSK_getclist) (
    MSKtask_t task,
    MSKint32t num,
    const MSKint32t * subj,
    MSKrealt * c)
```

Obtains a sequence of elements in *c*.

Parameters

- *task* (*MSKtask_t*) – An optimization task. (input)
- *num* (*MSKint32t*) – Number of variables for which the *c* values should be obtained. (input)
- *subj* (*MSKint32t**) – A list of variable indexes. (input)
- *c* (*MSKrealt**) – Linear terms of the requested list of the objective as a dense vector. (output)

Return (*MSKrescodee*) – The function response code.

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

MSK_getcodedesc

```
MSKrescodee (MSKAPI MSK_getcodedesc) (
    MSKrescodee code,
    char * symname,
    char * str)
```

Obtains a short description of the meaning of the response code given by *code*.

Parameters

- *code* (*MSKrescodee*) – A valid **MOSEK** response code. (input)
- *symname* (*MSKstring_t*) – Symbolic name corresponding to *code*. (output)
- *str* (*MSKstring_t*) – Obtains a short description of a response code. (output)

Return (*MSKrescodee*) – The function response code.

Groups *Names, Responses, errors and warnings*

MSK_getconbound

```
MSKrescodee (MSKAPI MSK_getconbound) (
    MSKtask_t task,
    MSKint32t i,
    MSKboundkeye * bk,
    MSKrealt * bl,
    MSKrealt * bu)
```

Obtains bound information for one constraint.

Parameters

- *task* (*MSKtask_t*) – An optimization task. (input)
- *i* (*MSKint32t*) – Index of the constraint for which the bound information should be obtained. (input)
- *bk* (*MSKboundkeye by reference*) – Bound keys. (output)
- *bl* (*MSKrealt by reference*) – Values for lower bounds. (output)
- *bu* (*MSKrealt by reference*) – Values for upper bounds. (output)

Return (*MSKrescodee*) – The function response code.

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

MSK_getconboundslice

```
MSKrescodee (MSKAPI MSK_getconboundslice) (  
    MSKtask_t task,  
    MSKint32t first,  
    MSKint32t last,  
    MSKboundkeye * bk,  
    MSKrealt * bl,  
    MSKrealt * bu)
```

Obtains bounds information for a slice of the constraints.

Parameters

- **task** (*MSKtask_t*) – An optimization task. (input)
- **first** (*MSKint32t*) – First index in the sequence. (input)
- **last** (*MSKint32t*) – Last index plus 1 in the sequence. (input)
- **bk** (*MSKboundkeye**) – Bound keys. (output)
- **bl** (*MSKrealt**) – Values for lower bounds. (output)
- **bu** (*MSKrealt**) – Values for upper bounds. (output)

Return (*MSKrescodee*) – The function response code.

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

MSK_getcone

```
MSKrescodee (MSKAPI MSK_getcone) (  
    MSKtask_t task,  
    MSKint32t k,  
    MSKconetypee * ct,  
    MSKrealt * coneapar,  
    MSKint32t * nummem,  
    MSKint32t * submem)
```

Obtains a cone.

Parameters

- **task** (*MSKtask_t*) – An optimization task. (input)
- **k** (*MSKint32t*) – Index of the cone. (input)
- **ct** (*MSKconetypee by reference*) – Specifies the type of the cone. (output)
- **coneapar** (*MSKrealt 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** (*MSKint32t by reference*) – Number of member variables in the cone. (output)
- **submem** (*MSKint32t**) – Variable subscripts of the members in the cone. (output)

Return (*MSKrescodee*) – The function response code.

Groups *Inspecting the task, Problem data - cones*

MSK_getconeinfo

```
MSKrescodee (MSKAPI MSK_getconeinfo) (  
    MSKtask_t task,  
    MSKint32t k,  
    MSKconetypee * ct,  
    MSKrealt * coneapar,  
    MSKint32t * nummem)
```

Obtains information about a cone.

Parameters

- `task` (*MSKtask_t*) – An optimization task. (input)
- `k` (*MSKint32t*) – Index of the cone. (input)
- `ct` (*MSKconetypee by reference*) – Specifies the type of the cone. (output)
- `conepar` (*MSKrealt 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` (*MSKint32t by reference*) – Number of member variables in the cone. (output)

Return (*MSKrescodee*) – The function response code.

Groups *Inspecting the task, Problem data - cones*

MSK_getconename

```
MSKrescodee (MSKAPI MSK_getconename) (  
    MSKtask_t task,  
    MSKint32t i,  
    MSKint32t sizename,  
    char * name)
```

Obtains the name of a cone.

Parameters

- `task` (*MSKtask_t*) – An optimization task. (input)
- `i` (*MSKint32t*) – Index of the cone. (input)
- `sizename` (*MSKint32t*) – Maximum length of a name that can be stored in `name`. (input)
- `name` (*MSKstring_t*) – The required name. (output)

Return (*MSKrescodee*) – The function response code.

Groups *Names, Problem data - cones, Inspecting the task*

MSK_getconenameindex

```
MSKrescodee (MSKAPI MSK_getconenameindex) (  
    MSKtask_t task,  
    const char * somename,  
    MSKint32t * asgn,  
    MSKint32t * index)
```

Checks whether the name `somename` has been assigned to any cone. If it has been assigned to a cone, then the index of the cone is reported.

Parameters

- `task` (*MSKtask_t*) – An optimization task. (input)
- `somename` (*MSKstring_t*) – The name which should be checked. (input)
- `asgn` (*MSKint32t by reference*) – Is non-zero if the name `somename` is assigned to some cone. (output)
- `index` (*MSKint32t by reference*) – If the name `somename` is assigned to some cone, then `index` is the index of the cone. (output)

Return (*MSKrescodee*) – The function response code.

Groups *Names, Problem data - cones, Inspecting the task*

MSK_getconenamelen

```
MSKrescodee (MSKAPI MSK_getconenamelen) (  
    MSKtask_t task,  
    MSKint32t i,  
    MSKint32t * len)
```

Obtains the length of the name of a cone.

Parameters

- **task** (*MSKtask_t*) – An optimization task. (input)
- **i** (*MSKint32t*) – Index of the cone. (input)
- **len** (*MSKint32t by reference*) – Returns the length of the indicated name. (output)

Return (*MSKrescodee*) – The function response code.

Groups *Names, Problem data - cones, Inspecting the task*

MSK_getconname

```
MSKrescodee (MSKAPI MSK_getconname) (  
    MSKtask_t task,  
    MSKint32t i,  
    MSKint32t sizename,  
    char * name)
```

Obtains the name of a constraint.

Parameters

- **task** (*MSKtask_t*) – An optimization task. (input)
- **i** (*MSKint32t*) – Index of the constraint. (input)
- **sizename** (*MSKint32t*) – Maximum length of name that can be stored in **name**. (input)
- **name** (*MSKstring_t*) – The required name. (output)

Return (*MSKrescodee*) – The function response code.

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

MSK_getconnameindex

```
MSKrescodee (MSKAPI MSK_getconnameindex) (  
    MSKtask_t task,  
    const char * somename,  
    MSKint32t * asgn,  
    MSKint32t * index)
```

Checks whether the name **somename** has been assigned to any constraint. If so, the index of the constraint is reported.

Parameters

- **task** (*MSKtask_t*) – An optimization task. (input)
- **somename** (*MSKstring_t*) – The name which should be checked. (input)
- **asgn** (*MSKint32t by reference*) – Is non-zero if the name **somename** is assigned to some constraint. (output)
- **index** (*MSKint32t by reference*) – If the name **somename** is assigned to a constraint, then **index** is the index of the constraint. (output)

Return (*MSKrescodee*) – The function response code.

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

MSK_getconnamelen

```
MSKrescodee (MSKAPI MSK_getconnamelen) (  
    MSKtask_t task,  
    MSKint32t i,  
    MSKint32t * len)
```

Obtains the length of the name of a constraint.

Parameters

- **task** (*MSKtask_t*) – An optimization task. (input)
- **i** (*MSKint32t*) – Index of the constraint. (input)
- **len** (*MSKint32t by reference*) – Returns the length of the indicated name. (output)

Return (*MSKrescodee*) – The function response code.

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

MSK_getcslice

```
MSKrescodee (MSKAPI MSK_getcslice) (  
    MSKtask_t task,  
    MSKint32t first,  
    MSKint32t last,  
    MSKrealt * c)
```

Obtains a sequence of elements in *c*.

Parameters

- **task** (*MSKtask_t*) – An optimization task. (input)
- **first** (*MSKint32t*) – First index in the sequence. (input)
- **last** (*MSKint32t*) – Last index plus 1 in the sequence. (input)
- **c** (*MSKrealt**) – Linear terms of the requested slice of the objective as a dense vector. The length is **last-first**. (output)

Return (*MSKrescodee*) – The function response code.

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

MSK_getdimbarvarj

```
MSKrescodee (MSKAPI MSK_getdimbarvarj) (  
    MSKtask_t task,  
    MSKint32t j,  
    MSKint32t * dimbarvarj)
```

Obtains the dimension of a symmetric matrix variable.

Parameters

- **task** (*MSKtask_t*) – An optimization task. (input)
- **j** (*MSKint32t*) – Index of the semidefinite variable whose dimension is requested. (input)
- **dimbarvarj** (*MSKint32t by reference*) – The dimension of the *j*-th semidefinite variable. (output)

Return (*MSKrescodee*) – The function response code.

Groups *Inspecting the task, Problem data - semidefinite*

MSK_getdouinf

```
MSKrescodee (MSKAPI MSK_getdouinf) (  
    MSKtask_t task,  
    MSKdinfiteme whichdinf,  
    MSKrealt * dvalue)
```

Obtains a double information item from the task information database.

Parameters

- `task` (*MSKtask_t*) – An optimization task. (input)
- `whichdinf` (*MSKdinfiteme*) – Specifies a double information item. (input)
- `dvalue` (*MSKrealt by reference*) – The value of the required double information item. (output)

Return (*MSKrescodee*) – The function response code.

Groups *Information items and statistics*

MSK_getdouparam

```
MSKrescodee (MSKAPI MSK_getdouparam) (
    MSKtask_t task,
    MSKdparame param,
    MSKrealt * parvalue)
```

Obtains the value of a double parameter.

Parameters

- `task` (*MSKtask_t*) – An optimization task. (input)
- `param` (*MSKdparame*) – Which parameter. (input)
- `parvalue` (*MSKrealt by reference*) – Parameter value. (output)

Return (*MSKrescodee*) – The function response code.

Groups *Parameters*

MSK_getdualobj

```
MSKrescodee (MSKAPI MSK_getdualobj) (
    MSKtask_t task,
    MSKsoltypee whichsol,
    MSKrealt * dualobj)
```

Computes the dual objective value associated with the solution. Note that if the solution is a primal infeasibility certificate, then the fixed term in the objective value is not included.

Moreover, since there is no dual solution associated with an integer solution, an error will be reported if the dual objective value is requested for the integer solution.

Parameters

- `task` (*MSKtask_t*) – An optimization task. (input)
- `whichsol` (*MSKsoltypee*) – Selects a solution. (input)
- `dualobj` (*MSKrealt by reference*) – Objective value corresponding to the dual solution. (output)

Return (*MSKrescodee*) – The function response code.

Groups *Solution information, Solution - dual*

MSK_getdualsolutionnorms

```
MSKrescodee (MSKAPI MSK_getdualsolutionnorms) (
    MSKtask_t task,
    MSKsoltypee whichsol,
    MSKrealt * nrmx,
    MSKrealt * nrmslc,
    MSKrealt * nrmsuc,
    MSKrealt * nrmslx,
    MSKrealt * nrmsux,
    MSKrealt * nrmsnx,
    MSKrealt * nrmbars)
```

Compute norms of the dual solution.

Parameters

- `task` (*MSKtask_t*) – An optimization task. (input)
- `whichsol` (*MSKsoltypee*) – Selects a solution. (input)
- `num` (*MSKint32t*) – Length of `sub` and `viol`. (input)
- `sub` (*MSKint32t**) – An array of indexes of x variables. (input)
- `viol` (*MSKrealt**) – `viol[k]` is the violation of dual solution associated with the variable `sub[k]`. (output)

Return (*MSKrescodee*) – The function response code.

Groups *Solution information*

MSK_getenv

```
MSKrescodee (MSKAPI MSK_getenv) (  
    MSKtask_t task,  
    MSKenv_t * env)
```

Obtains the environment used to create the task.

Parameters

- `task` (*MSKtask_t*) – An optimization task. (input)
- `env` (*MSKenv_t by reference*) – The MOSEK environment. (output)

Return (*MSKrescodee*) – The function response code.

Groups *Environment and task management*

MSK_getinfeasiblesubproblem

```
MSKrescodee (MSKAPI MSK_getinfeasiblesubproblem) (  
    MSKtask_t task,  
    MSKsoltypee whichsol,  
    MSKtask_t * inftask)
```

Given the solution is a certificate of primal or dual infeasibility then a primal or dual infeasible subproblem is obtained respectively. The subproblem tends to be much smaller than the original problem and hence it is easier to locate the infeasibility inspecting the subproblem than the original problem.

For the procedure to be useful it is important to assign meaningful names to constraints, variables etc. in the original task because those names will be duplicated in the subproblem.

The function is only applicable to linear and conic quadratic optimization problems.

For more information see [Sec. 8.3](#) and [Sec. 14.2](#).

Parameters

- `task` (*MSKtask_t*) – An optimization task. (input)
- `whichsol` (*MSKsoltypee*) – Which solution to use when determining the infeasible subproblem. (input)
- `inftask` (*MSKtask_t by reference*) – A new task containing the infeasible subproblem. (output)

Return (*MSKrescodee*) – The function response code.

Groups *Infeasibility diagnostic*

MSK_getinfindex

```
MSKrescodee (MSKAPI MSK_getinfindex) (  
    MSKtask_t task,  
    MSKinftypee inftype,  
    const char * infname,  
    MSKint32t * infindex)
```

Obtains the index of a named information item.

Parameters

- **task** (*MSKtask_t*) – An optimization task. (input)
- **inftype** (*MSKinftypee*) – Type of the information item. (input)
- **iname** (*MSKstring_t*) – Name of the information item. (input)
- **inindex** (*MSKint32t by reference*) – The item index. (output)

Return (*MSKrescodee*) – The function response code.

Groups *Information items and statistics*

MSK_getinfmax

```
MSKrescodee (MSKAPI MSK_getinfmax) (  
    MSKtask_t task,  
    MSKinftypee inftype,  
    MSKint32t * inmax)
```

Obtains the maximum index of an information item of a given type **inftype** plus 1.

Parameters

- **task** (*MSKtask_t*) – An optimization task. (input)
- **inftype** (*MSKinftypee*) – Type of the information item. (input)
- **inmax** (*MSKint32t**) – The maximum index (plus 1) requested. (output)

Return (*MSKrescodee*) – The function response code.

Groups *Information items and statistics*

MSK_getinfname

```
MSKrescodee (MSKAPI MSK_getinfname) (  
    MSKtask_t task,  
    MSKinftypee inftype,  
    MSKint32t whichinf,  
    char * inname)
```

Obtains the name of an information item.

Parameters

- **task** (*MSKtask_t*) – An optimization task. (input)
- **inftype** (*MSKinftypee*) – Type of the information item. (input)
- **whichinf** (*MSKint32t*) – An information item. (input)
- **iname** (*MSKstring_t*) – Name of the information item. (output)

Return (*MSKrescodee*) – The function response code.

Groups *Information items and statistics, Names*

MSK_getintinf

```
MSKrescodee (MSKAPI MSK_getintinf) (  
    MSKtask_t task,  
    MSKiinfiteme whichiinf,  
    MSKint32t * ivalue)
```

Obtains an integer information item from the task information database.

Parameters

- **task** (*MSKtask_t*) – An optimization task. (input)
- **whichiinf** (*MSKiinfiteme*) – Specifies an integer information item. (input)

- `ivalue` (*MSKint32t by reference*) – The value of the required integer information item. (output)

Return (*MSKrescodee*) – The function response code.

Groups *Information items and statistics*

MSK_getintparam

```
MSKrescodee (MSKAPI MSK_getintparam) (
    MSKtask_t task,
    MSKiparame param,
    MSKint32t * parvalue)
```

Obtains the value of an integer parameter.

Parameters

- `task` (*MSKtask_t*) – An optimization task. (input)
- `param` (*MSKiparame*) – Which parameter. (input)
- `parvalue` (*MSKint32t by reference*) – Parameter value. (output)

Return (*MSKrescodee*) – The function response code.

Groups *Parameters*

MSK_getlasterror

```
MSKrescodee (MSKAPI MSK_getlasterror) (
    MSKtask_t task,
    MSKrescodee * lastrescode,
    MSKint32t sizelastmsg,
    MSKint32t * lastmsglen,
    char * lastmsg)
```

Obtains the last response code and corresponding message reported in **MOSEK**.

If there is no previous error, warning or termination code for this task, `lastrescode` returns *MSK_RES_OK* and `lastmsg` returns an empty string, otherwise the last response code different from *MSK_RES_OK* and the corresponding message are returned.

Parameters

- `task` (*MSKtask_t*) – An optimization task. (input)
- `lastrescode` (*MSKrescodee by reference*) – Returns the last error code reported in the task. (output)
- `sizelastmsg` (*MSKint32t*) – The length of the `lastmsg` buffer. (input)
- `lastmsglen` (*MSKint32t by reference*) – Returns the length of the last error message reported in the task. (output)
- `lastmsg` (*MSKstring_t*) – Returns the last error message reported in the task. (output)

Return (*MSKrescodee*) – The function response code.

Groups *Responses, errors and warnings*

MSK_getlasterror64

```
MSKrescodee (MSKAPI MSK_getlasterror64) (
    MSKtask_t task,
    MSKrescodee * lastrescode,
    MSKint64t sizelastmsg,
    MSKint64t * lastmsglen,
    char * lastmsg)
```


Return (*MSKrescodee*) – The function response code.

Groups *Inspecting the task, Problem data - semidefinite*

MSK_getmaxnumcon

```
MSKrescodee (MSKAPI MSK_getmaxnumcon) (  
    MSKtask_t task,  
    MSKint32t * maxnumcon)
```

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

Parameters

- **task** (*MSKtask_t*) – An optimization task. (input)
- **maxnumcon** (*MSKint32t by reference*) – Number of preallocated constraints in the optimization task. (output)

Return (*MSKrescodee*) – The function response code.

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

MSK_getmaxnumcone

```
MSKrescodee (MSKAPI MSK_getmaxnumcone) (  
    MSKtask_t task,  
    MSKint32t * maxnumcone)
```

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

Parameters

- **task** (*MSKtask_t*) – An optimization task. (input)
- **maxnumcone** (*MSKint32t by reference*) – Number of preallocated conic constraints in the optimization task. (output)

Return (*MSKrescodee*) – The function response code.

Groups *Inspecting the task, Problem data - cones*

MSK_getmaxnumqnz

```
MSKrescodee (MSKAPI MSK_getmaxnumqnz) (  
    MSKtask_t task,  
    MSKint32t * maxnumqnz)
```

Obtains the number of preallocated non-zeros for Q (both objective and constraints). When this number of non-zeros is reached **MOSEK** will automatically allocate more space for Q .

Parameters

- **task** (*MSKtask_t*) – An optimization task. (input)
- **maxnumqnz** (*MSKint32t by reference*) – Number of non-zero elements preallocated in quadratic coefficient matrices. (output)

Return (*MSKrescodee*) – The function response code.

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

MSK_getmaxnumqnz64

```
MSKrescodee (MSKAPI MSK_getmaxnumqnz64) (  
    MSKtask_t task,  
    MSKint64t * maxnumqnz)
```


Return (*MSKrescodee*) – The function response code.

Groups *Information items and statistics*

MSK_getnadouparam

```
MSKrescodee (MSKAPI MSK_getnadouparam) (  
    MSKtask_t task,  
    const char * paramname,  
    MSKrealt * parvalue)
```

Obtains the value of a named double parameter.

Parameters

- task (*MSKtask_t*) – An optimization task. (input)
- paramname (*MSKstring_t*) – Name of a parameter. (input)
- parvalue (*MSKrealt by reference*) – Parameter value. (output)

Return (*MSKrescodee*) – The function response code.

Groups *Parameters*

MSK_getnaintinf

```
MSKrescodee (MSKAPI MSK_getnaintinf) (  
    MSKtask_t task,  
    const char * infitemname,  
    MSKint32t * ivalue)
```

Obtains a named integer information item from the task information database.

Parameters

- task (*MSKtask_t*) – An optimization task. (input)
- infitemname (*MSKstring_t*) – The name of an integer information item. (input)
- ivalue (*MSKint32t by reference*) – The value of the required integer information item. (output)

Return (*MSKrescodee*) – The function response code.

Groups *Information items and statistics*

MSK_getnaintparam

```
MSKrescodee (MSKAPI MSK_getnaintparam) (  
    MSKtask_t task,  
    const char * paramname,  
    MSKint32t * parvalue)
```

Obtains the value of a named integer parameter.

Parameters

- task (*MSKtask_t*) – An optimization task. (input)
- paramname (*MSKstring_t*) – Name of a parameter. (input)
- parvalue (*MSKint32t by reference*) – Parameter value. (output)

Return (*MSKrescodee*) – The function response code.

Groups *Parameters*

MSK_getnastrparam

```
MSKrescodee (MSKAPI MSK_getnastrparam) (
    MSKtask_t task,
    const char * paramname,
    MSKint32t sizeparamname,
    MSKint32t * len,
    char * parvalue)
```

Obtains the value of a named string parameter.

Parameters

- task (*MSKtask_t*) – An optimization task. (input)
- paramname (*MSKstring_t*) – Name of a parameter. (input)
- sizeparamname (*MSKint32t*) – Size of the name buffer parvalue. (input)
- len (*MSKint32t by reference*) – Length of the string in parvalue. (output)
- parvalue (*MSKstring_t*) – Parameter value. (output)

Return (*MSKrescodee*) – The function response code.

Groups *Parameters, Names*

MSK_getnastrparamal

```
MSKrescodee (MSKAPI MSK_getnastrparamal) (
    MSKtask_t task,
    const char * paramname,
    MSKint32t numaddchr,
    MSKstring_t * value)
```

Obtains the value of a named string parameter.

Parameters

- task (*MSKtask_t*) – An optimization task. (input)
- paramname (*MSKstring_t*) – Name of a parameter. (input)
- numaddchr (*MSKint32t*) – Number of additional characters for which room is left in value. (input)
- value (*MSKstring_t**) – Parameter value. **MOSEK** will allocate this char buffer of size equal to the actual length of the string parameter plus numaddchr. This memory must be freed by *MSK_freetask*. (input/output)

Return (*MSKrescodee*) – The function response code.

Groups *Parameters*

MSK_getnumanz

```
MSKrescodee (MSKAPI MSK_getnumanz) (
    MSKtask_t task,
    MSKint32t * numanz)
```

Obtains the number of non-zeros in A .

Parameters

- task (*MSKtask_t*) – An optimization task. (input)
- numanz (*MSKint32t by reference*) – Number of non-zero elements in the linear constraint matrix. (output)

Return (*MSKrescodee*) – The function response code.

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

MSK_getnumanz64

MSK_getnumconmem

```
MSKrescodee (MSKAPI MSK_getnumconmem) (  
    MSKtask_t task,  
    MSKint32t k,  
    MSKint32t * nummem)
```

Obtains the number of members in a cone.

Parameters

- **task** (*MSKtask_t*) – An optimization task. (input)
- **k** (*MSKint32t*) – Index of the cone. (input)
- **nummem** (*MSKint32t by reference*) – Number of member variables in the cone. (output)

Return (*MSKrescodee*) – The function response code.

Groups *Problem data - cones, Inspecting the task*

MSK_getnumintvar

```
MSKrescodee (MSKAPI MSK_getnumintvar) (  
    MSKtask_t task,  
    MSKint32t * numintvar)
```

Obtains the number of integer-constrained variables.

Parameters

- **task** (*MSKtask_t*) – An optimization task. (input)
- **numintvar** (*MSKint32t by reference*) – Number of integer variables. (output)

Return (*MSKrescodee*) – The function response code.

Groups *Inspecting the task, Problem data - variables*

MSK_getnumparam

```
MSKrescodee (MSKAPI MSK_getnumparam) (  
    MSKtask_t task,  
    MSKparametertypee partype,  
    MSKint32t * numparam)
```

Obtains the number of parameters of a given type.

Parameters

- **task** (*MSKtask_t*) – An optimization task. (input)
- **partype** (*MSKparametertypee*) – Parameter type. (input)
- **numparam** (*MSKint32t by reference*) – The number of parameters of type partype. (output)

Return (*MSKrescodee*) – The function response code.

Groups *Inspecting the task, Parameters*

MSK_getnumqconknz

```
MSKrescodee (MSKAPI MSK_getnumqconknz) (  
    MSKtask_t task,  
    MSKint32t k,  
    MSKint32t * numqcnz)
```

Obtains the number of non-zero quadratic terms in a constraint.

Parameters

- **task** (*MSKtask_t*) – An optimization task. (input)
- **k** (*MSKint32t*) – Index of the constraint for which the number of non-zero quadratic terms should be obtained. (input)
- **numqcnz** (*MSKint32t by reference*) – Number of quadratic terms. (output)

Return (*MSKrescodee*) – The function response code.

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

MSK_getnumqconknz64

```
MSKrescodee (MSKAPI MSK_getnumqconknz64) (  
    MSKtask_t task,  
    MSKint32t k,  
    MSKint64t * numqcnz)
```

Obtains the number of non-zero quadratic terms in a constraint.

Parameters

- **task** (*MSKtask_t*) – An optimization task. (input)
- **k** (*MSKint32t*) – Index of the constraint for which the number quadratic terms should be obtained. (input)
- **numqcnz** (*MSKint64t by reference*) – Number of quadratic terms. (output)

Return (*MSKrescodee*) – The function response code.

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

MSK_getnumqobjnz

```
MSKrescodee (MSKAPI MSK_getnumqobjnz) (  
    MSKtask_t task,  
    MSKint32t * numqonz)
```

Obtains the number of non-zero quadratic terms in the objective.

Parameters

- **task** (*MSKtask_t*) – An optimization task. (input)
- **numqonz** (*MSKint32t by reference*) – Number of non-zero elements in the quadratic objective terms. (output)

Return (*MSKrescodee*) – The function response code.

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

MSK_getnumqobjnz64

```
MSKrescodee (MSKAPI MSK_getnumqobjnz64) (  
    MSKtask_t task,  
    MSKint64t * numqonz)
```

Obtains the number of non-zero quadratic terms in the objective.

Parameters

- **task** (*MSKtask_t*) – An optimization task. (input)
- **numqonz** (*MSKint64t by reference*) – Number of non-zero elements in the quadratic objective terms. (output)

Return (*MSKrescodee*) – The function response code.

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

MSK_getnumsymmat

- **prosta** (*MSKprosta* by reference) – Problem status. (output)
- Return** (*MSKrescodee*) – The function response code.
- Groups** *Solution information*

MSK_getpviolbarvar

```
MSKrescodee (MSKAPI MSK_getpviolbarvar) (
    MSKtask_t task,
    MSKsoltypee whichsol,
    MSKint32t num,
    const MSKint32t * sub,
    MSKrealt * viol)
```

Computes the primal solution violation for a set of semidefinite variables. Let $(\bar{X}_j)^*$ be the value of the variable \bar{X}_j for the specified solution. Then the primal violation of the solution associated with variable \bar{X}_j is given by

$$\max(-\lambda_{\min}(\bar{X}_j), 0.0).$$

Both when the solution is a certificate of dual infeasibility or when it is primal feasible the violation should be small.

Parameters

- **task** (*MSKtask_t*) – An optimization task. (input)
- **whichsol** (*MSKsoltypee*) – Selects a solution. (input)
- **num** (*MSKint32t*) – Length of **sub** and **viol**. (input)
- **sub** (*MSKint32t**) – An array of indexes of \bar{X} variables. (input)
- **viol** (*MSKrealt**) – **viol[k]** is how much the solution violates the constraint $\bar{X}_{\text{sub}[k]} \in \mathcal{S}_+$. (output)

Return (*MSKrescodee*) – The function response code.

Groups *Solution information*

MSK_getpviolcon

```
MSKrescodee (MSKAPI MSK_getpviolcon) (
    MSKtask_t task,
    MSKsoltypee whichsol,
    MSKint32t num,
    const MSKint32t * sub,
    MSKrealt * viol)
```

Computes the primal solution violation for a set of constraints. The primal violation of the solution associated with the i -th constraint is given by

$$\max(\tau l_i^c - (x_i^c)^*, (x_i^c)^* - \tau u_i^c), \mid \sum_{j=0}^{\text{numvar}-1} a_{ij} x_j^* - x_i^c)$$

where $\tau = 0$ if the solution is a certificate of dual infeasibility and $\tau = 1$ otherwise. Both when the solution is a certificate of dual infeasibility and when it is primal feasible the violation should be small. The above formula applies for the linear case but is appropriately generalized in other cases.

Parameters

- **task** (*MSKtask_t*) – An optimization task. (input)
- **whichsol** (*MSKsoltypee*) – Selects a solution. (input)
- **num** (*MSKint32t*) – Length of **sub** and **viol**. (input)
- **sub** (*MSKint32t**) – An array of indexes of constraints. (input)

Return (*MSKrescodee*) – The function response code.

Groups *Solution - dual*

MSK_getresponseclass

```
MSKrescodee (MSKAPI MSK_getresponseclass) (  
    MSKrescodee r,  
    MSKrescodetypee * rc)
```

Obtain the class of a response code.

Parameters

- **r** (*MSKrescodee*) – A response code indicating the result of function call. (input)
- **rc** (*MSKrescodetypee by reference*) – The response class. (output)

Return (*MSKrescodee*) – The function response code.

Groups *Responses, errors and warnings*

MSK_getskc

```
MSKrescodee (MSKAPI MSK_getskc) (  
    MSKtask_t task,  
    MSKsoltypee whichsol,  
    MSKstakeye * skc)
```

Obtains the status keys for the constraints.

Parameters

- **task** (*MSKtask_t*) – An optimization task. (input)
- **whichsol** (*MSKsoltypee*) – Selects a solution. (input)
- **skc** (*MSKstakeye**) – Status keys for the constraints. (output)

Return (*MSKrescodee*) – The function response code.

Groups *Solution information*

MSK_getskcslice

```
MSKrescodee (MSKAPI MSK_getskcslice) (  
    MSKtask_t task,  
    MSKsoltypee whichsol,  
    MSKint32t first,  
    MSKint32t last,  
    MSKstakeye * skc)
```

Obtains the status keys for a slice of the constraints.

Parameters

- **task** (*MSKtask_t*) – An optimization task. (input)
- **whichsol** (*MSKsoltypee*) – Selects a solution. (input)
- **first** (*MSKint32t*) – First index in the sequence. (input)
- **last** (*MSKint32t*) – Last index plus 1 in the sequence. (input)
- **skc** (*MSKstakeye**) – Status keys for the constraints. (output)

Return (*MSKrescodee*) – The function response code.

Groups *Solution information*

MSK_getskn

```
MSKrescodee (MSKAPI MSK_getskn) (
    MSKtask_t task,
    MSKsoltypee whichsol,
    MSKstakeye * skn)
```

Obtains the status keys for the conic constraints.

Parameters

- **task** (*MSKtask_t*) – An optimization task. (input)
- **whichsol** (*MSKsoltypee*) – Selects a solution. (input)
- **skn** (*MSKstakeye**) – Status keys for the conic constraints. (output)

Return (*MSKrescodee*) – The function response code.

Groups *Solution information*

MSK_getskx

```
MSKrescodee (MSKAPI MSK_getskx) (
    MSKtask_t task,
    MSKsoltypee whichsol,
    MSKstakeye * skx)
```

Obtains the status keys for the scalar variables.

Parameters

- **task** (*MSKtask_t*) – An optimization task. (input)
- **whichsol** (*MSKsoltypee*) – Selects a solution. (input)
- **skx** (*MSKstakeye**) – Status keys for the variables. (output)

Return (*MSKrescodee*) – The function response code.

Groups *Solution information*

MSK_getskxslice

```
MSKrescodee (MSKAPI MSK_getskxslice) (
    MSKtask_t task,
    MSKsoltypee whichsol,
    MSKint32t first,
    MSKint32t last,
    MSKstakeye * skx)
```

Obtains the status keys for a slice of the scalar variables.

Parameters

- **task** (*MSKtask_t*) – An optimization task. (input)
- **whichsol** (*MSKsoltypee*) – Selects a solution. (input)
- **first** (*MSKint32t*) – First index in the sequence. (input)
- **last** (*MSKint32t*) – Last index plus 1 in the sequence. (input)
- **skx** (*MSKstakeye**) – Status keys for the variables. (output)

Return (*MSKrescodee*) – The function response code.

Groups *Solution information*

MSK_getslc

```
MSKrescodee (MSKAPI MSK_getslc) (
    MSKtask_t task,
    MSKsoltypee whichsol,
    MSKrealt * slc)
```

Obtains the s_l^c vector for a solution.

Parameters

- **task** (*MSKtask_t*) – An optimization task. (input)
- **whichsol** (*MSKsoltypee*) – Selects a solution. (input)
- **slc** (*MSKrealt**) – Dual variables corresponding to the lower bounds on the constraints. (output)

Return (*MSKrescodee*) – The function response code.

Groups *Solution - dual*

MSK_getslcslice

```
MSKrescodee (MSKAPI MSK_getslcslice) (  
    MSKtask_t task,  
    MSKsoltypee whichsol,  
    MSKint32t first,  
    MSKint32t last,  
    MSKrealt * slc)
```

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

Parameters

- **task** (*MSKtask_t*) – An optimization task. (input)
- **whichsol** (*MSKsoltypee*) – Selects a solution. (input)
- **first** (*MSKint32t*) – First index in the sequence. (input)
- **last** (*MSKint32t*) – Last index plus 1 in the sequence. (input)
- **slc** (*MSKrealt**) – Dual variables corresponding to the lower bounds on the constraints. (output)

Return (*MSKrescodee*) – The function response code.

Groups *Solution - dual*

MSK_getslx

```
MSKrescodee (MSKAPI MSK_getslx) (  
    MSKtask_t task,  
    MSKsoltypee whichsol,  
    MSKrealt * slx)
```

Obtains the s_l^x vector for a solution.

Parameters

- **task** (*MSKtask_t*) – An optimization task. (input)
- **whichsol** (*MSKsoltypee*) – Selects a solution. (input)
- **slx** (*MSKrealt**) – Dual variables corresponding to the lower bounds on the variables. (output)

Return (*MSKrescodee*) – The function response code.

Groups *Solution - dual*

MSK_getslxslice

```
MSKrescodee (MSKAPI MSK_getslxslice) (  
    MSKtask_t task,  
    MSKsoltypee whichsol,  
    MSKint32t first,  
    MSKint32t last,  
    MSKrealt * slx)
```

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

Parameters

- **task** (*MSKtask_t*) – An optimization task. (input)
- **whichsol** (*MSKsoltypee*) – Selects a solution. (input)
- **first** (*MSKint32t*) – First index in the sequence. (input)
- **last** (*MSKint32t*) – Last index plus 1 in the sequence. (input)
- **slx** (*MSKrealt**) – Dual variables corresponding to the lower bounds on the variables. (output)

Return (*MSKrescodee*) – The function response code.

Groups *Solution - dual*

MSK_getsnx

```
MSKrescodee (MSKAPI MSK_getsnx) (  
    MSKtask_t task,  
    MSKsoltypee whichsol,  
    MSKrealt * snx)
```

Obtains the s_n^x vector for a solution.

Parameters

- **task** (*MSKtask_t*) – An optimization task. (input)
- **whichsol** (*MSKsoltypee*) – Selects a solution. (input)
- **snx** (*MSKrealt**) – Dual variables corresponding to the conic constraints on the variables. (output)

Return (*MSKrescodee*) – The function response code.

Groups *Solution - dual*

MSK_getsnxslice

```
MSKrescodee (MSKAPI MSK_getsnxslice) (  
    MSKtask_t task,  
    MSKsoltypee whichsol,  
    MSKint32t first,  
    MSKint32t last,  
    MSKrealt * snx)
```

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

Parameters

- **task** (*MSKtask_t*) – An optimization task. (input)
- **whichsol** (*MSKsoltypee*) – Selects a solution. (input)
- **first** (*MSKint32t*) – First index in the sequence. (input)
- **last** (*MSKint32t*) – Last index plus 1 in the sequence. (input)
- **snx** (*MSKrealt**) – Dual variables corresponding to the conic constraints on the variables. (output)

Return (*MSKrescodee*) – The function response code.

Groups *Solution - dual*

MSK_getsolsta

```
MSKrescodee (MSKAPI MSK_getsolsta) (  
    MSKtask_t task,  
    MSKsoltypee whichsol,  
    MSKsolstae * solsta)
```

Obtains the solution status.

Parameters

- **task** (*MSKtask_t*) – An optimization task. (input)
- **whichsol** (*MSKsoltypee*) – Selects a solution. (input)
- **solsta** (*MSKsolstae by reference*) – Solution status. (output)

Return (*MSKrescodee*) – The function response code.

Groups *Solution information*

MSK_getsolution

```
MSKrescodee (MSKAPI MSK_getsolution) (
    MSKtask_t task,
    MSKsoltypee whichsol,
    MSKprosta * prosta,
    MSKsolstae * solsta,
    MSKstakeye * skc,
    MSKstakeye * skx,
    MSKstakeye * skn,
    MSKrealt * xc,
    MSKrealt * xx,
    MSKrealt * y,
    MSKrealt * slc,
    MSKrealt * suc,
    MSKrealt * slx,
    MSKrealt * sux,
    MSKrealt * snx)
```

Obtains the complete solution.

Consider the case of linear programming. The primal problem is given by

$$\begin{aligned} & \text{minimize} && c^T x + c^f \\ & \text{subject to} && \begin{array}{ccc} l^c & \leq & Ax & \leq & u^c, \\ l^x & \leq & x & \leq & u^x. \end{array} \end{aligned}$$

and the corresponding dual problem is

$$\begin{aligned} & \text{maximize} && (l^c)^T s_l^c - (u^c)^T s_u^c \\ & && + (l^x)^T s_l^x - (u^x)^T s_u^x + c^f \\ & \text{subject to} && \begin{array}{ccc} A^T y + s_l^x - s_u^x & = & c, \\ -y + s_l^c - s_u^c & = & 0, \\ s_l^c, s_u^c, s_l^x, s_u^x & \geq & 0. \end{array} \end{aligned}$$

A conic optimization problem has the same primal variables as in the linear case. Recall that the dual of a conic optimization problem is given by:

$$\begin{aligned} & \text{maximize} && (l^c)^T s_l^c - (u^c)^T s_u^c \\ & && + (l^x)^T s_l^x - (u^x)^T s_u^x + c^f \\ & \text{subject to} && \begin{array}{ccc} A^T y + s_l^x - s_u^x + s_n^x & = & c, \\ -y + s_l^c - s_u^c & = & 0, \\ s_l^c, s_u^c, s_l^x, s_u^x & \geq & 0, \\ s_n^x & \in & \mathcal{K}^* \end{array} \end{aligned}$$

The mapping between variables and arguments to the function is as follows:

- **xx** : Corresponds to variable x (also denoted x^x).
- **xc** : Corresponds to $x^c := Ax$.
- **y** : Corresponds to variable y .
- **slc**: Corresponds to variable s_l^c .

- **suc**: Corresponds to variable s_u^c .
- **slx**: Corresponds to variable s_l^x .
- **sux**: Corresponds to variable s_u^x .
- **snx**: Corresponds to variable s_n^x .

The meaning of the values returned by this function depend on the *solution status* returned in the argument **solsta**. The most important possible values of **solsta** are:

- **MSK_SOL_STA_OPTIMAL** : An optimal solution satisfying the optimality criteria for continuous problems is returned.
- **MSK_SOL_STA_INTEGER_OPTIMAL** : An optimal solution satisfying the optimality criteria for integer problems is returned.
- **MSK_SOL_STA_PRIM_FEAS** : A solution satisfying the feasibility criteria.
- **MSK_SOL_STA_PRIM_INFEAS_CER** : A primal certificate of infeasibility is returned.
- **MSK_SOL_STA_DUAL_INFEAS_CER** : A dual certificate of infeasibility is returned.

In order to retrieve the primal and dual values of semidefinite variables see **MSK_getbarxj** and **MSK_getbarsj**.

Parameters

- **task** (**MSKtask_t**) – An optimization task. (input)
- **whichsol** (**MSKsoltypee**) – Selects a solution. (input)
- **prosta** (**MSKprosta_e by reference**) – Problem status. (output)
- **solsta** (**MSKsolsta_e by reference**) – Solution status. (output)
- **skc** (**MSKstakeye***) – Status keys for the constraints. (output)
- **skx** (**MSKstakeye***) – Status keys for the variables. (output)
- **skn** (**MSKstakeye***) – Status keys for the conic constraints. (output)
- **xc** (**MSKrealt***) – Primal constraint solution. (output)
- **xx** (**MSKrealt***) – Primal variable solution. (output)
- **y** (**MSKrealt***) – Vector of dual variables corresponding to the constraints. (output)
- **slc** (**MSKrealt***) – Dual variables corresponding to the lower bounds on the constraints. (output)
- **suc** (**MSKrealt***) – Dual variables corresponding to the upper bounds on the constraints. (output)
- **slx** (**MSKrealt***) – Dual variables corresponding to the lower bounds on the variables. (output)
- **sux** (**MSKrealt***) – Dual variables corresponding to the upper bounds on the variables. (output)
- **snx** (**MSKrealt***) – Dual variables corresponding to the conic constraints on the variables. (output)

Return (**MSKrescodee**) – The function response code.

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

MSK_getsolutioninfo

```
MSKrescodee (MSKAPI MSK_getsolutioninfo) (
    MSKtask_t task,
    MSKsoltypee whichsol,
    MSKrealt * pobj,
    MSKrealt * pviolcon,
    MSKrealt * pviolvar,
    MSKrealt * pviolbarvar,
    MSKrealt * pviolcone,
```

(continues on next page)

```

MSKrealt * pviolitg,
MSKrealt * dobj,
MSKrealt * dviolcon,
MSKrealt * dviolvar,
MSKrealt * dviolbarvar,
MSKrealt * dviolcone)

```

Obtains information about a solution.

Parameters

- **task** (*MSKtask_t*) – An optimization task. (input)
- **whichsol** (*MSKsoltypee*) – Selects a solution. (input)
- **pobj** (*MSKrealt by reference*) – The primal objective value as computed by *MSK_getprimalobj*. (output)
- **pviolcon** (*MSKrealt by reference*) – Maximal primal violation of the solution associated with the x^c variables where the violations are computed by *MSK_getpviolcon*. (output)
- **pviolvar** (*MSKrealt by reference*) – Maximal primal violation of the solution for the x variables where the violations are computed by *MSK_getpviolvar*. (output)
- **pviolbarvar** (*MSKrealt by reference*) – Maximal primal violation of solution for the \bar{X} variables where the violations are computed by *MSK_getpviolbarvar*. (output)
- **pviolcone** (*MSKrealt by reference*) – Maximal primal violation of solution for the conic constraints where the violations are computed by *MSK_getpviolcones*. (output)
- **pviolitg** (*MSKrealt 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** (*MSKrealt by reference*) – Dual objective value as computed by *MSK_getdualobj*. (output)
- **dviolcon** (*MSKrealt by reference*) – Maximal violation of the dual solution associated with the x^c variable as computed by *MSK_getdviolcon*. (output)
- **dviolvar** (*MSKrealt by reference*) – Maximal violation of the dual solution associated with the x variable as computed by *MSK_getdviolvar*. (output)
- **dviolbarvar** (*MSKrealt by reference*) – Maximal violation of the dual solution associated with the \bar{S} variable as computed by *MSK_getdviolbarvar*. (output)
- **dviolcone** (*MSKrealt by reference*) – Maximal violation of the dual solution associated with the dual conic constraints as computed by *MSK_getdviolcones*. (output)

Return (*MSKrescodee*) – The function response code.

Groups *Solution information*

MSK_getsolutionslice

```

MSKrescodee (MSKAPI MSK_getsolutionslice) (
    MSKtask_t task,
    MSKsoltypee whichsol,
    MSKsoliteme solitem,
    MSKint32t first,
    MSKint32t last,
    MSKrealt * values)

```

Obtains a slice of one item from the solution. The format of the solution is exactly as in *MSK_getsolution*. The parameter *solitem* determines which of the solution vectors should be returned.

Parameters

- `task` (*MSKtask_t*) – An optimization task. (input)
- `whichsol` (*MSKsoltypee*) – Selects a solution. (input)
- `solitem` (*MSKsoliteme*) – Which part of the solution is required. (input)
- `first` (*MSKint32t*) – First index in the sequence. (input)
- `last` (*MSKint32t*) – Last index plus 1 in the sequence. (input)
- `values` (*MSKrealt**) – The values in the required sequence are stored sequentially in `values`. (output)

Return (*MSKrescodee*) – The function response code.

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

MSK_getsparsesymmat

```
MSKrescodee (MSKAPI MSK_getsparsesymmat) (  
    MSKtask_t task,  
    MSKint64t idx,  
    MSKint64t maxlen,  
    MSKint32t * subi,  
    MSKint32t * subj,  
    MSKrealt * valij)
```

Get a single symmetric matrix from the matrix store.

Parameters

- `task` (*MSKtask_t*) – An optimization task. (input)
- `idx` (*MSKint64t*) – Index of the matrix to retrieve. (input)
- `maxlen` (*MSKint64t*) – Length of the output arrays `subi`, `subj` and `valij`. (input)
- `subi` (*MSKint32t**) – Row subscripts of the matrix non-zero elements. (output)
- `subj` (*MSKint32t**) – Column subscripts of the matrix non-zero elements. (output)
- `valij` (*MSKrealt**) – Coefficients of the matrix non-zero elements. (output)

Return (*MSKrescodee*) – The function response code.

Groups *Problem data - semidefinite, Inspecting the task*

MSK_getstrparam

```
MSKrescodee (MSKAPI MSK_getstrparam) (  
    MSKtask_t task,  
    MSKsparame param,  
    MSKint32t maxlen,  
    MSKint32t * len,  
    char * parvalue)
```

Obtains the value of a string parameter.

Parameters

- `task` (*MSKtask_t*) – An optimization task. (input)
- `param` (*MSKsparame*) – Which parameter. (input)
- `maxlen` (*MSKint32t*) – Length of the `parvalue` buffer. (input)
- `len` (*MSKint32t by reference*) – The length of the parameter value. (output)
- `parvalue` (*MSKstring_t*) – Parameter value. (output)

Return (*MSKrescodee*) – The function response code.

Groups *Names, Parameters*

MSK_getstrparamal

```
MSKrescodee (MSKAPI MSK_getstrparamal) (
    MSKtask_t task,
    MSKsparame param,
    MSKint32t numaddchr,
    MSKstring_t * value)
```

Obtains the value of a string parameter.

Parameters

- task (*MSKtask_t*) – An optimization task. (input)
- param (*MSKsparame*) – Which parameter. (input)
- numaddchr (*MSKint32t*) – Number of additional characters for which room is left in value. (input)
- value (*MSKstring_t by reference*) – Parameter value. **MOSEK** will allocate this char buffer of size equal to the actual length of the string parameter plus numaddchr. This memory must be freed by *MSK_freetask*. (input/output)

Return (*MSKrescodee*) – The function response code.

Groups *Parameters*

MSK_getstrparamlen

```
MSKrescodee (MSKAPI MSK_getstrparamlen) (
    MSKtask_t task,
    MSKsparame param,
    MSKint32t * len)
```

Obtains the length of a string parameter.

Parameters

- task (*MSKtask_t*) – An optimization task. (input)
- param (*MSKsparame*) – Which parameter. (input)
- len (*MSKint32t by reference*) – The length of the parameter value. (output)

Return (*MSKrescodee*) – The function response code.

Groups *Names, Parameters*

MSK_getsuc

```
MSKrescodee (MSKAPI MSK_getsuc) (
    MSKtask_t task,
    MSKsoltypee whichsol,
    MSKrealt * suc)
```

Obtains the s_u^c vector for a solution.

Parameters

- task (*MSKtask_t*) – An optimization task. (input)
- whichsol (*MSKsoltypee*) – Selects a solution. (input)
- suc (*MSKrealt**) – Dual variables corresponding to the upper bounds on the constraints. (output)

Return (*MSKrescodee*) – The function response code.

Groups *Solution - dual*

MSK_getsucslice

```
MSKrescodee (MSKAPI MSK_getsucslice) (
    MSKtask_t task,
    MSKsoltypee whichsol,
    MSKint32t first,
    MSKint32t last,
    MSKrealt * suc)
```

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

Parameters

- task (*MSKtask_t*) – An optimization task. (input)
- whichsol (*MSKsoltypee*) – Selects a solution. (input)
- first (*MSKint32t*) – First index in the sequence. (input)
- last (*MSKint32t*) – Last index plus 1 in the sequence. (input)
- suc (*MSKrealt**) – Dual variables corresponding to the upper bounds on the constraints. (output)

Return (*MSKrescodee*) – The function response code.

Groups *Solution - dual*

MSK_getsux

```
MSKrescodee (MSKAPI MSK_getsux) (
    MSKtask_t task,
    MSKsoltypee whichsol,
    MSKrealt * sux)
```

Obtains the s_u^x vector for a solution.

Parameters

- task (*MSKtask_t*) – An optimization task. (input)
- whichsol (*MSKsoltypee*) – Selects a solution. (input)
- sux (*MSKrealt**) – Dual variables corresponding to the upper bounds on the variables. (output)

Return (*MSKrescodee*) – The function response code.

Groups *Solution - dual*

MSK_getsuxslice

```
MSKrescodee (MSKAPI MSK_getsuxslice) (
    MSKtask_t task,
    MSKsoltypee whichsol,
    MSKint32t first,
    MSKint32t last,
    MSKrealt * sux)
```

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

Parameters

- task (*MSKtask_t*) – An optimization task. (input)
- whichsol (*MSKsoltypee*) – Selects a solution. (input)
- first (*MSKint32t*) – First index in the sequence. (input)
- last (*MSKint32t*) – Last index plus 1 in the sequence. (input)
- sux (*MSKrealt**) – Dual variables corresponding to the upper bounds on the variables. (output)

Return (*MSKrescodee*) – The function response code.

Groups *Solution - dual*

MSK_getsymbcon

```
MSKrescodee (MSKAPI MSK_getsymbcon) (  
    MSKtask_t task,  
    MSKint32t i,  
    MSKint32t sizevalue,  
    char * name,  
    MSKint32t * value)
```

Obtains the name and corresponding value for the i th symbolic constant.

Parameters

- **task** (*MSKtask_t*) – An optimization task. (input)
- **i** (*MSKint32t*) – Index. (input)
- **sizevalue** (*MSKint32t*) – The length of the buffer pointed to by the **value** argument. (input)
- **name** (*MSKstring_t*) – Name of the i th symbolic constant. (output)
- **value** (*MSKint32t by reference*) – The corresponding value. (output)

Return (*MSKrescodee*) – The function response code.

Groups *Names*

MSK_getsymbcondim

```
MSKrescodee (MSKAPI MSK_getsymbcondim) (  
    MSKenv_t env,  
    MSKint32t * num,  
    size_t * maxlen)
```

Obtains the number of symbolic constants defined by **MOSEK** and the maximum length of the name of any symbolic constant.

Parameters

- **env** (*MSKenv_t*) – The MOSEK environment. (input)
- **num** (*MSKint32t by reference*) – Number of symbolic constants defined by **MOSEK**. (output)
- **maxlen** (*size_t by reference*) – Maximum length of the name of any symbolic constants. (output)

Return (*MSKrescodee*) – The function response code.

Groups *Parameters*

MSK_getsymmatinfo

```
MSKrescodee (MSKAPI MSK_getsymmatinfo) (  
    MSKtask_t task,  
    MSKint64t idx,  
    MSKint32t * dim,  
    MSKint64t * nz,  
    MSKsymmattypee * type)
```

MOSEK maintains a vector denoted by E of symmetric data matrices. This function makes it possible to obtain important information about a single matrix in E .

Parameters

- **task** (*MSKtask_t*) – An optimization task. (input)
- **idx** (*MSKint64t*) – Index of the matrix for which information is requested. (input)
- **dim** (*MSKint32t by reference*) – Returns the dimension of the requested matrix. (output)

- **nz** (*MSKint64t by reference*) – Returns the number of non-zeros in the requested matrix. (output)
- **type** (*MSKsymmatttypee by reference*) – Returns the type of the requested matrix. (output)

Return (*MSKrescodee*) – The function response code.

Groups *Problem data - semidefinite, Inspecting the task*

MSK_gettaskname

```
MSKrescodee (MSKAPI MSK_gettaskname) (
    MSKtask_t task,
    MSKint32t sizetaskname,
    char * taskname)
```

Obtains the name assigned to the task.

Parameters

- **task** (*MSKtask_t*) – An optimization task. (input)
- **sizetaskname** (*MSKint32t*) – Length of the **taskname** buffer. (input)
- **taskname** (*MSKstring_t*) – Returns the task name. (output)

Return (*MSKrescodee*) – The function response code.

Groups *Names, Inspecting the task*

MSK_gettasknamelen

```
MSKrescodee (MSKAPI MSK_gettasknamelen) (
    MSKtask_t task,
    MSKint32t * len)
```

Obtains the length the task name.

Parameters

- **task** (*MSKtask_t*) – An optimization task. (input)
- **len** (*MSKint32t by reference*) – Returns the length of the task name. (output)

Return (*MSKrescodee*) – The function response code.

Groups *Names, Inspecting the task*

MSK_getvarbound

```
MSKrescodee (MSKAPI MSK_getvarbound) (
    MSKtask_t task,
    MSKint32t i,
    MSKboundkeye * bk,
    MSKrealt * bl,
    MSKrealt * bu)
```

Obtains bound information for one variable.

Parameters

- **task** (*MSKtask_t*) – An optimization task. (input)
- **i** (*MSKint32t*) – Index of the variable for which the bound information should be obtained. (input)
- **bk** (*MSKboundkeye by reference*) – Bound keys. (output)
- **bl** (*MSKrealt by reference*) – Values for lower bounds. (output)
- **bu** (*MSKrealt by reference*) – Values for upper bounds. (output)

Return (*MSKrescodee*) – The function response code.

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

MSK_getvarboundslice

```
MSKrescodee (MSKAPI MSK_getvarboundslice) (
    MSKtask_t task,
    MSKint32t first,
    MSKint32t last,
    MSKboundkeye * bk,
    MSKrealt * bl,
    MSKrealt * bu)
```

Obtains bounds information for a slice of the variables.

Parameters

- **task** (*MSKtask_t*) – An optimization task. (input)
- **first** (*MSKint32t*) – First index in the sequence. (input)
- **last** (*MSKint32t*) – Last index plus 1 in the sequence. (input)
- **bk** (*MSKboundkeye**) – Bound keys. (output)
- **bl** (*MSKrealt**) – Values for lower bounds. (output)
- **bu** (*MSKrealt**) – Values for upper bounds. (output)

Return (*MSKrescodee*) – The function response code.

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

MSK_getvarname

```
MSKrescodee (MSKAPI MSK_getvarname) (
    MSKtask_t task,
    MSKint32t j,
    MSKint32t sizename,
    char * name)
```

Obtains the name of a variable.

Parameters

- **task** (*MSKtask_t*) – An optimization task. (input)
- **j** (*MSKint32t*) – Index of a variable. (input)
- **sizename** (*MSKint32t*) – The length of the buffer pointed to by the **name** argument. (input)
- **name** (*MSKstring_t*) – Returns the required name. (output)

Return (*MSKrescodee*) – The function response code.

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

MSK_getvarnameindex

```
MSKrescodee (MSKAPI MSK_getvarnameindex) (
    MSKtask_t task,
    const char * somename,
    MSKint32t * asgn,
    MSKint32t * index)
```

Checks whether the name **somename** has been assigned to any variable. If so, the index of the variable is reported.

Parameters

- **task** (*MSKtask_t*) – An optimization task. (input)
- **somename** (*MSKstring_t*) – The name which should be checked. (input)
- **asgn** (*MSKint32t by reference*) – Is non-zero if the name **somename** is assigned to a variable. (output)
- **index** (*MSKint32t by reference*) – If the name **somename** is assigned to a variable, then **index** is the index of the variable. (output)

Return (*MSKrescodee*) – The function response code.

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

MSK_getvarnamelen

```
MSKrescodee (MSKAPI MSK_getvarnamelen) (
    MSKtask_t task,
    MSKint32t i,
    MSKint32t * len)
```

Obtains the length of the name of a variable.

Parameters

- **task** (*MSKtask_t*) – An optimization task. (input)
- **i** (*MSKint32t*) – Index of a variable. (input)
- **len** (*MSKint32t by reference*) – Returns the length of the indicated name. (output)

Return (*MSKrescodee*) – The function response code.

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

MSK_getvartype

```
MSKrescodee (MSKAPI MSK_getvartype) (
    MSKtask_t task,
    MSKint32t j,
    MSKvariabletypee * vartype)
```

Gets the variable type of one variable.

Parameters

- **task** (*MSKtask_t*) – An optimization task. (input)
- **j** (*MSKint32t*) – Index of the variable. (input)
- **vartype** (*MSKvariabletypee by reference*) – Variable type of the *j*-th variable. (output)

Return (*MSKrescodee*) – The function response code.

Groups *Inspecting the task, Problem data - variables*

MSK_getvartypelist

```
MSKrescodee (MSKAPI MSK_getvartypelist) (
    MSKtask_t task,
    MSKint32t num,
    const MSKint32t * subj,
    MSKvariabletypee * vartype)
```

Obtains the variable type of one or more variables. Upon return **vartype[k]** is the variable type of variable **subj[k]**.

Parameters

- `task` (*MSKtask_t*) – An optimization task. (input)
- `num` (*MSKint32t*) – Number of variables for which the variable type should be obtained. (input)
- `subj` (*MSKint32t**) – A list of variable indexes. (input)
- `vartype` (*MSKvariabletypee**) – The variables types corresponding to the variables specified by `subj`. (output)

Return (*MSKrescodee*) – The function response code.

Groups *Inspecting the task, Problem data - variables*

MSK_getversion

```
MSKrescodee (MSKAPI MSK_getversion) (
    MSKint32t * major,
    MSKint32t * minor,
    MSKint32t * revision)
```

Obtains MOSEK version information.

Parameters

- `major` (*MSKint32t by reference*) – Major version number. (output)
- `minor` (*MSKint32t by reference*) – Minor version number. (output)
- `revision` (*MSKint32t by reference*) – Revision number. (output)

Return (*MSKrescodee*) – The function response code.

Groups *Versions*

MSK_getxc

```
MSKrescodee (MSKAPI MSK_getxc) (
    MSKtask_t task,
    MSKsoltypee whichsol,
    MSKrealt * xc)
```

Obtains the x^c vector for a solution.

Parameters

- `task` (*MSKtask_t*) – An optimization task. (input)
- `whichsol` (*MSKsoltypee*) – Selects a solution. (input)
- `xc` (*MSKrealt**) – Primal constraint solution. (output)

Return (*MSKrescodee*) – The function response code.

Groups *Solution - primal*

MSK_getxcslice

```
MSKrescodee (MSKAPI MSK_getxcslice) (
    MSKtask_t task,
    MSKsoltypee whichsol,
    MSKint32t first,
    MSKint32t last,
    MSKrealt * xc)
```

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

Parameters

- `task` (*MSKtask_t*) – An optimization task. (input)
- `whichsol` (*MSKsoltypee*) – Selects a solution. (input)
- `first` (*MSKint32t*) – First index in the sequence. (input)
- `last` (*MSKint32t*) – Last index plus 1 in the sequence. (input)

- xc (*MSKrealt**) – Primal constraint solution. (output)
- Return** (*MSKrescodee*) – The function response code.
- Groups** *Solution - primal*

MSK_getxx

```
MSKrescodee (MSKAPI MSK_getxx) (
    MSKtask_t task,
    MSKsoltypee whichsol,
    MSKrealt * xx)
```

Obtains the x^x vector for a solution.

Parameters

- task (*MSKtask_t*) – An optimization task. (input)
- whichsol (*MSKsoltypee*) – Selects a solution. (input)
- xx (*MSKrealt**) – Primal variable solution. (output)

Return (*MSKrescodee*) – The function response code.

Groups *Solution - primal*

MSK_getxxslice

```
MSKrescodee (MSKAPI MSK_getxxslice) (
    MSKtask_t task,
    MSKsoltypee whichsol,
    MSKint32t first,
    MSKint32t last,
    MSKrealt * xx)
```

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

Parameters

- task (*MSKtask_t*) – An optimization task. (input)
- whichsol (*MSKsoltypee*) – Selects a solution. (input)
- first (*MSKint32t*) – First index in the sequence. (input)
- last (*MSKint32t*) – Last index plus 1 in the sequence. (input)
- xx (*MSKrealt**) – Primal variable solution. (output)

Return (*MSKrescodee*) – The function response code.

Groups *Solution - primal*

MSK_gety

```
MSKrescodee (MSKAPI MSK_gety) (
    MSKtask_t task,
    MSKsoltypee whichsol,
    MSKrealt * y)
```

Obtains the y vector for a solution.

Parameters

- task (*MSKtask_t*) – An optimization task. (input)
- whichsol (*MSKsoltypee*) – Selects a solution. (input)
- y (*MSKrealt**) – Vector of dual variables corresponding to the constraints. (output)

Return (*MSKrescodee*) – The function response code.

Groups *Solution - dual*

MSK_getyslice

```
MSKrescodee (MSKAPI MSK_getyslice) (  
    MSKtask_t task,  
    MSKsoltypee whichsol,  
    MSKint32t first,  
    MSKint32t last,  
    MSKrealt * y)
```

Obtains a slice of the y vector for a solution.

Parameters

- **task** (*MSKtask_t*) – An optimization task. (input)
- **whichsol** (*MSKsoltypee*) – Selects a solution. (input)
- **first** (*MSKint32t*) – First index in the sequence. (input)
- **last** (*MSKint32t*) – Last index plus 1 in the sequence. (input)
- **y** (*MSKrealt**) – Vector of dual variables corresponding to the constraints. (output)

Return (*MSKrescodee*) – The function response code.

Groups *Solution - dual*

MSK_initbasissolve

```
MSKrescodee (MSKAPI MSK_initbasissolve) (  
    MSKtask_t task,  
    MSKint32t * basis)
```

Prepare a task for use with the *MSK_solvewithbasis* function.

This function should be called

- immediately before the first call to *MSK_solvewithbasis*, and
- immediately before any subsequent call to *MSK_solvewithbasis* if the task has been modified.

If the basis is singular i.e. not invertible, then the error *MSK_RES_ERR_BASIS_SINGULAR* is reported.

Parameters

- **task** (*MSKtask_t*) – An optimization task. (input)
- **basis** (*MSKint32t**) – The array of basis indexes to use. The array is interpreted as follows: If $\text{basis}[i] \leq \text{numcon} - 1$, then $x_{\text{basis}[i]}^c$ is in the basis at position i , otherwise $x_{\text{basis}[i] - \text{numcon}}$ is in the basis at position i . (output)

Return (*MSKrescodee*) – The function response code.

Groups *Solving systems with basis matrix*

MSK_inputdata

```
MSKrescodee (MSKAPI MSK_inputdata) (  
    MSKtask_t task,  
    MSKint32t maxnumcon,  
    MSKint32t maxnumvar,  
    MSKint32t numcon,  
    MSKint32t numvar,  
    const MSKrealt * c,  
    MSKrealt cfix,  
    const MSKint32t * aptrb,  
    const MSKint32t * aptre,  
    const MSKint32t * asub,  
    const MSKrealt * aval,
```

(continues on next page)

```

const MSKboundkeye * bkc,
const MSKrealt * blc,
const MSKrealt * buc,
const MSKboundkeye * bkc,
const MSKrealt * blx,
const MSKrealt * bux)

```

Input the linear part of an optimization problem.

The non-zeros of A are inputted column-wise in the format described in Section *Column or Row Ordered Sparse Matrix*.

For an explained code example see Section *Linear Optimization* and Section *Matrix Formats*.

Parameters

- **task** (*MSKtask_t*) – An optimization task. (input)
- **maxnumcon** (*MSKint32t*) – Number of preallocated constraints in the optimization task. (input)
- **maxnumvar** (*MSKint32t*) – Number of preallocated variables in the optimization task. (input)
- **numcon** (*MSKint32t*) – Number of constraints. (input)
- **numvar** (*MSKint32t*) – Number of variables. (input)
- **c** (*MSKrealt**) – Linear terms of the objective as a dense vector. The length is the number of variables. (input)
- **cfix** (*MSKrealt*) – Fixed term in the objective. (input)
- **aptrb** (*MSKint32t**) – Row or column start pointers. (input)
- **aptre** (*MSKint32t**) – Row or column end pointers. (input)
- **asub** (*MSKint32t**) – Coefficient subscripts. (input)
- **aval** (*MSKrealt**) – Coefficient values. (input)
- **bkc** (*MSKboundkeye**) – Bound keys for the constraints. (input)
- **blc** (*MSKrealt**) – Lower bounds for the constraints. (input)
- **buc** (*MSKrealt**) – Upper bounds for the constraints. (input)
- **bkc** (*MSKboundkeye**) – Bound keys for the variables. (input)
- **blx** (*MSKrealt**) – Lower bounds for the variables. (input)
- **bux** (*MSKrealt**) – Upper bounds for the variables. (input)

Return (*MSKrescodee*) – The function response code.

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

MSK_inputdata64

```

MSKrescodee (MSKAPI MSK_inputdata64) (
    MSKtask_t task,
    MSKint32t maxnumcon,
    MSKint32t maxnumvar,
    MSKint32t numcon,
    MSKint32t numvar,
    const MSKrealt * c,
    MSKrealt cfix,
    const MSKint64t * aptrb,
    const MSKint64t * aptre,
    const MSKint32t * asub,
    const MSKrealt * aval,
    const MSKboundkeye * bkc,
    const MSKrealt * blc,
    const MSKrealt * buc,
    const MSKboundkeye * bkc,
    const MSKrealt * blx,
    const MSKrealt * bux)

```

Input the linear part of an optimization problem.

The non-zeros of A are inputted column-wise in the format described in Section *Column or Row Ordered Sparse Matrix*.

For an explained code example see Section *Linear Optimization* and Section *Matrix Formats*.

Parameters

- **task** (*MSKtask_t*) – An optimization task. (input)
- **maxnumcon** (*MSKint32t*) – Number of preallocated constraints in the optimization task. (input)
- **maxnumvar** (*MSKint32t*) – Number of preallocated variables in the optimization task. (input)
- **numcon** (*MSKint32t*) – Number of constraints. (input)
- **numvar** (*MSKint32t*) – Number of variables. (input)
- **c** (*MSKrealt**) – Linear terms of the objective as a dense vector. The length is the number of variables. (input)
- **cfix** (*MSKrealt*) – Fixed term in the objective. (input)
- **aptrb** (*MSKint64t**) – Row or column start pointers. (input)
- **aptre** (*MSKint64t**) – Row or column end pointers. (input)
- **asub** (*MSKint32t**) – Coefficient subscripts. (input)
- **aval** (*MSKrealt**) – Coefficient values. (input)
- **bkc** (*MSKboundkeye**) – Bound keys for the constraints. (input)
- **blc** (*MSKrealt**) – Lower bounds for the constraints. (input)
- **buc** (*MSKrealt**) – Upper bounds for the constraints. (input)
- **bkx** (*MSKboundkeye**) – Bound keys for the variables. (input)
- **blx** (*MSKrealt**) – Lower bounds for the variables. (input)
- **bux** (*MSKrealt**) – Upper bounds for the variables. (input)

Return (*MSKrescodee*) – The function response code.

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

MSK_iparvaltosymnam

```
MSKrescodee (MSKAPI MSK_iparvaltosymnam) (
    MSKenv_t env,
    MSKiparam whichparam,
    MSKint32t whichvalue,
    char * symbolicname)
```

Obtains the symbolic name corresponding to a value that can be assigned to an integer parameter.

Parameters

- **env** (*MSKenv_t*) – The MOSEK environment. (input)
- **whichparam** (*MSKiparam*) – Which parameter. (input)
- **whichvalue** (*MSKint32t*) – Which value. (input)
- **symbolicname** (*MSKstring_t*) – The symbolic name corresponding to whichvalue. (output)

Return (*MSKrescodee*) – The function response code.

Groups *Parameters, Names*

MSK_isdoupaname

```
MSKrescodee (MSKAPI MSK_isdoupaname) (
    MSKtask_t task,
    const char * parname,
    MSKdparam * param)
```

Checks whether `parname` is a valid double parameter name.

Parameters

- `task` (*MSKtask_t*) – An optimization task. (input)
- `parname` (*MSKstring_t*) – Parameter name. (input)
- `param` (*MSKdparame by reference*) – Returns the parameter corresponding to the name, if one exists. (output)

Return (*MSKrescodee*) – The function response code.

Groups *Parameters, Names*

MSK_isinfinity

```
MSKboolean (MSKAPI MSK_isinfinity) (  
    MSKrealt value)
```

Return true if `value` is considered infinity by **MOSEK**.

Parameters `value` (*MSKrealt*) – The value to be checked (input)

Return (*MSKboolean*) – True if the value represents infinity.

MSK_isintparname

```
MSKrescodee (MSKAPI MSK_isintparname) (  
    MSKtask_t task,  
    const char * parname,  
    MSKiparam * param)
```

Checks whether `parname` is a valid integer parameter name.

Parameters

- `task` (*MSKtask_t*) – An optimization task. (input)
- `parname` (*MSKstring_t*) – Parameter name. (input)
- `param` (*MSKiparame by reference*) – Returns the parameter corresponding to the name, if one exists. (output)

Return (*MSKrescodee*) – The function response code.

Groups *Parameters, Names*

MSK_isstrparname

```
MSKrescodee (MSKAPI MSK_isstrparname) (  
    MSKtask_t task,  
    const char * parname,  
    MSKsparam * param)
```

Checks whether `parname` is a valid string parameter name.

Parameters

- `task` (*MSKtask_t*) – An optimization task. (input)
- `parname` (*MSKstring_t*) – Parameter name. (input)
- `param` (*MSKsparame by reference*) – Returns the parameter corresponding to the name, if one exists. (output)

Return (*MSKrescodee*) – The function response code.

Groups *Parameters, Names*

MSK_licensecleanup

```
MSKrescodee (MSKAPI MSK_licensecleanup) (
void)
```

Stops all threads and deletes all handles used by the license system. If this function is called, it must be called as the last **MOSEK** API call. No other **MOSEK** API calls are valid after this.

Return (*MSKrescodee*) – The function response code.

Groups *License system*

MSK_linkfiletoenvstream

```
MSKrescodee (MSKAPI MSK_linkfiletoenvstream) (
    MSKenv_t env,
    MSKstreamtypee whichstream,
    const char * filename,
    MSKint32t append)
```

Sends all output from the stream defined by **whichstream** to the file given by **filename**.

Parameters

- **env** (*MSKenv_t*) – The MOSEK environment. (input)
- **whichstream** (*MSKstreamtypee*) – Index of the stream. (input)
- **filename** (*MSKstring_t*) – A valid file name. (input)
- **append** (*MSKint32t*) – If this argument is 0 the file will be overwritten, otherwise it will be appended to. (input)

Return (*MSKrescodee*) – The function response code.

Groups *Logging*

MSK_linkfiletotaskstream

```
MSKrescodee (MSKAPI MSK_linkfiletotaskstream) (
    MSKtask_t task,
    MSKstreamtypee whichstream,
    const char * filename,
    MSKint32t append)
```

Directs all output from a task stream **whichstream** to a file **filename**.

Parameters

- **task** (*MSKtask_t*) – An optimization task. (input)
- **whichstream** (*MSKstreamtypee*) – Index of the stream. (input)
- **filename** (*MSKstring_t*) – A valid file name. (input)
- **append** (*MSKint32t*) – If this argument is 0 the output file will be overwritten, otherwise it will be appended to. (input)

Return (*MSKrescodee*) – The function response code.

Groups *Logging*

MSK_linkfunctoenvstream

```
MSKrescodee (MSKAPI MSK_linkfunctoenvstream) (
    MSKenv_t env,
    MSKstreamtypee whichstream,
    MSKuserhandle_t handle,
    MSKstreamfunc func)
```

Connects a user-defined function to a stream.

Parameters

- `env` (*MSKenv_t*) – The MOSEK environment. (input)
- `whichstream` (*MSKstreamtypee*) – Index of the stream. (input)
- `handle` (*MSKuserhandle_t*) – A user-defined handle which is passed to the user-defined function `func`. (input)
- `func` (*MSKstreamfunc*) – All output to the stream `whichstream` is passed to `func`. (input)

Return (*MSKrescodee*) – The function response code.

Groups *Logging, Callback*

MSK_linkfunctotaskstream

```
MSKrescodee (MSKAPI MSK_linkfunctotaskstream) (
    MSKtask_t task,
    MSKstreamtypee whichstream,
    MSKuserhandle_t handle,
    MSKstreamfunc func)
```

Connects a user-defined function to a task stream.

Parameters

- `task` (*MSKtask_t*) – An optimization task. (input)
- `whichstream` (*MSKstreamtypee*) – Index of the stream. (input)
- `handle` (*MSKuserhandle_t*) – A user-defined handle which is passed to the user-defined function `func`. (input)
- `func` (*MSKstreamfunc*) – All output to the stream `whichstream` is passed to `func`. (input)

Return (*MSKrescodee*) – The function response code.

Groups *Logging, Callback*

MSK_makeemptytask

```
MSKrescodee (MSKAPI MSK_makeemptytask) (
    MSKenv_t env,
    MSKtask_t * task)
```

Creates a new optimization task.

Parameters

- `env` (*MSKenv_t*) – The MOSEK environment. (input)
- `task` (*MSKtask_t by reference*) – An optimization task. (output)

Return (*MSKrescodee*) – The function response code.

Groups *Environment and task management*

MSK_makeenv

```
MSKrescodee (MSKAPI MSK_makeenv) (
    MSKenv_t * env,
    const char * dbgfile)
```

Creates a new **MOSEK** environment. The environment must be shared among all tasks in a program.

Parameters

- `env` (*MSKenv_t by reference*) – The MOSEK environment. (output)
- `dbgfile` (*MSKstring_t*) – A user-defined file debug file. (input)

Return (*MSKrescodee*) – The function response code.

Groups *Environment and task management*

MSK_makeenvalloc

```
MSKrescodee (MSKAPI MSK_makeenvalloc) (  
    MSKenv_t * env,  
    MSKuserhandle_t usrptr,  
    MSKmallocfunc usrmalloc,  
    MSKcallocfunc usrcalloc,  
    MSKreallocfunc usrrealloc,  
    MSKfreefunc usrfree,  
    const char * dbgfile)
```

Creates a new **MOSEK** environment with user-defined memory management functions. The environment must be shared among all tasks in a program.

Parameters

- `env` (*MSKenv_t by reference*) – The MOSEK environment. (output)
- `usrptr` (*MSKuserhandle_t*) – A pointer to a user-defined data structure. The pointer is fed into `usrmalloc` and `usrfree`. (input)
- `usrmalloc` (*MSKmallocfunc*) – A user-defined `malloc` function or a NULL pointer. (input)
- `usrcalloc` (*MSKcallocfunc*) – A user-defined `calloc` function or a NULL pointer. (input)
- `usrrealloc` (*MSKreallocfunc*) – A user-defined `realloc` function or a NULL pointer. (input)
- `usrfree` (*MSKfreefunc*) – A user-defined `free` function which is used to deallocate space allocated by `usrmalloc`. This function must be defined if `usrmalloc != null`. (input)
- `dbgfile` (*MSKstring_t*) – A user-defined file debug file. (input)

Return (*MSKrescodee*) – The function response code.

Groups *Environment and task management*

MSK_maketask

```
MSKrescodee (MSKAPI MSK_maketask) (  
    MSKenv_t env,  
    MSKint32t maxnumcon,  
    MSKint32t maxnumvar,  
    MSKtask_t * task)
```

Creates a new task.

Parameters

- `env` (*MSKenv_t*) – The MOSEK environment. (input)
- `maxnumcon` (*MSKint32t*) – An optional estimate on the maximum number of constraints in the task. Can be 0 if no such estimate is known. (input)
- `maxnumvar` (*MSKint32t*) – An optional estimate on the maximum number of variables in the task. Can be 0 if no such estimate is known. (input)
- `task` (*MSKtask_t by reference*) – An optimization task. (output)

Return (*MSKrescodee*) – The function response code.

Groups *Environment and task management*

MSK_onesolutionsummary

```
MSKrescodee (MSKAPI MSK_onesolutionsummary) (
    MSKtask_t task,
    MSKstreamtypee whichstream,
    MSKsoltypee whichsol)
```

Prints a short summary of a specified solution.

Parameters

- **task** (*MSKtask_t*) – An optimization task. (input)
- **whichstream** (*MSKstreamtypee*) – Index of the stream. (input)
- **whichsol** (*MSKsoltypee*) – Selects a solution. (input)

Return (*MSKrescodee*) – The function response code.

Groups *Logging, Solution information*

MSK_optimize

```
MSKrescodee (MSKAPI MSK_optimize) (
    MSKtask_t task)
```

Calls the optimizer. Depending on the problem type and the selected optimizer this will call one of the optimizers in **MOSEK**. By default the interior point optimizer will be selected for continuous problems. The optimizer may be selected manually by setting the parameter *MSK_IPAR_OPTIMIZER*.

Response codes come in three categories:

- **Errors**: Indicate that an error has occurred during the optimization, e.g the optimizer has run out of memory (*MSK_RES_ERR_SPACE*).
- **Warnings**: Less fatal than errors. E.g *MSK_RES_WRN_LARGE_CJ* indicating possibly problematic problem data.
- **Termination codes**: Relaying information about the conditions under which the optimizer terminated. E.g *MSK_RES_TRM_MAX_ITERATIONS* indicates that the optimizer finished because it reached the maximum number of iterations specified by the user.

Parameters **task** (*MSKtask_t*) – An optimization task. (input)

Return (*MSKrescodee*) – The function response code.

Groups *Optimization*

MSK_optimizermt

```
MSKrescodee (MSKAPI MSK_optimizermt) (
    MSKtask_t task,
    const char * server,
    const char * port,
    MSKrescodee * trmcode)
```

Offload the optimization task to a solver server defined by **server:port**. The call will block until a result is available or the connection closes.

If the string parameter *MSK_SPAR_REMOTE_ACCESS_TOKEN* is not blank, it will be passed to the server as authentication.

Parameters

- **task** (*MSKtask_t*) – An optimization task. (input)
- **server** (*MSKstring_t*) – Name or IP address of the solver server. (input)
- **port** (*MSKstring_t*) – Network port of the solver server. (input)
- **trmcode** (*MSKrescodee by reference*) – Is either *MSK_RES_OK* or a termination response code. (output)

Return (*MSKrescodee*) – The function response code.

Groups *Remote optimization*

MSK_optimizersummary

```
MSKrescodee (MSKAPI MSK_optimizersummary) (  
    MSKtask_t task,  
    MSKstreamtypee whichstream)
```

Prints a short summary with optimizer statistics from last optimization.

Parameters

- **task** (*MSKtask_t*) – An optimization task. (input)
- **whichstream** (*MSKstreamtypee*) – Index of the stream. (input)

Return (*MSKrescodee*) – The function response code.

Groups *Logging*

MSK_optimizetrm

```
MSKrescodee (MSKAPI MSK_optimizetrm) (  
    MSKtask_t task,  
    MSKrescodee * trmcode)
```

Calls the optimizer. Depending on the problem type and the selected optimizer this will call one of the optimizers in **MOSEK**. By default the interior point optimizer will be selected for continuous problems. The optimizer may be selected manually by setting the parameter *MSK_IPAR_OPTIMIZER*.

This function is equivalent to *MSK_optimize* except for the handling of return values. This function returns errors on the left hand side. Warnings are not returned and termination codes are returned through the separate argument **trmcode**.

Parameters

- **task** (*MSKtask_t*) – An optimization task. (input)
- **trmcode** (*MSKrescodee by reference*) – Is either *MSK_RES_OK* or a termination response code. (output)

Return (*MSKrescodee*) – The function response code.

Groups *Optimization*

MSK_potrf

```
MSKrescodee (MSKAPI MSK_potrf) (  
    MSKenv_t env,  
    MSKuploe uplo,  
    MSKint32t n,  
    MSKrealt * a)
```

Computes a Cholesky factorization of a real symmetric positive definite dense matrix.

Parameters

- **env** (*MSKenv_t*) – The MOSEK environment. (input)
- **uplo** (*MSKuploe*) – Indicates whether the upper or lower triangular part of the matrix is stored. (input)
- **n** (*MSKint32t*) – Dimension of the symmetric matrix. (input)
- **a** (*MSKrealt**) – A symmetric matrix stored in column-major order. Only the lower or the upper triangular part is used, accordingly with the **uplo** parameter. It will contain the result on exit. (input/output)

Return (*MSKrescodee*) – The function response code.

Groups *Linear algebra*

MSK_primalrepair

```
MSKrescodee (MSKAPI MSK_primalrepair) (  
    MSKtask_t task,  
    const MSKrealt * wlc,  
    const MSKrealt * wuc,  
    const MSKrealt * wlx,  
    const MSKrealt * wux)
```

The function repairs a primal infeasible optimization problem by adjusting the bounds on the constraints and variables where the adjustment is computed as the minimal weighted sum of relaxations to the bounds on the constraints and variables. Observe the function only repairs the problem but does not solve it. If an optimal solution is required the problem should be optimized after the repair.

The function is applicable to linear and conic problems possibly with integer variables.

Observe that when computing the minimal weighted relaxation the termination tolerance specified by the parameters of the task is employed. For instance the parameter *MSK_IPAR_MIO_MODE* can be used to make **MOSEK** ignore the integer constraints during the repair which usually leads to a much faster repair. However, the drawback is of course that the repaired problem may not have an integer feasible solution.

Note the function modifies the task in place. If this is not desired, then apply the function to a cloned task.

Parameters

- **task** (*MSKtask_t*) – An optimization task. (input)
- **wlc** (*MSKrealt**) – $(w_l^c)_i$ is the weight associated with relaxing the lower bound on constraint i . If the weight is negative, then the lower bound is not relaxed. Moreover, if the argument is NULL, then all the weights are assumed to be 1. (input)
- **wuc** (*MSKrealt**) – $(w_u^c)_i$ is the weight associated with relaxing the upper bound on constraint i . If the weight is negative, then the upper bound is not relaxed. Moreover, if the argument is NULL, then all the weights are assumed to be 1. (input)
- **wlx** (*MSKrealt**) – $(w_l^x)_j$ is the weight associated with relaxing the lower bound on variable j . If the weight is negative, then the lower bound is not relaxed. Moreover, if the argument is NULL, then all the weights are assumed to be 1. (input)
- **wux** (*MSKrealt**) – $(w_u^x)_j$ is the weight associated with relaxing the upper bound on variable j . If the weight is negative, then the upper bound is not relaxed. Moreover, if the argument is NULL, then all the weights are assumed to be 1. (input)

Return (*MSKrescodee*) – The function response code.

Groups *Infeasibility diagnostic*

MSK_primalsensitivity

```
MSKrescodee (MSKAPI MSK_primalsensitivity) (  
    MSKtask_t task,  
    MSKint32t numi,  
    const MSKint32t * subi,  
    const MSKmarke * marki,  
    MSKint32t numj,  
    const MSKint32t * subj,  
    const MSKmarke * markj,  
    MSKrealt * leftpricei,  
    MSKrealt * rightpricei,
```

(continues on next page)

```

MSKrealt * leftrangei,
MSKrealt * rightrangei,
MSKrealt * leftpricej,
MSKrealt * rightpricej,
MSKrealt * leftrangej,
MSKrealt * rightrangej)

```

Calculates sensitivity information for bounds on variables and constraints. For details on sensitivity analysis, the definitions of *shadow price* and *linearity interval* and an example see Section [Sensitivity Analysis](#).

The type of sensitivity analysis to be performed (basis or optimal partition) is controlled by the parameter `MSK_IPAR_SENSITIVITY_TYPE`.

Parameters

- `task` (`MSKtask_t`) – An optimization task. (input)
- `numi` (`MSKint32t`) – Number of bounds on constraints to be analyzed. Length of `subi` and `marki`. (input)
- `subi` (`MSKint32t*`) – Indexes of constraints to analyze. (input)
- `marki` (`MSKmarke*`) – The value of `marki[i]` indicates for which bound of constraint `subi[i]` sensitivity analysis is performed. If `marki[i] = MSK_MARK_UP` the upper bound of constraint `subi[i]` is analyzed, and if `marki[i] = MSK_MARK_LO` the lower bound is analyzed. If `subi[i]` is an equality constraint, either `MSK_MARK_LO` or `MSK_MARK_UP` can be used to select the constraint for sensitivity analysis. (input)
- `numj` (`MSKint32t`) – Number of bounds on variables to be analyzed. Length of `subj` and `markj`. (input)
- `subj` (`MSKint32t*`) – Indexes of variables to analyze. (input)
- `markj` (`MSKmarke*`) – The value of `markj[j]` indicates for which bound of variable `subj[j]` sensitivity analysis is performed. If `markj[j] = MSK_MARK_UP` the upper bound of variable `subj[j]` is analyzed, and if `markj[j] = MSK_MARK_LO` the lower bound is analyzed. If `subj[j]` is a fixed variable, either `MSK_MARK_LO` or `MSK_MARK_UP` can be used to select the bound for sensitivity analysis. (input)
- `leftpricei` (`MSKrealt*`) – `leftpricei[i]` is the left shadow price for the bound `marki[i]` of constraint `subi[i]`. (output)
- `rightpricei` (`MSKrealt*`) – `rightpricei[i]` is the right shadow price for the bound `marki[i]` of constraint `subi[i]`. (output)
- `leftrangei` (`MSKrealt*`) – `leftrangei[i]` is the left range β_1 for the bound `marki[i]` of constraint `subi[i]`. (output)
- `rightrangei` (`MSKrealt*`) – `rightrangei[i]` is the right range β_2 for the bound `marki[i]` of constraint `subi[i]`. (output)
- `leftpricej` (`MSKrealt*`) – `leftpricej[j]` is the left shadow price for the bound `markj[j]` of variable `subj[j]`. (output)
- `rightpricej` (`MSKrealt*`) – `rightpricej[j]` is the right shadow price for the bound `markj[j]` of variable `subj[j]`. (output)
- `leftrangej` (`MSKrealt*`) – `leftrangej[j]` is the left range β_1 for the bound `markj[j]` of variable `subj[j]`. (output)
- `rightrangej` (`MSKrealt*`) – `rightrangej[j]` is the right range β_2 for the bound `markj[j]` of variable `subj[j]`. (output)

Return (`MSKrescodee`) – The function response code.

Groups [Sensitivity analysis](#)

`MSK_printparam`

```
MSKrescodee (MSKAPI MSK_printparam) (
    MSKtask_t task)
```

Prints the current parameter settings to the message stream.

Parameters *task* (*MSKtask_t*) – An optimization task. (input)

Return (*MSKrescodee*) – The function response code.

Groups *Inspecting the task, Logging*

MSK_probtypetostr

```
MSKrescodee (MSKAPI MSK_probtypetostr) (
    MSKtask_t task,
    MSKproblemtypee probtype,
    char * str)
```

Obtains a string containing the name of a given problem type.

Parameters

- *task* (*MSKtask_t*) – An optimization task. (input)
- *probtype* (*MSKproblemtypee*) – Problem type. (input)
- *str* (*MSKstring_t*) – String corresponding to the problem type key *probtype*. (output)

Return (*MSKrescodee*) – The function response code.

Groups *Inspecting the task, Names*

MSK_prostatostr

```
MSKrescodee (MSKAPI MSK_prostatostr) (
    MSKtask_t task,
    MSKprosta_e prosta,
    char * str)
```

Obtains a string containing the name of a given problem status.

Parameters

- *task* (*MSKtask_t*) – An optimization task. (input)
- *prosta* (*MSKprosta_e*) – Problem status. (input)
- *str* (*MSKstring_t*) – String corresponding to the status key *prosta*. (output)

Return (*MSKrescodee*) – The function response code.

Groups *Solution information, Names*

MSK_putacol

```
MSKrescodee (MSKAPI MSK_putacol) (
    MSKtask_t task,
    MSKint32t j,
    MSKint32t nzj,
    const MSKint32t * subj,
    const MSKrealt * valj)
```

Change one column of the linear constraint matrix *A*. Resets all the elements in column *j* to zero and then sets

$$a_{\text{subj}[k],j} = \text{valj}[k], \quad k = 0, \dots, \text{nzj} - 1.$$

Parameters

- *task* (*MSKtask_t*) – An optimization task. (input)

- j (*MSKint32t*) – Index of a column in A . (input)
- nzj (*MSKint32t*) – Number of non-zeros in column j of A . (input)
- $subj$ (*MSKint32t**) – Row indexes of non-zero values in column j of A . (input)
- $valj$ (*MSKrealt**) – New non-zero values of column j in A . (input)

Return (*MSKrescodee*) – The function response code.

Groups *Problem data - linear part*

MSK_putacollist

```
MSKrescodee (MSKAPI MSK_putacollist) (
    MSKtask_t task,
    MSKint32t num,
    const MSKint32t * sub,
    const MSKint32t * ptrb,
    const MSKint32t * ptre,
    const MSKint32t * asub,
    const MSKrealt * aval)
```

Change a set of columns in the linear constraint matrix A with data in sparse triplet format. The requested columns are set to zero and then updated with:

$$\text{for } i = 0, \dots, num - 1 \\ a_{asub[k], sub[i]} = aval[k], \quad k = ptrb[i], \dots, ptre[i] - 1.$$

Parameters

- $task$ (*MSKtask_t*) – An optimization task. (input)
- num (*MSKint32t*) – Number of columns of A to replace. (input)
- sub (*MSKint32t**) – Indexes of columns that should be replaced, no duplicates. (input)
- $ptrb$ (*MSKint32t**) – Array of pointers to the first element in each column. (input)
- $ptre$ (*MSKint32t**) – Array of pointers to the last element plus one in each column. (input)
- $asub$ (*MSKint32t**) – Row indexes of new elements. (input)
- $aval$ (*MSKrealt**) – Coefficient values. (input)

Return (*MSKrescodee*) – The function response code.

Groups *Problem data - linear part*

MSK_putacollist64

```
MSKrescodee (MSKAPI MSK_putacollist64) (
    MSKtask_t task,
    MSKint32t num,
    const MSKint32t * sub,
    const MSKint64t * ptrb,
    const MSKint64t * ptre,
    const MSKint32t * asub,
    const MSKrealt * aval)
```

Change a set of columns in the linear constraint matrix A with data in sparse triplet format. The requested columns are set to zero and then updated with:

$$\text{for } i = 0, \dots, num - 1 \\ a_{asub[k], sub[i]} = aval[k], \quad k = ptrb[i], \dots, ptre[i] - 1.$$

Parameters

- $task$ (*MSKtask_t*) – An optimization task. (input)
- num (*MSKint32t*) – Number of columns of A to replace. (input)

- `sub` (*MSKint32t**) – Indexes of columns that should be replaced, no duplicates. (input)
- `ptrb` (*MSKint64t**) – Array of pointers to the first element in each column. (input)
- `ptre` (*MSKint64t**) – Array of pointers to the last element plus one in each column. (input)
- `asub` (*MSKint32t**) – Row indexes of new elements. (input)
- `aval` (*MSKrealt**) – Coefficient values. (input)

Return (*MSKrescodee*) – The function response code.

Groups *Problem data - linear part*

MSK_putacolslice

```
MSKrescodee (MSKAPI MSK_putacolslice) (
    MSKtask_t task,
    MSKint32t first,
    MSKint32t last,
    const MSKint32t * ptrb,
    const MSKint32t * ptre,
    const MSKint32t * asub,
    const MSKrealt * aval)
```

Change a slice of columns in the linear constraint matrix A with data in sparse triplet format. The requested columns are set to zero and then updated with:

$$\begin{aligned} \text{for } i = \text{first}, \dots, \text{last} - 1 \\ a_{\text{asub}[k], i} = \text{aval}[k], \quad k = \text{ptrb}[i], \dots, \text{ptre}[i] - 1. \end{aligned}$$

Parameters

- `task` (*MSKtask_t*) – An optimization task. (input)
- `first` (*MSKint32t*) – First column in the slice. (input)
- `last` (*MSKint32t*) – Last column plus one in the slice. (input)
- `ptrb` (*MSKint32t**) – Array of pointers to the first element in each column. (input)
- `ptre` (*MSKint32t**) – Array of pointers to the last element plus one in each column. (input)
- `asub` (*MSKint32t**) – Row indexes of new elements. (input)
- `aval` (*MSKrealt**) – Coefficient values. (input)

Return (*MSKrescodee*) – The function response code.

Groups *Problem data - linear part*

MSK_putacolslice64

```
MSKrescodee (MSKAPI MSK_putacolslice64) (
    MSKtask_t task,
    MSKint32t first,
    MSKint32t last,
    const MSKint64t * ptrb,
    const MSKint64t * ptre,
    const MSKint32t * asub,
    const MSKrealt * aval)
```

Change a slice of columns in the linear constraint matrix A with data in sparse triplet format. The requested columns are set to zero and then updated with:

$$\begin{aligned} \text{for } i = \text{first}, \dots, \text{last} - 1 \\ a_{\text{asub}[k], i} = \text{aval}[k], \quad k = \text{ptrb}[i], \dots, \text{ptre}[i] - 1. \end{aligned}$$

Parameters

- **task** (*MSKtask_t*) – An optimization task. (input)
- **first** (*MSKint32t*) – First column in the slice. (input)
- **last** (*MSKint32t*) – Last column plus one in the slice. (input)
- **ptrb** (*MSKint64t**) – Array of pointers to the first element in each column. (input)
- **ptre** (*MSKint64t**) – Array of pointers to the last element plus one in each column. (input)
- **asub** (*MSKint32t**) – Row indexes of new elements. (input)
- **aval** (*MSKrealt**) – Coefficient values. (input)

Return (*MSKrescodee*) – The function response code.

Groups *Problem data - linear part*

MSK_putaij

```
MSKrescodee (MSKAPI MSK_putaij) (  
    MSKtask_t task,  
    MSKint32t i,  
    MSKint32t j,  
    MSKrealt aij)
```

Changes a coefficient in the linear coefficient matrix A using the method

$$a_{i,j} = \text{aij}.$$

Parameters

- **task** (*MSKtask_t*) – An optimization task. (input)
- **i** (*MSKint32t*) – Constraint (row) index. (input)
- **j** (*MSKint32t*) – Variable (column) index. (input)
- **aij** (*MSKrealt*) – New coefficient for $a_{i,j}$. (input)

Return (*MSKrescodee*) – The function response code.

Groups *Problem data - linear part*

MSK_putaijlist

```
MSKrescodee (MSKAPI MSK_putaijlist) (  
    MSKtask_t task,  
    MSKint32t num,  
    const MSKint32t * subi,  
    const MSKint32t * subj,  
    const MSKrealt * valij)
```

Changes one or more coefficients in A using the method

$$a_{\text{subi}[k], \text{subj}[k]} = \text{valij}[k], \quad k = 0, \dots, \text{num} - 1.$$

Duplicates are not allowed.

Parameters

- **task** (*MSKtask_t*) – An optimization task. (input)
- **num** (*MSKint32t*) – Number of coefficients that should be changed. (input)
- **subi** (*MSKint32t**) – Constraint (row) indices. (input)
- **subj** (*MSKint32t**) – Variable (column) indices. (input)
- **valij** (*MSKrealt**) – New coefficient values for $a_{i,j}$. (input)

Return (*MSKrescodee*) – The function response code.

Groups *Problem data - linear part*

MSK_putaijlist64

```
MSKrescodee (MSKAPI MSK_putaijlist64) (  
    MSKtask_t task,  
    MSKint64t num,  
    const MSKint32t * subi,  
    const MSKint32t * subj,  
    const MSKrealt * valij)
```

Changes one or more coefficients in A using the method

$$a_{\text{subi}[k], \text{subj}[k]} = \text{valij}[k], \quad k = 0, \dots, \text{num} - 1.$$

Duplicates are not allowed.

Parameters

- `task` (*MSKtask_t*) – An optimization task. (input)
- `num` (*MSKint64t*) – Number of coefficients that should be changed. (input)
- `subi` (*MSKint32t**) – Constraint (row) indices. (input)
- `subj` (*MSKint32t**) – Variable (column) indices. (input)
- `valij` (*MSKrealt**) – New coefficient values for $a_{i,j}$. (input)

Return (*MSKrescodee*) – The function response code.

Groups *Problem data - linear part*

MSK_putarow

```
MSKrescodee (MSKAPI MSK_putarow) (  
    MSKtask_t task,  
    MSKint32t i,  
    MSKint32t nzi,  
    const MSKint32t * subi,  
    const MSKrealt * vali)
```

Change one row of the linear constraint matrix A . Resets all the elements in row i to zero and then sets

$$a_{i, \text{subi}[k]} = \text{vali}[k], \quad k = 0, \dots, \text{nzi} - 1.$$

Parameters

- `task` (*MSKtask_t*) – An optimization task. (input)
- `i` (*MSKint32t*) – Index of a row in A . (input)
- `nzi` (*MSKint32t*) – Number of non-zeros in row i of A . (input)
- `subi` (*MSKint32t**) – Column indexes of non-zero values in row i of A . (input)
- `vali` (*MSKrealt**) – New non-zero values of row i in A . (input)

Return (*MSKrescodee*) – The function response code.

Groups *Problem data - linear part*

MSK_putarowlist

```
MSKrescodee (MSKAPI MSK_putarowlist) (  
    MSKtask_t task,  
    MSKint32t num,  
    const MSKint32t * sub,  
    const MSKint32t * ptrb,  
    const MSKint32t * ptre,  
    const MSKint32t * asub,  
    const MSKrealt * aval)
```

Change a set of rows in the linear constraint matrix A with data in sparse triplet format. The requested rows are set to zero and then updated with:

```
for i = 0, ..., num - 1
    asub[i],asub[k] = aval[k], k = ptrb[i], ..., ptre[i] - 1.
```

Parameters

- task (*MSKtask_t*) – An optimization task. (input)
- num (*MSKint32t*) – Number of rows of A to replace. (input)
- sub (*MSKint32t**) – Indexes of rows that should be replaced, no duplicates. (input)
- ptrb (*MSKint32t**) – Array of pointers to the first element in each row. (input)
- ptre (*MSKint32t**) – Array of pointers to the last element plus one in each row. (input)
- asub (*MSKint32t**) – Column indexes of new elements. (input)
- aval (*MSKrealt**) – Coefficient values. (input)

Return (*MSKrescodee*) – The function response code.

Groups *Problem data - linear part*

MSK_putarowlist64

```
MSKrescodee (MSKAPI MSK_putarowlist64) (
    MSKtask_t task,
    MSKint32t num,
    const MSKint32t * sub,
    const MSKint64t * ptrb,
    const MSKint64t * ptre,
    const MSKint32t * asub,
    const MSKrealt * aval)
```

Change a set of rows in the linear constraint matrix A with data in sparse triplet format. The requested rows are set to zero and then updated with:

```
for i = 0, ..., num - 1
    asub[i],asub[k] = aval[k], k = ptrb[i], ..., ptre[i] - 1.
```

Parameters

- task (*MSKtask_t*) – An optimization task. (input)
- num (*MSKint32t*) – Number of rows of A to replace. (input)
- sub (*MSKint32t**) – Indexes of rows that should be replaced, no duplicates. (input)
- ptrb (*MSKint64t**) – Array of pointers to the first element in each row. (input)
- ptre (*MSKint64t**) – Array of pointers to the last element plus one in each row. (input)
- asub (*MSKint32t**) – Column indexes of new elements. (input)
- aval (*MSKrealt**) – Coefficient values. (input)

Return (*MSKrescodee*) – The function response code.

Groups *Problem data - linear part*

MSK_putarowslice

```
MSKrescodee (MSKAPI MSK_putarowslice) (
    MSKtask_t task,
    MSKint32t first,
    MSKint32t last,
    const MSKint32t * ptrb,
    const MSKint32t * ptre,
    const MSKint32t * asub,
    const MSKrealt * aval)
```

Change a slice of rows in the linear constraint matrix A with data in sparse triplet format. The requested columns are set to zero and then updated with:

```
for i = first,...,last - 1
    asub[i],asub[k] = aval[k], k = ptrb[i],...,ptre[i] - 1.
```

Parameters

- task (*MSKtask_t*) – An optimization task. (input)
- first (*MSKint32t*) – First row in the slice. (input)
- last (*MSKint32t*) – Last row plus one in the slice. (input)
- ptrb (*MSKint32t**) – Array of pointers to the first element in each row. (input)
- ptre (*MSKint32t**) – Array of pointers to the last element plus one in each row. (input)
- asub (*MSKint32t**) – Column indexes of new elements. (input)
- aval (*MSKrealt**) – Coefficient values. (input)

Return (*MSKrescodee*) – The function response code.

Groups *Problem data - linear part*

MSK_putarowslice64

```
MSKrescodee (MSKAPI MSK_putarowslice64) (
    MSKtask_t task,
    MSKint32t first,
    MSKint32t last,
    const MSKint64t * ptrb,
    const MSKint64t * ptre,
    const MSKint32t * asub,
    const MSKrealt * aval)
```

Change a slice of rows in the linear constraint matrix A with data in sparse triplet format. The requested columns are set to zero and then updated with:

```
for i = first,...,last - 1
    asub[i],asub[k] = aval[k], k = ptrb[i],...,ptre[i] - 1.
```

Parameters

- task (*MSKtask_t*) – An optimization task. (input)
- first (*MSKint32t*) – First row in the slice. (input)
- last (*MSKint32t*) – Last row plus one in the slice. (input)
- ptrb (*MSKint64t**) – Array of pointers to the first element in each row. (input)
- ptre (*MSKint64t**) – Array of pointers to the last element plus one in each row. (input)
- asub (*MSKint32t**) – Column indexes of new elements. (input)
- aval (*MSKrealt**) – Coefficient values. (input)

Return (*MSKrescodee*) – The function response code.

Groups *Problem data - linear part*

MSK_putatruncatetol

```
MSKrescodee (MSKAPI MSK_putatruncatetol) (
    MSKtask_t task,
    MSKrealt tolzero)
```

Truncates (sets to zero) all elements in A that satisfy

$$|a_{i,j}| \leq \text{tolzero}.$$

Parameters

- **task** (*MSKtask_t*) – An optimization task. (input)
- **tolzero** (*MSKrealt*) – Truncation tolerance. (input)

Return (*MSKrescodee*) – The function response code.

Groups *Problem data - linear part*

MSK_putbarablocktriplet

```
MSKrescodee (MSKAPI MSK_putbarablocktriplet) (  
    MSKtask_t task,  
    MSKint64t num,  
    const MSKint32t * subi,  
    const MSKint32t * subj,  
    const MSKint32t * subk,  
    const MSKint32t * subl,  
    const MSKrealt * valijkl)
```

Inputs the \bar{A} matrix in block triplet form.

Parameters

- **task** (*MSKtask_t*) – An optimization task. (input)
- **num** (*MSKint64t*) – Number of elements in the block triplet form. (input)
- **subi** (*MSKint32t**) – Constraint index. (input)
- **subj** (*MSKint32t**) – Symmetric matrix variable index. (input)
- **subk** (*MSKint32t**) – Block row index. (input)
- **subl** (*MSKint32t**) – Block column index. (input)
- **valijkl** (*MSKrealt**) – The numerical value associated with each block triplet. (input)

Return (*MSKrescodee*) – The function response code.

Groups *Problem data - semidefinite*

MSK_putbaraij

```
MSKrescodee (MSKAPI MSK_putbaraij) (  
    MSKtask_t task,  
    MSKint32t i,  
    MSKint32t j,  
    MSKint64t num,  
    const MSKint64t * sub,  
    const MSKrealt * weights)
```

This function sets one element in the \bar{A} matrix.

Each element in the \bar{A} matrix is a weighted sum of symmetric matrices from the symmetric matrix storage E , so \bar{A}_{ij} is a symmetric matrix. By default all elements in \bar{A} are 0, so only non-zero elements need be added. Setting the same element again will overwrite the earlier entry.

The symmetric matrices from E are defined separately using the function *MSK_appendsparsesymmat*.

Parameters

- **task** (*MSKtask_t*) – An optimization task. (input)
- **i** (*MSKint32t*) – Row index of \bar{A} . (input)
- **j** (*MSKint32t*) – Column index of \bar{A} . (input)
- **num** (*MSKint64t*) – The number of terms in the weighted sum that forms \bar{A}_{ij} . (input)
- **sub** (*MSKint64t**) – Indices in E of the matrices appearing in the weighted sum for \bar{A}_{ij} . (input)

- `weights (MSKrealt*)` – `weights[k]` is the coefficient of the `sub[k]`-th element of E in the weighted sum forming \bar{A}_{ij} . (input)

Return (`MSKrescodee`) – The function response code.

Groups *Problem data - semidefinite*

MSK_putbaraijlist

```
MSKrescodee (MSKAPI MSK_putbaraijlist) (
    MSKtask_t task,
    MSKint32t num,
    const MSKint32t * subi,
    const MSKint32t * subj,
    const MSKint64t * alphaptrb,
    const MSKint64t * alphaptre,
    const MSKint64t * matidx,
    const MSKrealt * weights)
```

This function sets a list of elements in the \bar{A} matrix.

Each element in the \bar{A} matrix is a weighted sum of symmetric matrices from the symmetric matrix storage E , so \bar{A}_{ij} is a symmetric matrix. By default all elements in \bar{A} are 0, so only non-zero elements need be added. Setting the same element again will overwrite the earlier entry.

The symmetric matrices from E are defined separately using the function *MSK_appendsparsesymmat*.

Parameters

- `task (MSKtask_t)` – An optimization task. (input)
- `num (MSKint32t)` – Number of \bar{A} entries to add. (input)
- `subi (MSKint32t*)` – Row index of \bar{A} . (input)
- `subj (MSKint32t*)` – Column index of \bar{A} . (input)
- `alphaptrb (MSKint64t*)` – Start entries for terms in the weighted sum that forms \bar{A}_{ij} . (input)
- `alphaptre (MSKint64t*)` – End entries for terms in the weighted sum that forms \bar{A}_{ij} . (input)
- `matidx (MSKint64t*)` – Indices in E of the matrices appearing in the weighted sum for \bar{A}_{ij} . (input)
- `weights (MSKrealt*)` – `weights[k]` is the coefficient of the `sub[k]`-th element of E in the weighted sum forming \bar{A}_{ij} . (input)

Return (`MSKrescodee`) – The function response code.

Groups *Problem data - semidefinite*

MSK_putbararowlist

```
MSKrescodee (MSKAPI MSK_putbararowlist) (
    MSKtask_t task,
    MSKint32t num,
    const MSKint32t * subi,
    const MSKint64t * ptrb,
    const MSKint64t * ptre,
    const MSKint32t * subj,
    const MSKint64t * nummat,
    const MSKint64t * matidx,
    const MSKrealt * weights)
```

This function replaces a list of rows in the \bar{A} matrix.

Parameters

- `task (MSKtask_t)` – An optimization task. (input)

- num (*MSKint32t*) – Number of \bar{A} entries to add. (input)
- subi (*MSKint32t**) – Row indexes of \bar{A} . (input)
- ptrb (*MSKint64t**) – Start of rows in \bar{A} . (input)
- ptre (*MSKint64t**) – End of rows in \bar{A} . (input)
- subj (*MSKint32t**) – Column index of \bar{A} . (input)
- nummat (*MSKint64t**) – Number of entries in weighted sum of matrixes. (input)
- matidx (*MSKint64t**) – Matrix indexes for weighted sum of matrixes. (input)
- weights (*MSKrealt**) – Weights for weighted sum of matrixes. (input)

Return (*MSKrescodee*) – The function response code.

Groups *Problem data - semidefinite*

MSK_putbarcblocktriplet

```
MSKrescodee (MSKAPI MSK_putbarcblocktriplet) (
    MSKtask_t task,
    MSKint64t num,
    const MSKint32t * subj,
    const MSKint32t * subk,
    const MSKint32t * subl,
    const MSKrealt * valjkl)
```

Inputs the \bar{C} matrix in block triplet form.

Parameters

- task (*MSKtask_t*) – An optimization task. (input)
- num (*MSKint64t*) – Number of elements in the block triplet form. (input)
- subj (*MSKint32t**) – Symmetric matrix variable index. (input)
- subk (*MSKint32t**) – Block row index. (input)
- subl (*MSKint32t**) – Block column index. (input)
- valjkl (*MSKrealt**) – The numerical value associated with each block triplet. (input)

Return (*MSKrescodee*) – The function response code.

Groups *Problem data - semidefinite*

MSK_putbarcj

```
MSKrescodee (MSKAPI MSK_putbarcj) (
    MSKtask_t task,
    MSKint32t j,
    MSKint64t num,
    const MSKint64t * sub,
    const MSKrealt * weights)
```

This function sets one entry in the \bar{C} vector.

Each element in the \bar{C} vector is a weighted sum of symmetric matrices from the symmetric matrix storage E , so \bar{C}_j is a symmetric matrix. By default all elements in \bar{C} are 0, so only non-zero elements need be added. Setting the same element again will overwrite the earlier entry.

The symmetric matrices from E are defined separately using the function *MSK_appendsparsesymmat*.

Parameters

- task (*MSKtask_t*) – An optimization task. (input)
- j (*MSKint32t*) – Index of the element in \bar{C} that should be changed. (input)
- num (*MSKint64t*) – The number of elements in the weighted sum that forms \bar{C}_j . (input)

- `sub (MSKint64t*)` – Indices in E of matrices appearing in the weighted sum for \overline{C}_j (input)
- `weights (MSKrealt*)` – `weights[k]` is the coefficient of the `sub[k]`-th element of E in the weighted sum forming \overline{C}_j . (input)

Return (*MSKrescodee*) – The function response code.

Groups *Problem data - semidefinite, Problem data - objective*

MSK_putbarsj

```
MSKrescodee (MSKAPI MSK_putbarsj) (
    MSKtask_t task,
    MSKsoltypee whichsol,
    MSKint32t j,
    const MSKrealt * barsj)
```

Sets the dual solution for a semidefinite variable.

Parameters

- `task (MSKtask_t)` – An optimization task. (input)
- `whichsol (MSKsoltypee)` – Selects a solution. (input)
- `j (MSKint32t)` – Index of the semidefinite variable. (input)
- `barsj (MSKrealt*)` – Value of \overline{S}_j . Format as in *MSK_getbarsj*. (input)

Return (*MSKrescodee*) – The function response code.

Groups *Solution - semidefinite*

MSK_putbarvarname

```
MSKrescodee (MSKAPI MSK_putbarvarname) (
    MSKtask_t task,
    MSKint32t j,
    const char * name)
```

Sets the name of a semidefinite variable.

Parameters

- `task (MSKtask_t)` – An optimization task. (input)
- `j (MSKint32t)` – Index of the variable. (input)
- `name (MSKstring_t)` – The variable name. (input)

Return (*MSKrescodee*) – The function response code.

Groups *Names, Problem data - semidefinite*

MSK_putbarxj

```
MSKrescodee (MSKAPI MSK_putbarxj) (
    MSKtask_t task,
    MSKsoltypee whichsol,
    MSKint32t j,
    const MSKrealt * barxj)
```

Sets the primal solution for a semidefinite variable.

Parameters

- `task (MSKtask_t)` – An optimization task. (input)
- `whichsol (MSKsoltypee)` – Selects a solution. (input)
- `j (MSKint32t)` – Index of the semidefinite variable. (input)
- `barxj (MSKrealt*)` – Value of \overline{X}_j . Format as in *MSK_getbarxj*. (input)

Return (*MSKrescodee*) – The function response code.

Groups *Solution - semidefinite*

MSK_putcallbackfunc

```
MSKrescodee (MSKAPI MSK_putcallbackfunc) (  
    MSKtask_t task,  
    MSKcallbackfunc func,  
    MSKuserhandle_t handle)
```

Sets a user-defined progress callback function of type *MSKcallbackfunc*. The callback function is called frequently during the optimization process. See Section *Progress and data callback* for an example.

Parameters

- **task** (*MSKtask_t*) – An optimization task. (input)
- **func** (*MSKcallbackfunc*) – A user-defined function which will be called occasionally from within the **MOSEK** optimizers. If the argument is a NULL pointer, then a previously defined callback function is removed. The progress function has the type *MSKcallbackfunc*. (input)
- **handle** (*MSKuserhandle_t*) – A pointer to a user-defined data structure. Whenever the function **func** is called, then **handle** is passed to the function. (input)

Return (*MSKrescodee*) – The function response code.

Groups *Callback*

MSK_putcfix

```
MSKrescodee (MSKAPI MSK_putcfix) (  
    MSKtask_t task,  
    MSKrealt cfix)
```

Replaces the fixed term in the objective by a new one.

Parameters

- **task** (*MSKtask_t*) – An optimization task. (input)
- **cfix** (*MSKrealt*) – Fixed term in the objective. (input)

Return (*MSKrescodee*) – The function response code.

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

MSK_putcj

```
MSKrescodee (MSKAPI MSK_putcj) (  
    MSKtask_t task,  
    MSKint32t j,  
    MSKrealt cj)
```

Modifies one coefficient in the linear objective vector c , i.e.

$$c_j = cj.$$

If the absolute value exceeds *MSK_DPAR_DATA_TOL_C_HUGE* an error is generated. If the absolute value exceeds *MSK_DPAR_DATA_TOL_CJ_LARGE*, a warning is generated, but the coefficient is inputted as specified.

Parameters

- **task** (*MSKtask_t*) – An optimization task. (input)
- **j** (*MSKint32t*) – Index of the variable for which c should be changed. (input)
- **cj** (*MSKrealt*) – New value of c_j . (input)

Return (*MSKrescodee*) – The function response code.

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

MSK_putclist

```
MSKrescodee (MSKAPI MSK_putclist) (
    MSKtask_t task,
    MSKint32t num,
    const MSKint32t * subj,
    const MSKrealt * val)
```

Modifies the coefficients in the linear term c in the objective using the principle

$$c_{\text{subj}[t]} = \text{val}[t], \quad t = 0, \dots, \text{num} - 1.$$

If a variable index is specified multiple times in `subj` only the last entry is used. Data checks are performed as in *MSK_putcj*.

Parameters

- `task` (*MSKtask_t*) – An optimization task. (input)
- `num` (*MSKint32t*) – Number of coefficients that should be changed. (input)
- `subj` (*MSKint32t**) – Indices of variables for which the coefficient in c should be changed. (input)
- `val` (*MSKrealt**) – New numerical values for coefficients in c that should be modified. (input)

Return (*MSKrescodee*) – The function response code.

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

MSK_putconbound

```
MSKrescodee (MSKAPI MSK_putconbound) (
    MSKtask_t task,
    MSKint32t i,
    MSKboundkeye bkc,
    MSKrealt blc,
    MSKrealt buc)
```

Changes the bounds for one constraint.

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

Parameters

- `task` (*MSKtask_t*) – An optimization task. (input)
- `i` (*MSKint32t*) – Index of the constraint. (input)
- `bkc` (*MSKboundkeye*) – New bound key. (input)
- `blc` (*MSKrealt*) – New lower bound. (input)
- `buc` (*MSKrealt*) – New upper bound. (input)

Return (*MSKrescodee*) – The function response code.

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

MSK_putconboundlist

```
MSKrescodee (MSKAPI MSK_putconboundlist) (
    MSKtask_t task,
    MSKint32t num,
    const MSKint32t * sub,
    const MSKboundkeye * bkc,
    const MSKrealt * blc,
    const MSKrealt * buc)
```

Changes the bounds for a list of constraints. If multiple bound changes are specified for a constraint, then only the last change takes effect. Data checks are performed as in *MSK_putconbound*.

Parameters

- **task** (*MSKtask_t*) – An optimization task. (input)
- **num** (*MSKint32t*) – Number of bounds that should be changed. (input)
- **sub** (*MSKint32t**) – List of constraint indexes. (input)
- **bkc** (*MSKboundkeye**) – Bound keys for the constraints. (input)
- **blc** (*MSKrealt**) – Lower bounds for the constraints. (input)
- **buc** (*MSKrealt**) – Upper bounds for the constraints. (input)

Return (*MSKrescodee*) – The function response code.

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

MSK_putconboundlistconst

```
MSKrescodee (MSKAPI MSK_putconboundlistconst) (
    MSKtask_t task,
    MSKint32t num,
    const MSKint32t * sub,
    MSKboundkeye bkc,
    MSKrealt blc,
    MSKrealt buc)
```

Changes the bounds for one or more constraints. Data checks are performed as in *MSK_putconbound*.

Parameters

- **task** (*MSKtask_t*) – An optimization task. (input)
- **num** (*MSKint32t*) – Number of bounds that should be changed. (input)
- **sub** (*MSKint32t**) – List of constraint indexes. (input)
- **bkc** (*MSKboundkeye*) – New bound key for all constraints in the list. (input)
- **blc** (*MSKrealt*) – New lower bound for all constraints in the list. (input)
- **buc** (*MSKrealt*) – New upper bound for all constraints in the list. (input)

Return (*MSKrescodee*) – The function response code.

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

MSK_putconboundslice

```
MSKrescodee (MSKAPI MSK_putconboundslice) (
    MSKtask_t task,
    MSKint32t first,
    MSKint32t last,
    const MSKboundkeye * bkc,
    const MSKrealt * blc,
    const MSKrealt * buc)
```

Changes the bounds for a slice of the constraints. Data checks are performed as in *MSK_putconbound*.

Parameters

- `task` (*MSKtask_t*) – An optimization task. (input)
- `first` (*MSKint32t*) – First index in the sequence. (input)
- `last` (*MSKint32t*) – Last index plus 1 in the sequence. (input)
- `bkc` (*MSKboundkeye**) – Bound keys for the constraints. (input)
- `blc` (*MSKrealt**) – Lower bounds for the constraints. (input)
- `buc` (*MSKrealt**) – Upper bounds for the constraints. (input)

Return (*MSKrescodee*) – The function response code.

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

MSK_putconboundsliceconst

```
MSKrescodee (MSKAPI MSK_putconboundsliceconst) (
    MSKtask_t task,
    MSKint32t first,
    MSKint32t last,
    MSKboundkeye bkc,
    MSKrealt blc,
    MSKrealt buc)
```

Changes the bounds for a slice of the constraints. Data checks are performed as in *MSK_putconbound*.

Parameters

- `task` (*MSKtask_t*) – An optimization task. (input)
- `first` (*MSKint32t*) – First index in the sequence. (input)
- `last` (*MSKint32t*) – Last index plus 1 in the sequence. (input)
- `bkc` (*MSKboundkeye*) – New bound key for all constraints in the slice. (input)
- `blc` (*MSKrealt*) – New lower bound for all constraints in the slice. (input)
- `buc` (*MSKrealt*) – New upper bound for all constraints in the slice. (input)

Return (*MSKrescodee*) – The function response code.

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

MSK_putcone

```
MSKrescodee (MSKAPI MSK_putcone) (
    MSKtask_t task,
    MSKint32t k,
    MSKconetypee ct,
    MSKrealt coneapar,
    MSKint32t nummem,
    const MSKint32t * submem)
```

Replaces a conic constraint.

Parameters

- `task` (*MSKtask_t*) – An optimization task. (input)
- `k` (*MSKint32t*) – Index of the cone. (input)
- `ct` (*MSKconetypee*) – Specifies the type of the cone. (input)
- `coneapar` (*MSKrealt*) – For the power cone it denotes the exponent alpha. For other cone types it is unused and can be set to 0. (input)
- `nummem` (*MSKint32t*) – Number of member variables in the cone. (input)
- `submem` (*MSKint32t**) – Variable subscripts of the members in the cone. (input)

Return (*MSKrescodee*) – The function response code.

Groups *Problem data - cones*

MSK_putconename

```
MSKrescodee (MSKAPI MSK_putconename) (
    MSKtask_t task,
    MSKint32t j,
    const char * name)
```

Sets the name of a cone.

Parameters

- **task** (*MSKtask_t*) – An optimization task. (input)
- **j** (*MSKint32t*) – Index of the cone. (input)
- **name** (*MSKstring_t*) – The name of the cone. (input)

Return (*MSKrescodee*) – The function response code.

Groups *Names, Problem data - cones*

MSK_putconname

```
MSKrescodee (MSKAPI MSK_putconname) (
    MSKtask_t task,
    MSKint32t i,
    const char * name)
```

Sets the name of a constraint.

Parameters

- **task** (*MSKtask_t*) – An optimization task. (input)
- **i** (*MSKint32t*) – Index of the constraint. (input)
- **name** (*MSKstring_t*) – The name of the constraint. (input)

Return (*MSKrescodee*) – The function response code.

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

MSK_putconsolutioni

```
MSKrescodee (MSKAPI MSK_putconsolutioni) (
    MSKtask_t task,
    MSKint32t i,
    MSKsoltypee whichsol,
    MSKstakeye sk,
    MSKrealt x,
    MSKrealt sl,
    MSKrealt su)
```

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

Parameters

- **task** (*MSKtask_t*) – An optimization task. (input)
- **i** (*MSKint32t*) – Index of the constraint. (input)
- **whichsol** (*MSKsoltypee*) – Selects a solution. (input)
- **sk** (*MSKstakeye*) – Status key of the constraint. (input)
- **x** (*MSKrealt*) – Primal solution value of the constraint. (input)
- **sl** (*MSKrealt*) – Solution value of the dual variable associated with the lower bound. (input)
- **su** (*MSKrealt*) – Solution value of the dual variable associated with the upper bound. (input)

Return (*MSKrescodee*) – The function response code.

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

MSK_putcslice

```
MSKrescodee (MSKAPI MSK_putcslice) (  
    MSKtask_t task,  
    MSKint32t first,  
    MSKint32t last,  
    const MSKrealt * 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 *MSK_putcj*.

Parameters

- **task** (*MSKtask_t*) – An optimization task. (input)
- **first** (*MSKint32t*) – First element in the slice of c . (input)
- **last** (*MSKint32t*) – Last element plus 1 of the slice in c to be changed. (input)
- **slice** (*MSKrealt**) – New numerical values for coefficients in c that should be modified. (input)

Return (*MSKrescodee*) – The function response code.

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

MSK_putdouparam

```
MSKrescodee (MSKAPI MSK_putdouparam) (  
    MSKtask_t task,  
    MSKdparam param,  
    MSKrealt parvalue)
```

Sets the value of a double parameter.

Parameters

- **task** (*MSKtask_t*) – An optimization task. (input)
- **param** (*MSKdparam*) – Which parameter. (input)
- **parvalue** (*MSKrealt*) – Parameter value. (input)

Return (*MSKrescodee*) – The function response code.

Groups *Parameters*

MSK_putexitfunc

```
MSKrescodee (MSKAPI MSK_putexitfunc) (  
    MSKenv_t env,  
    MSKexitfunc exitfunc,  
    MSKuserhandle_t handle)
```

In case **MOSEK** experiences a fatal error, then a user-defined exit function can be called. The exit function should terminate **MOSEK**. In general it is not necessary to define an exit function.

Parameters

- **env** (*MSKenv_t*) – The MOSEK environment. (input)
- **exitfunc** (*MSKexitfunc*) – A user-defined exit function. (input)
- **handle** (*MSKuserhandle_t*) – A pointer to a user-defined data structure which is passed to **exitfunc** when called. (input)

Return (*MSKrescodee*) – The function response code.

Groups *Logging, Callback*

MSK_putintparam

```
MSKrescodee (MSKAPI MSK_putintparam) (  
    MSKtask_t task,  
    MSKiparam param,  
    MSKint32t parvalue)
```

Sets the value of an integer parameter.

Parameters

- task (*MSKtask_t*) – An optimization task. (input)
- param (*MSKiparam*) – Which parameter. (input)
- parvalue (*MSKint32t*) – Parameter value. (input)

Return (*MSKrescodee*) – The function response code.

Groups *Parameters*

MSK_putlicensecode

```
MSKrescodee (MSKAPI MSK_putlicensecode) (  
    MSKenv_t env,  
    const MSKint32t * code)
```

Input a runtime license code.

Parameters

- env (*MSKenv_t*) – The MOSEK environment. (input)
- code (*MSKint32t**) – A runtime license code. (input)

Return (*MSKrescodee*) – The function response code.

Groups *License system*

MSK_putlicensedebug

```
MSKrescodee (MSKAPI MSK_putlicensedebug) (  
    MSKenv_t env,  
    MSKint32t licdebug)
```

Enables debug information for the license system. If *licdebug* is non-zero, then **MOSEK** will print debug info regarding the license checkout.

Parameters

- env (*MSKenv_t*) – The MOSEK environment. (input)
- licdebug (*MSKint32t*) – Whether license checkout debug info should be printed. (input)

Return (*MSKrescodee*) – The function response code.

Groups *License system*

MSK_putlicensepath

```
MSKrescodee (MSKAPI MSK_putlicensepath) (  
    MSKenv_t env,  
    const char * licensepath)
```

Set the path to the license file.

Parameters

- env (*MSKenv_t*) – The MOSEK environment. (input)

- `licensepath` (*MSKstring_t*) – A path specifying where to search for the license. (input)

Return (*MSKrescodee*) – The function response code.

Groups *License system*

MSK_putlicensewait

```
MSKrescodee (MSKAPI MSK_putlicensewait) (
    MSKenv_t env,
    MSKint32t licwait)
```

Control whether **MOSEK** should wait for an available license if no license is available. If `licwait` is non-zero, then **MOSEK** will wait for `licwait`-1 milliseconds between each check for an available license.

Parameters

- `env` (*MSKenv_t*) – The MOSEK environment. (input)
- `licwait` (*MSKint32t*) – Whether **MOSEK** should wait for a license if no license is available. (input)

Return (*MSKrescodee*) – The function response code.

Groups *License system*

MSK_putmaxnumanz

```
MSKrescodee (MSKAPI MSK_putmaxnumanz) (
    MSKtask_t task,
    MSKint64t 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

- `task` (*MSKtask_t*) – An optimization task. (input)
- `maxnumanz` (*MSKint64t*) – Number of preallocated non-zeros in A . (input)

Return (*MSKrescodee*) – The function response code.

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

MSK_putmaxnumberbarvar

```
MSKrescodee (MSKAPI MSK_putmaxnumberbarvar) (
    MSKtask_t task,
    MSKint32t 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

- **task** (*MSKtask_t*) – An optimization task. (input)
- **maxnumbarvar** (*MSKint32t*) – Number of preallocated symmetric matrix variables. (input)

Return (*MSKrescodee*) – The function response code.

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

MSK_putmaxnumcon

```
MSKrescodee (MSKAPI MSK_putmaxnumcon) (  
    MSKtask_t task,  
    MSKint32t 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

- **task** (*MSKtask_t*) – An optimization task. (input)
- **maxnumcon** (*MSKint32t*) – Number of preallocated constraints in the optimization task. (input)

Return (*MSKrescodee*) – The function response code.

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

MSK_putmaxnumcone

```
MSKrescodee (MSKAPI MSK_putmaxnumcone) (  
    MSKtask_t task,  
    MSKint32t 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

- **task** (*MSKtask_t*) – An optimization task. (input)
- **maxnumcone** (*MSKint32t*) – Number of preallocated conic constraints in the optimization task. (input)

Return (*MSKrescodee*) – The function response code.

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

MSK_putmaxnumqnz

```
MSKrescodee (MSKAPI MSK_putmaxnumqnz) (  
    MSKtask_t task,  
    MSKint64t 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

- **task** (*MSKtask_t*) – An optimization task. (input)
- **maxnumqnz** (*MSKint64t*) – Number of non-zero elements preallocated in quadratic coefficient matrices. (input)

Return (*MSKrescodee*) – The function response code.

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

MSK_putmaxnumvar

```
MSKrescodee (MSKAPI MSK_putmaxnumvar) (
    MSKtask_t task,
    MSKint32t 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

- **task** (*MSKtask_t*) – An optimization task. (input)
- **maxnumvar** (*MSKint32t*) – Number of preallocated variables in the optimization task. (input)

Return (*MSKrescodee*) – The function response code.

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

MSK_putnadoupam

```
MSKrescodee (MSKAPI MSK_putnadoupam) (
    MSKtask_t task,
    const char * paramname,
    MSKrealt parvalue)
```

Sets the value of a named double parameter.

Parameters

- **task** (*MSKtask_t*) – An optimization task. (input)
- **paramname** (*MSKstring_t*) – Name of a parameter. (input)
- **parvalue** (*MSKrealt*) – Parameter value. (input)

Return (*MSKrescodee*) – The function response code.

Groups *Parameters*

MSK_putnaintparam

```
MSKrescodee (MSKAPI MSK_putnaintparam) (
    MSKtask_t task,
    const char * paramname,
    MSKint32t parvalue)
```

Sets the value of a named integer parameter.

Parameters

- task (*MSKtask_t*) – An optimization task. (input)
- paramname (*MSKstring_t*) – Name of a parameter. (input)
- parvalue (*MSKint32t*) – Parameter value. (input)

Return (*MSKrescodee*) – The function response code.

Groups *Parameters*

MSK_putnastrparam

```
MSKrescodee (MSKAPI MSK_putnastrparam) (  
    MSKtask_t task,  
    const char * paramname,  
    const char * parvalue)
```

Sets the value of a named string parameter.

Parameters

- task (*MSKtask_t*) – An optimization task. (input)
- paramname (*MSKstring_t*) – Name of a parameter. (input)
- parvalue (*MSKstring_t*) – Parameter value. (input)

Return (*MSKrescodee*) – The function response code.

Groups *Parameters*

MSK_putobjname

```
MSKrescodee (MSKAPI MSK_putobjname) (  
    MSKtask_t task,  
    const char * objname)
```

Assigns a new name to the objective.

Parameters

- task (*MSKtask_t*) – An optimization task. (input)
- objname (*MSKstring_t*) – Name of the objective. (input)

Return (*MSKrescodee*) – The function response code.

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

MSK_putobjsense

```
MSKrescodee (MSKAPI MSK_putobjsense) (  
    MSKtask_t task,  
    MSKobjsensee sense)
```

Sets the objective sense of the task.

Parameters

- task (*MSKtask_t*) – An optimization task. (input)
- sense (*MSKobjsensee*) – The objective sense of the task. The values *MSK_OBJECTIVE_SENSE_MAXIMIZE* and *MSK_OBJECTIVE_SENSE_MINIMIZE* mean that the problem is maximized or minimized respectively. (input)

Return (*MSKrescodee*) – The function response code.

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

MSK_putparam

```
MSKrescodee (MSKAPI MSK_putparam) (
    MSKtask_t task,
    const char * parname,
    const char * parvalue)
```

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

Parameters

- `task` (*MSKtask_t*) – An optimization task. (input)
- `parname` (*MSKstring_t*) – Parameter name. (input)
- `parvalue` (*MSKstring_t*) – Parameter value. (input)

Return (*MSKrescodee*) – The function response code.

Groups *Parameters*

MSK_putqcon

```
MSKrescodee (MSKAPI MSK_putqcon) (
    MSKtask_t task,
    MSKint32t numqcnz,
    const MSKint32t * qcsbk,
    const MSKint32t * qcsubi,
    const MSKint32t * qcsubj,
    const MSKrealt * 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:

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

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

Please note that:

- For large problems it is essential for the efficiency that the function *MSK_putmaxnumqcnz* 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

- `task` (*MSKtask_t*) – An optimization task. (input)
- `numqcnz` (*MSKint32t*) – Number of quadratic terms. (input)
- `qcsbk` (*MSKint32t**) – Constraint subscripts for quadratic coefficients. (input)
- `qcsubi` (*MSKint32t**) – Row subscripts for quadratic constraint matrix. (input)
- `qcsubj` (*MSKint32t**) – Column subscripts for quadratic constraint matrix. (input)
- `qcval` (*MSKrealt**) – Quadratic constraint coefficient values. (input)

Return (*MSKrescodee*) – The function response code.

MSK_putqconk

```
MSKrescodee (MSKAPI MSK_putqconk) (
    MSKtask_t task,
    MSKint32t k,
    MSKint32t numqcnz,
    const MSKint32t * qcsubi,
    const MSKint32t * qcsubj,
    const MSKrealt * qcval)
```

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

Parameters

- *task* (*MSKtask_t*) – An optimization task. (input)
- *k* (*MSKint32t*) – The constraint in which the new *Q* elements are inserted. (input)
- *numqcnz* (*MSKint32t*) – Number of quadratic terms. (input)
- *qcsubi* (*MSKint32t**) – Row subscripts for quadratic constraint matrix. (input)
- *qcsubj* (*MSKint32t**) – Column subscripts for quadratic constraint matrix. (input)
- *qcval* (*MSKrealt**) – Quadratic constraint coefficient values. (input)

Return (*MSKrescodee*) – The function response code.

Groups *Problem data - quadratic part*

MSK_putqobj

```
MSKrescodee (MSKAPI MSK_putqobj) (
    MSKtask_t task,
    MSKint32t numqonz,
    const MSKint32t * qosubi,
    const MSKint32t * qosubj,
    const MSKrealt * 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 *MSK_putqcon* for important remarks and example.

Parameters

- *task* (*MSKtask_t*) – An optimization task. (input)
- *numqonz* (*MSKint32t*) – Number of non-zero elements in the quadratic objective terms. (input)
- *qosubi* (*MSKint32t**) – Row subscripts for quadratic objective coefficients. (input)
- *qosubj* (*MSKint32t**) – Column subscripts for quadratic objective coefficients. (input)

- `qoval (MSKrealt*)` – Quadratic objective coefficient values. (input)
- Return** (*MSKrescodee*) – The function response code.
- Groups** *Problem data - quadratic part, Problem data - objective*

MSK_putqobjij

```
MSKrescodee (MSKAPI MSK_putqobjij) (
    MSKtask_t task,
    MSKint32t i,
    MSKint32t j,
    MSKrealt 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 *MSK_putqobj* instead whenever possible.

Parameters

- `task (MSKtask_t)` – An optimization task. (input)
- `i (MSKint32t)` – Row index for the coefficient to be replaced. (input)
- `j (MSKint32t)` – Column index for the coefficient to be replaced. (input)
- `qoij (MSKrealt)` – The new value for q_{ij}^o . (input)

Return (*MSKrescodee*) – The function response code.

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

MSK_putresponsefunc

```
MSKrescodee (MSKAPI MSK_putresponsefunc) (
    MSKtask_t task,
    MSKresponsefunc responsefunc,
    MSKuserhandle_t handle)
```

Inputs a user-defined error callback which is called when an error or warning occurs.

Parameters

- `task (MSKtask_t)` – An optimization task. (input)
- `responsefunc (MSKresponsefunc)` – A user-defined response handling function. (input)
- `handle (MSKuserhandle_t)` – A user-defined data structure that is passed to the function `responsefunc` whenever it is called. (input)

Return (*MSKrescodee*) – The function response code.

Groups *Callback, Logging*

MSK_putskc

```
MSKrescodee (MSKAPI MSK_putskc) (
    MSKtask_t task,
    MSKsoltypee whichsol,
    const MSKstakeye * skc)
```

Sets the status keys for the constraints.

Parameters

- `task` (*MSKtask_t*) – An optimization task. (input)
 - `whichsol` (*MSKsoltypee*) – Selects a solution. (input)
 - `skc` (*MSKstakeye**) – Status keys for the constraints. (input)
- Return** (*MSKrescodee*) – The function response code.
- Groups** *Solution information*

MSK_putskcslice

```
MSKrescodee (MSKAPI MSK_putskcslice) (
    MSKtask_t task,
    MSKsoltypee whichsol,
    MSKint32t first,
    MSKint32t last,
    const MSKstakeye * skc)
```

Sets the status keys for a slice of the constraints.

Parameters

- `task` (*MSKtask_t*) – An optimization task. (input)
 - `whichsol` (*MSKsoltypee*) – Selects a solution. (input)
 - `first` (*MSKint32t*) – First index in the sequence. (input)
 - `last` (*MSKint32t*) – Last index plus 1 in the sequence. (input)
 - `skc` (*MSKstakeye**) – Status keys for the constraints. (input)
- Return** (*MSKrescodee*) – The function response code.
- Groups** *Solution information*

MSK_putskx

```
MSKrescodee (MSKAPI MSK_putskx) (
    MSKtask_t task,
    MSKsoltypee whichsol,
    const MSKstakeye * skx)
```

Sets the status keys for the scalar variables.

Parameters

- `task` (*MSKtask_t*) – An optimization task. (input)
 - `whichsol` (*MSKsoltypee*) – Selects a solution. (input)
 - `skx` (*MSKstakeye**) – Status keys for the variables. (input)
- Return** (*MSKrescodee*) – The function response code.
- Groups** *Solution information*

MSK_putskxslice

```
MSKrescodee (MSKAPI MSK_putskxslice) (
    MSKtask_t task,
    MSKsoltypee whichsol,
    MSKint32t first,
    MSKint32t last,
    const MSKstakeye * skx)
```

Sets the status keys for a slice of the variables.

Parameters

- `task` (*MSKtask_t*) – An optimization task. (input)
- `whichsol` (*MSKsoltypee*) – Selects a solution. (input)
- `first` (*MSKint32t*) – First index in the sequence. (input)

- last (*MSKint32t*) – Last index plus 1 in the sequence. (input)
- skx (*MSKstakeye**) – Status keys for the variables. (input)

Return (*MSKrescodee*) – The function response code.

Groups *Solution information*

MSK_putslc

```
MSKrescodee (MSKAPI MSK_putslc) (
    MSKtask_t task,
    MSKsoltypee whichsol,
    const MSKrealt * slc)
```

Sets the s_l^c vector for a solution.

Parameters

- task (*MSKtask_t*) – An optimization task. (input)
- whichsol (*MSKsoltypee*) – Selects a solution. (input)
- slc (*MSKrealt**) – Dual variables corresponding to the lower bounds on the constraints. (input)

Return (*MSKrescodee*) – The function response code.

Groups *Solution - dual*

MSK_putslcslice

```
MSKrescodee (MSKAPI MSK_putslcslice) (
    MSKtask_t task,
    MSKsoltypee whichsol,
    MSKint32t first,
    MSKint32t last,
    const MSKrealt * slc)
```

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

Parameters

- task (*MSKtask_t*) – An optimization task. (input)
- whichsol (*MSKsoltypee*) – Selects a solution. (input)
- first (*MSKint32t*) – First index in the sequence. (input)
- last (*MSKint32t*) – Last index plus 1 in the sequence. (input)
- slc (*MSKrealt**) – Dual variables corresponding to the lower bounds on the constraints. (input)

Return (*MSKrescodee*) – The function response code.

Groups *Solution - dual*

MSK_putslx

```
MSKrescodee (MSKAPI MSK_putslx) (
    MSKtask_t task,
    MSKsoltypee whichsol,
    const MSKrealt * slx)
```

Sets the s_l^x vector for a solution.

Parameters

- task (*MSKtask_t*) – An optimization task. (input)
- whichsol (*MSKsoltypee*) – Selects a solution. (input)
- slx (*MSKrealt**) – Dual variables corresponding to the lower bounds on the variables. (input)

Return (*MSKrescodee*) – The function response code.

Groups *Solution - dual*

MSK_putslxslice

```
MSKrescodee (MSKAPI MSK_putslxslice) (  
    MSKtask_t task,  
    MSKsoltypee whichsol,  
    MSKint32t first,  
    MSKint32t last,  
    const MSKrealt * slx)
```

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

Parameters

- **task** (*MSKtask_t*) – An optimization task. (input)
- **whichsol** (*MSKsoltypee*) – Selects a solution. (input)
- **first** (*MSKint32t*) – First index in the sequence. (input)
- **last** (*MSKint32t*) – Last index plus 1 in the sequence. (input)
- **slx** (*MSKrealt**) – Dual variables corresponding to the lower bounds on the variables. (input)

Return (*MSKrescodee*) – The function response code.

Groups *Solution - dual*

MSK_putsnx

```
MSKrescodee (MSKAPI MSK_putsnx) (  
    MSKtask_t task,  
    MSKsoltypee whichsol,  
    const MSKrealt * sux)
```

Sets the s_n^x vector for a solution.

Parameters

- **task** (*MSKtask_t*) – An optimization task. (input)
- **whichsol** (*MSKsoltypee*) – Selects a solution. (input)
- **sux** (*MSKrealt**) – Dual variables corresponding to the upper bounds on the variables. (input)

Return (*MSKrescodee*) – The function response code.

Groups *Solution - dual*

MSK_putsnxslice

```
MSKrescodee (MSKAPI MSK_putsnxslice) (  
    MSKtask_t task,  
    MSKsoltypee whichsol,  
    MSKint32t first,  
    MSKint32t last,  
    const MSKrealt * snx)
```

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

Parameters

- **task** (*MSKtask_t*) – An optimization task. (input)
- **whichsol** (*MSKsoltypee*) – Selects a solution. (input)
- **first** (*MSKint32t*) – First index in the sequence. (input)
- **last** (*MSKint32t*) – Last index plus 1 in the sequence. (input)

- **snx** (*MSKrealt**) – Dual variables corresponding to the conic constraints on the variables. (input)

Return (*MSKrescodee*) – The function response code.

Groups *Solution - dual*

MSK_putsolution

```
MSKrescodee (MSKAPI MSK_putsolution) (
    MSKtask_t task,
    MSKsoltypee whichsol,
    const MSKstakeye * skc,
    const MSKstakeye * skx,
    const MSKstakeye * skn,
    const MSKrealt * xc,
    const MSKrealt * xx,
    const MSKrealt * y,
    const MSKrealt * slc,
    const MSKrealt * suc,
    const MSKrealt * slx,
    const MSKrealt * sux,
    const MSKrealt * snx)
```

Inserts a solution into the task.

Parameters

- **task** (*MSKtask_t*) – An optimization task. (input)
- **whichsol** (*MSKsoltypee*) – Selects a solution. (input)
- **skc** (*MSKstakeye**) – Status keys for the constraints. (input)
- **skx** (*MSKstakeye**) – Status keys for the variables. (input)
- **skn** (*MSKstakeye**) – Status keys for the conic constraints. (input)
- **xc** (*MSKrealt**) – Primal constraint solution. (input)
- **xx** (*MSKrealt**) – Primal variable solution. (input)
- **y** (*MSKrealt**) – Vector of dual variables corresponding to the constraints. (input)
- **slc** (*MSKrealt**) – Dual variables corresponding to the lower bounds on the constraints. (input)
- **suc** (*MSKrealt**) – Dual variables corresponding to the upper bounds on the constraints. (input)
- **slx** (*MSKrealt**) – Dual variables corresponding to the lower bounds on the variables. (input)
- **sux** (*MSKrealt**) – Dual variables corresponding to the upper bounds on the variables. (input)
- **snx** (*MSKrealt**) – Dual variables corresponding to the conic constraints on the variables. (input)

Return (*MSKrescodee*) – The function response code.

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

MSK_putsolutionyi

```
MSKrescodee (MSKAPI MSK_putsolutionyi) (
    MSKtask_t task,
    MSKint32t i,
    MSKsoltypee whichsol,
    MSKrealt y)
```

Inputs the dual variable of a solution.

Parameters

- **task** (*MSKtask_t*) – An optimization task. (input)
- **i** (*MSKint32t*) – Index of the dual variable. (input)
- **whichsol** (*MSKsoltypee*) – Selects a solution. (input)
- **y** (*MSKrealt*) – Solution value of the dual variable. (input)

Return (*MSKrescodee*) – The function response code.

Groups *Solution information, Solution - dual*

MSK_putstrparam

```
MSKrescodee (MSKAPI MSK_putstrparam) (
    MSKtask_t task,
    MSKsparame param,
    const char * parvalue)
```

Sets the value of a string parameter.

Parameters

- **task** (*MSKtask_t*) – An optimization task. (input)
- **param** (*MSKsparame*) – Which parameter. (input)
- **parvalue** (*MSKstring_t*) – Parameter value. (input)

Return (*MSKrescodee*) – The function response code.

Groups *Parameters*

MSK_putsuc

```
MSKrescodee (MSKAPI MSK_putsuc) (
    MSKtask_t task,
    MSKsoltypee whichsol,
    const MSKrealt * suc)
```

Sets the s_u^c vector for a solution.

Parameters

- **task** (*MSKtask_t*) – An optimization task. (input)
- **whichsol** (*MSKsoltypee*) – Selects a solution. (input)
- **suc** (*MSKrealt**) – Dual variables corresponding to the upper bounds on the constraints. (input)

Return (*MSKrescodee*) – The function response code.

Groups *Solution - dual*

MSK_putsucslice

```
MSKrescodee (MSKAPI MSK_putsucslice) (
    MSKtask_t task,
    MSKsoltypee whichsol,
    MSKint32t first,
    MSKint32t last,
    const MSKrealt * suc)
```

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

Parameters

- **task** (*MSKtask_t*) – An optimization task. (input)
- **whichsol** (*MSKsoltypee*) – Selects a solution. (input)
- **first** (*MSKint32t*) – First index in the sequence. (input)
- **last** (*MSKint32t*) – Last index plus 1 in the sequence. (input)

- `sux` (*MSKrealt**) – Dual variables corresponding to the upper bounds on the constraints. (input)

Return (*MSKrescodee*) – The function response code.

Groups *Solution - dual*

MSK_putsux

```
MSKrescodee (MSKAPI MSK_putsux) (
    MSKtask_t task,
    MSKsoltypee whichsol,
    const MSKrealt * sux)
```

Sets the s_u^x vector for a solution.

Parameters

- `task` (*MSKtask_t*) – An optimization task. (input)
- `whichsol` (*MSKsoltypee*) – Selects a solution. (input)
- `sux` (*MSKrealt**) – Dual variables corresponding to the upper bounds on the variables. (input)

Return (*MSKrescodee*) – The function response code.

Groups *Solution - dual*

MSK_putsuxslice

```
MSKrescodee (MSKAPI MSK_putsuxslice) (
    MSKtask_t task,
    MSKsoltypee whichsol,
    MSKint32t first,
    MSKint32t last,
    const MSKrealt * sux)
```

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

Parameters

- `task` (*MSKtask_t*) – An optimization task. (input)
- `whichsol` (*MSKsoltypee*) – Selects a solution. (input)
- `first` (*MSKint32t*) – First index in the sequence. (input)
- `last` (*MSKint32t*) – Last index plus 1 in the sequence. (input)
- `sux` (*MSKrealt**) – Dual variables corresponding to the upper bounds on the variables. (input)

Return (*MSKrescodee*) – The function response code.

Groups *Solution - dual*

MSK_puttaskname

```
MSKrescodee (MSKAPI MSK_puttaskname) (
    MSKtask_t task,
    const char * taskname)
```

Assigns a new name to the task.

Parameters

- `task` (*MSKtask_t*) – An optimization task. (input)
- `taskname` (*MSKstring_t*) – Name assigned to the task. (input)

Return (*MSKrescodee*) – The function response code.

Groups *Names, Environment and task management*

MSK_putvarbound

```
MSKrescodee (MSKAPI MSK_putvarbound) (  
    MSKtask_t task,  
    MSKint32t j,  
    MSKboundkeye bxx,  
    MSKrealt blx,  
    MSKrealt bux)
```

Changes the bounds for one variable.

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

Parameters

- task (*MSKtask_t*) – An optimization task. (input)
- j (*MSKint32t*) – Index of the variable. (input)
- bxx (*MSKboundkeye*) – New bound key. (input)
- blx (*MSKrealt*) – New lower bound. (input)
- bux (*MSKrealt*) – New upper bound. (input)

Return (*MSKrescodee*) – The function response code.

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

MSK_putvarboundlist

```
MSKrescodee (MSKAPI MSK_putvarboundlist) (  
    MSKtask_t task,  
    MSKint32t num,  
    const MSKint32t * sub,  
    const MSKboundkeye * bxx,  
    const MSKrealt * blx,  
    const MSKrealt * 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 *MSK_putvarbound*.

Parameters

- task (*MSKtask_t*) – An optimization task. (input)
- num (*MSKint32t*) – Number of bounds that should be changed. (input)
- sub (*MSKint32t**) – List of variable indexes. (input)
- bxx (*MSKboundkeye**) – Bound keys for the variables. (input)
- blx (*MSKrealt**) – Lower bounds for the variables. (input)
- bux (*MSKrealt**) – Upper bounds for the variables. (input)

Return (*MSKrescodee*) – The function response code.

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

MSK_putvarboundlistconst

```
MSKrescodee (MSKAPI MSK_putvarboundlistconst) (  
    MSKtask_t task,  
    MSKint32t num,  
    const MSKint32t * sub,  
    MSKboundkeye bxx,  
    MSKrealt blx,  
    MSKrealt bux)
```

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

Parameters

- `task` (*MSKtask_t*) – An optimization task. (input)
- `num` (*MSKint32t*) – Number of bounds that should be changed. (input)
- `sub` (*MSKint32t**) – List of variable indexes. (input)
- `bkx` (*MSKboundkeye*) – New bound key for all variables in the list. (input)
- `blx` (*MSKrealt*) – New lower bound for all variables in the list. (input)
- `bux` (*MSKrealt*) – New upper bound for all variables in the list. (input)

Return (*MSKrescodee*) – The function response code.

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

MSK_putvarboundslice

```
MSKrescodee (MSKAPI MSK_putvarboundslice) (  
    MSKtask_t task,  
    MSKint32t first,  
    MSKint32t last,  
    const MSKboundkeye * bkx,  
    const MSKrealt * blx,  
    const MSKrealt * bux)
```

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

Parameters

- `task` (*MSKtask_t*) – An optimization task. (input)
- `first` (*MSKint32t*) – First index in the sequence. (input)
- `last` (*MSKint32t*) – Last index plus 1 in the sequence. (input)
- `bkx` (*MSKboundkeye**) – Bound keys for the variables. (input)
- `blx` (*MSKrealt**) – Lower bounds for the variables. (input)
- `bux` (*MSKrealt**) – Upper bounds for the variables. (input)

Return (*MSKrescodee*) – The function response code.

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

MSK_putvarboundsliceconst

```
MSKrescodee (MSKAPI MSK_putvarboundsliceconst) (  
    MSKtask_t task,  
    MSKint32t first,  
    MSKint32t last,  
    MSKboundkeye bkx,  
    MSKrealt blx,  
    MSKrealt bux)
```

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

Parameters

- `task` (*MSKtask_t*) – An optimization task. (input)
- `first` (*MSKint32t*) – First index in the sequence. (input)
- `last` (*MSKint32t*) – Last index plus 1 in the sequence. (input)
- `bkx` (*MSKboundkeye*) – New bound key for all variables in the slice. (input)
- `blx` (*MSKrealt*) – New lower bound for all variables in the slice. (input)
- `bux` (*MSKrealt*) – New upper bound for all variables in the slice. (input)

Return (*MSKrescodee*) – The function response code.

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

MSK_putvarname

```
MSKrescodee (MSKAPI MSK_putvarname) (  
    MSKtask_t task,  
    MSKint32t j,  
    const char * name)
```

Sets the name of a variable.

Parameters

- **task** (*MSKtask_t*) – An optimization task. (input)
- **j** (*MSKint32t*) – Index of the variable. (input)
- **name** (*MSKstring_t*) – The variable name. (input)

Return (*MSKrescodee*) – The function response code.

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

MSK_putvarsolutionj

```
MSKrescodee (MSKAPI MSK_putvarsolutionj) (  
    MSKtask_t task,  
    MSKint32t j,  
    MSKsoltypee whichsol,  
    MSKstakeye sk,  
    MSKrealt x,  
    MSKrealt sl,  
    MSKrealt su,  
    MSKrealt sn)
```

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

Parameters

- **task** (*MSKtask_t*) – An optimization task. (input)
- **j** (*MSKint32t*) – Index of the variable. (input)
- **whichsol** (*MSKsoltypee*) – Selects a solution. (input)
- **sk** (*MSKstakeye*) – Status key of the variable. (input)
- **x** (*MSKrealt*) – Primal solution value of the variable. (input)
- **sl** (*MSKrealt*) – Solution value of the dual variable associated with the lower bound. (input)
- **su** (*MSKrealt*) – Solution value of the dual variable associated with the upper bound. (input)
- **sn** (*MSKrealt*) – Solution value of the dual variable associated with the conic constraint. (input)

Return (*MSKrescodee*) – The function response code.

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

MSK_putvartype

```
MSKrescodee (MSKAPI MSK_putvartype) (  
    MSKtask_t task,  
    MSKint32t j,  
    MSKvariabletypee vartype)
```

Sets the variable type of one variable.

Parameters

- **task** (*MSKtask_t*) – An optimization task. (input)
- **j** (*MSKint32t*) – Index of the variable. (input)

- **vartype** (*MSKvariabletypee*) – The new variable type. (input)
- Return** (*MSKrescodee*) – The function response code.
- Groups** *Problem data - variables*

MSK_putvartypelist

```
MSKrescodee (MSKAPI MSK_putvartypelist) (
    MSKtask_t task,
    MSKint32t num,
    const MSKint32t * subj,
    const MSKvariabletypee * 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

- **task** (*MSKtask_t*) – An optimization task. (input)
- **num** (*MSKint32t*) – Number of variables for which the variable type should be set. (input)
- **subj** (*MSKint32t**) – A list of variable indexes for which the variable type should be changed. (input)
- **vartype** (*MSKvariabletypee**) – A list of variable types that should be assigned to the variables specified by subj. (input)

Return (*MSKrescodee*) – The function response code.

Groups *Problem data - variables*

MSK_putxc

```
MSKrescodee (MSKAPI MSK_putxc) (
    MSKtask_t task,
    MSKsoltypee whichsol,
    MSKrealt * xc)
```

Sets the x^c vector for a solution.

Parameters

- **task** (*MSKtask_t*) – An optimization task. (input)
- **whichsol** (*MSKsoltypee*) – Selects a solution. (input)
- **xc** (*MSKrealt**) – Primal constraint solution. (output)

Return (*MSKrescodee*) – The function response code.

Groups *Solution - primal*

MSK_putxcslice

```
MSKrescodee (MSKAPI MSK_putxcslice) (
    MSKtask_t task,
    MSKsoltypee whichsol,
    MSKint32t first,
    MSKint32t last,
    const MSKrealt * xc)
```

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

Parameters

- **task** (*MSKtask_t*) – An optimization task. (input)
- **whichsol** (*MSKsoltypee*) – Selects a solution. (input)
- **first** (*MSKint32t*) – First index in the sequence. (input)

- last (*MSKint32t*) – Last index plus 1 in the sequence. (input)
 - xc (*MSKrealt**) – Primal constraint solution. (input)
- Return** (*MSKrescodee*) – The function response code.
- Groups** *Solution - primal*

MSK_putxx

```
MSKrescodee (MSKAPI MSK_putxx) (
    MSKtask_t task,
    MSKsoltypee whichsol,
    const MSKrealt * xx)
```

Sets the x^x vector for a solution.

Parameters

- task (*MSKtask_t*) – An optimization task. (input)
 - whichsol (*MSKsoltypee*) – Selects a solution. (input)
 - xx (*MSKrealt**) – Primal variable solution. (input)
- Return** (*MSKrescodee*) – The function response code.
- Groups** *Solution - primal*

MSK_putxxslice

```
MSKrescodee (MSKAPI MSK_putxxslice) (
    MSKtask_t task,
    MSKsoltypee whichsol,
    MSKint32t first,
    MSKint32t last,
    const MSKrealt * xx)
```

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

Parameters

- task (*MSKtask_t*) – An optimization task. (input)
 - whichsol (*MSKsoltypee*) – Selects a solution. (input)
 - first (*MSKint32t*) – First index in the sequence. (input)
 - last (*MSKint32t*) – Last index plus 1 in the sequence. (input)
 - xx (*MSKrealt**) – Primal variable solution. (input)
- Return** (*MSKrescodee*) – The function response code.
- Groups** *Solution - primal*

MSK_puty

```
MSKrescodee (MSKAPI MSK_puty) (
    MSKtask_t task,
    MSKsoltypee whichsol,
    const MSKrealt * y)
```

Sets the y vector for a solution.

Parameters

- task (*MSKtask_t*) – An optimization task. (input)
 - whichsol (*MSKsoltypee*) – Selects a solution. (input)
 - y (*MSKrealt**) – Vector of dual variables corresponding to the constraints. (input)
- Return** (*MSKrescodee*) – The function response code.

Groups *Solution - primal*

MSK_putyslice

```
MSKrescodee (MSKAPI MSK_putyslice) (  
    MSKtask_t task,  
    MSKsoltypee whichsol,  
    MSKint32t first,  
    MSKint32t last,  
    const MSKrealt * y)
```

Sets a slice of the y vector for a solution.

Parameters

- **task** (*MSKtask_t*) – An optimization task. (input)
- **whichsol** (*MSKsoltypee*) – Selects a solution. (input)
- **first** (*MSKint32t*) – First index in the sequence. (input)
- **last** (*MSKint32t*) – Last index plus 1 in the sequence. (input)
- **y** (*MSKrealt**) – Vector of dual variables corresponding to the constraints. (input)

Return (*MSKrescodee*) – The function response code.

Groups *Solution - dual*

MSK_readdata

```
MSKrescodee (MSKAPI MSK_readdata) (  
    MSKtask_t task,  
    const char * filename)
```

Reads an optimization problem and associated data from a file. The extension of the file name is used to deduce the file format.

For a list of supported file types and their extensions see *Supported File Formats*.

Data is read from the file **filename** if it is a nonempty string. Otherwise data is read from the file specified by *MSK_SPAR_DATA_FILE_NAME*.

Parameters

- **task** (*MSKtask_t*) – An optimization task. (input)
- **filename** (*MSKstring_t*) – A valid file name. (input)

Return (*MSKrescodee*) – The function response code.

Groups *Input/Output*

MSK_readdataautoformat

```
MSKrescodee (MSKAPI MSK_readdataautoformat) (  
    MSKtask_t task,  
    const char * filename)
```

Reads an optimization problem and associated data from a file.

Parameters

- **task** (*MSKtask_t*) – An optimization task. (input)
- **filename** (*MSKstring_t*) – A valid file name. (input)

Return (*MSKrescodee*) – The function response code.

Groups *Input/Output*

MSK_readdataformat

```
MSKrescodee (MSKAPI MSK_readdataformat) (
    MSKtask_t task,
    const char * filename,
    int format,
    int compress)
```

Reads an optimization problem and associated data from a file.

Parameters

- **task** (*MSKtask_t*) – An optimization task. (input)
- **filename** (*MSKstring_t*) – A valid file name. (input)
- **format** (*MSKdataformat_e*) – File data format. (input)
- **compress** (*MSKcompress_type_e*) – File compression type. (input)

Return (*MSKrescodee*) – The function response code.

Groups *Input/Output*

MSK_readjsonstring

```
MSKrescodee (MSKAPI MSK_readjsonstring) (
    MSKtask_t task,
    const char * 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

- **task** (*MSKtask_t*) – An optimization task. (input)
- **data** (*MSKstring_t*) – Problem data in text format. (input)

Return (*MSKrescodee*) – The function response code.

Groups *Input/Output*

MSK_readlpstring

```
MSKrescodee (MSKAPI MSK_readlpstring) (
    MSKtask_t task,
    const char * data)
```

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

Parameters

- **task** (*MSKtask_t*) – An optimization task. (input)
- **data** (*MSKstring_t*) – Problem data in text format. (input)

Return (*MSKrescodee*) – The function response code.

Groups *Input/Output*

MSK_readopfstring

```
MSKrescodee (MSKAPI MSK_readopfstring) (
    MSKtask_t task,
    const char * data)
```

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

Parameters

- **task** (*MSKtask_t*) – An optimization task. (input)
 - **data** (*MSKstring_t*) – Problem data in text format. (input)
- Return** (*MSKrescodee*) – The function response code.
- Groups** *Input/Output*

MSK_readparamfile

```
MSKrescodee (MSKAPI MSK_readparamfile) (
    MSKtask_t task,
    const char * 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 *MSK_SPAR_PARAM_READ_FILE_NAME*.

Parameters

- **task** (*MSKtask_t*) – An optimization task. (input)
 - **filename** (*MSKstring_t*) – A valid file name. (input)
- Return** (*MSKrescodee*) – The function response code.
- Groups** *Input/Output*

MSK_readptfstring

```
MSKrescodee (MSKAPI MSK_readptfstring) (
    MSKtask_t task,
    const char * 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

- **task** (*MSKtask_t*) – An optimization task. (input)
 - **data** (*MSKstring_t*) – Problem data in text format. (input)
- Return** (*MSKrescodee*) – The function response code.
- Groups** *Input/Output*

MSK_readsolution

```
MSKrescodee (MSKAPI MSK_readsolution) (
    MSKtask_t task,
    MSKsoltypee whichsol,
    const char * 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 *MSK_SPAR_BAS_SOL_FILE_NAME*, *MSK_SPAR_ITR_SOL_FILE_NAME* or *MSK_SPAR_INT_SOL_FILE_NAME* depending on which solution is chosen.

Parameters

- **task** (*MSKtask_t*) – An optimization task. (input)
 - **whichsol** (*MSKsoltypee*) – Selects a solution. (input)
 - **filename** (*MSKstring_t*) – A valid file name. (input)
- Return** (*MSKrescodee*) – The function response code.
- Groups** *Input/Output*

MSK_readsummary

```
MSKrescodee (MSKAPI MSK_readsummary) (
    MSKtask_t task,
    MSKstreamtypee whichstream)
```

Prints a short summary of last file that was read.

Parameters

- **task** (*MSKtask_t*) – An optimization task. (input)
- **whichstream** (*MSKstreamtypee*) – Index of the stream. (input)

Return (*MSKrescodee*) – The function response code.

Groups *Input/Output, Inspecting the task*

MSK_readtask

```
MSKrescodee (MSKAPI MSK_readtask) (
    MSKtask_t task,
    const char * 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

- **task** (*MSKtask_t*) – An optimization task. (input)
- **filename** (*MSKstring_t*) – A valid file name. (input)

Return (*MSKrescodee*) – The function response code.

Groups *Input/Output*

MSK_removebarvars

```
MSKrescodee (MSKAPI MSK_removebarvars) (
    MSKtask_t task,
    MSKint32t num,
    const MSKint32t * subset)
```

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

Parameters

- **task** (*MSKtask_t*) – An optimization task. (input)
- **num** (*MSKint32t*) – Number of symmetric matrices which should be removed. (input)
- **subset** (*MSKint32t**) – Indexes of symmetric matrices which should be removed. (input)

Return (*MSKrescodee*) – The function response code.

Groups *Problem data - semidefinite*

MSK_removecones

```
MSKrescodee (MSKAPI MSK_removecones) (
    MSKtask_t task,
    MSKint32t num,
    const MSKint32t * 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

- `task` (*MSKtask_t*) – An optimization task. (input)
- `num` (*MSKint32t*) – Number of cones which should be removed. (input)
- `subset` (*MSKint32t**) – Indexes of cones which should be removed. (input)

Return (*MSKrescodee*) – The function response code.

Groups *Problem data - cones*

MSK_removecons

```
MSKrescodee (MSKAPI MSK_removecons) (
    MSKtask_t task,
    MSKint32t num,
    const MSKint32t * subset)
```

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

Parameters

- `task` (*MSKtask_t*) – An optimization task. (input)
- `num` (*MSKint32t*) – Number of constraints which should be removed. (input)
- `subset` (*MSKint32t**) – Indexes of constraints which should be removed. (input)

Return (*MSKrescodee*) – The function response code.

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

MSK_removevars

```
MSKrescodee (MSKAPI MSK_removevars) (
    MSKtask_t task,
    MSKint32t num,
    const MSKint32t * subset)
```

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

Parameters

- `task` (*MSKtask_t*) – An optimization task. (input)
- `num` (*MSKint32t*) – Number of variables which should be removed. (input)
- `subset` (*MSKint32t**) – Indexes of variables which should be removed. (input)

Return (*MSKrescodee*) – The function response code.

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

MSK_resizetask

```
MSKrescodee (MSKAPI MSK_resizetask) (
    MSKtask_t task,
    MSKint32t maxnumcon,
    MSKint32t maxnumvar,
    MSKint32t maxnumcone,
    MSKint64t maxnumanz,
    MSKint64t 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

- **task** (*MSKtask_t*) – An optimization task. (input)
- **maxnumcon** (*MSKint32t*) – New maximum number of constraints. (input)
- **maxnumvar** (*MSKint32t*) – New maximum number of variables. (input)
- **maxnumcone** (*MSKint32t*) – New maximum number of cones. (input)
- **maxnumanz** (*MSKint64t*) – New maximum number of non-zeros in A . (input)
- **maxnumqnz** (*MSKint64t*) – New maximum number of non-zeros in all Q matrices. (input)

Return (*MSKrescodee*) – The function response code.

Groups *Environment and task management*

MSK_sensitivityreport

```
MSKrescodee (MSKAPI MSK_sensitivityreport) (
    MSKtask_t task,
    MSKstreamtypee whichstream)
```

Reads a sensitivity format file from a location given by *MSK_SPAR_SENSITIVITY_FILE_NAME* and writes the result to the stream *whichstream*. If *MSK_SPAR_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

- **task** (*MSKtask_t*) – An optimization task. (input)
- **whichstream** (*MSKstreamtypee*) – Index of the stream. (input)

Return (*MSKrescodee*) – The function response code.

Groups *Sensitivity analysis*

MSK_setdefaults

```
MSKrescodee (MSKAPI MSK_setdefaults) (
    MSKtask_t task)
```

Resets all the parameters to their default values.

Parameters **task** (*MSKtask_t*) – An optimization task. (input)

Return (*MSKrescodee*) – The function response code.

Groups *Parameters*

MSK_setupthreads

```
MSKrescodee (MSKAPI MSK_setupthreads) (
    MSKenv_t env,
    MSKint32t numthreads)
```

Preallocates a thread pool for the interior-point and conic optimizers in the current process. This function should only be called once per process, before first optimization. Future settings of the parameter *MSK_IPAR_NUM_THREADS* will be irrelevant for the conic optimizer.

Parameters

- **env** (*MSKenv_t*) – The MOSEK environment. (input)

- numthreads (*MSKint32t*) – Number of threads. (input)
- Return** (*MSKrescodee*) – The function response code.
- Groups** *System, memory and debugging*

MSK_sktostr

```
MSKrescodee (MSKAPI MSK_sktostr) (
    MSKtask_t task,
    MSKstakeye sk,
    char * str)
```

Obtains a status key abbreviation string, one of UN, BS, SB, LL, UL, EQ, **.

Parameters

- task (*MSKtask_t*) – An optimization task. (input)
- sk (*MSKstakeye*) – A valid status key. (input)
- str (*MSKstring_t*) – Abbreviation string corresponding to the status key sk. (output)

Return (*MSKrescodee*) – The function response code.

Groups *Names*

MSK_solstatostr

```
MSKrescodee (MSKAPI MSK_solstatostr) (
    MSKtask_t task,
    MSKsolstae solsta,
    char * str)
```

Obtains an explanatory string corresponding to a solution status.

Parameters

- task (*MSKtask_t*) – An optimization task. (input)
- solsta (*MSKsolstae*) – Solution status. (input)
- str (*MSKstring_t*) – String corresponding to the solution status solsta. (output)

Return (*MSKrescodee*) – The function response code.

Groups *Names*

MSK_solutiondef

```
MSKrescodee (MSKAPI MSK_solutiondef) (
    MSKtask_t task,
    MSKsoltypee whichsol,
    MSKboolean * isdef)
```

Checks whether a solution is defined.

Parameters

- task (*MSKtask_t*) – An optimization task. (input)
- whichsol (*MSKsoltypee*) – Selects a solution. (input)
- isdef (*MSKboolean by reference*) – Is non-zero if the requested solution is defined. (output)

Return (*MSKrescodee*) – The function response code.

Groups *Solution information*

MSK_solutionsummary

```
MSKrescodee (MSKAPI MSK_solutionssummary) (
    MSKtask_t task,
    MSKstreamtypee whichstream)
```

Prints a short summary of the current solutions.

Parameters

- **task** (*MSKtask_t*) – An optimization task. (input)
- **whichstream** (*MSKstreamtypee*) – Index of the stream. (input)

Return (*MSKrescodee*) – The function response code.

Groups *Logging, Solution information*

MSK_solvewithbasis

```
MSKrescodee (MSKAPI MSK_solvewithbasis) (
    MSKtask_t task,
    MSKint32t transp,
    MSKint32t * numnz,
    MSKint32t * sub,
    MSKrealt * 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

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

or the system

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

for the unknowns \bar{X} , with b being a user-defined vector. In order to make sense of the solution \bar{X} it is important to know the ordering of the variables in the basis because the ordering specifies how B is constructed. When calling *MSK_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 *MSK_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

- **task** (*MSKtask_t*) – An optimization task. (input)
- **transp** (*MSKint32t*) – If this argument is zero, then (15.3) is solved, if non-zero then (15.4) is solved. (input)
- **numnz** (*MSKint32t by reference*) – As input it is the number of non-zeros in b . As output it is the number of non-zeros in \bar{X} . (input/output)
- **sub** (*MSKint32t**) – As input it contains the positions of non-zeros in b . As output it contains the positions of the non-zeros in \bar{X} . It must have room for *numcon* elements. (input/output)
- **val** (*MSKrealt**) – 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 \bar{X} as a dense vector. It must have length *numcon*. (input/output)

Return (*MSKrescodee*) – The function response code.

Groups *Solving systems with basis matrix*

MSK_sparsetriangularsolvedense

```
MSKrescodee (MSKAPI MSK_sparsetriangularsolvedense) (  
    MSKenv_t env,  
    MSKtransposee transposed,  
    MSKint32t n,  
    const MSKint32t * lnzc,  
    const MSKint64t * lptrc,  
    MSKint64t lensubnval,  
    const MSKint32t * lsubc,  
    const MSKrealt * lvalc,  
    MSKrealt * b)
```

The function solves a triangular system of the form

$$Lx = b$$

or

$$L^T x = b$$

where L is a sparse lower triangular nonsingular matrix. This implies in particular that diagonals in L are nonzero.

Parameters

- **env** (*MSKenv_t*) – The MOSEK environment. (input)
- **transposed** (*MSKtransposee*) – Controls whether to use with L or L^T . (input)
- **n** (*MSKint32t*) – Dimension of L . (input)
- **lnzc** (*MSKint32t**) – `lnzc[j]` is the number of nonzeros in column j . (input)
- **lptrc** (*MSKint64t**) – `lptrc[j]` is a pointer to the first row index and value in column j . (input)
- **lensubnval** (*MSKint64t*) – Number of elements in `lsubc` and `lvalc`. (input)
- **lsubc** (*MSKint32t**) – Row indexes for each column stored sequentially. Must be stored in increasing order for each column. (input)
- **lvalc** (*MSKrealt**) – The value corresponding to the row index stored in `lsubc`. (input)
- **b** (*MSKrealt**) – The right-hand side of linear equation system to be solved as a dense vector. (input/output)

Return (*MSKrescodee*) – The function response code.

Groups *Linear algebra*

MSK_strdupbgtask

```
char * (MSKAPI MSK_strdupbgtask) (  
    MSKtask_t task,  
    const char * str,  
    const char * file,  
    const unsigned line)
```

Make a copy of a string. The string created by this procedure must be freed by *MSK_freetask*.

Parameters

- **task** (*MSKtask_t*) – An optimization task. (input)
- **str** (*MSKstring_t*) – String that should be copied. (input)
- **file** (*MSKstring_t*) – File from which the function is called. (input)

- **line (unsigned)** – Line in the file from which the function is called. (input)
- Return** (*MSKstring_t*) – A copy of the given string.
- Groups** *System, memory and debugging*

MSK_strduptask

```
char * (MSKAPI MSK_strduptask) (
    MSKtask_t task,
    const char * str)
```

Make a copy of a string. The string created by this procedure must be freed by *MSK_freetask*.

Parameters

- **task** (*MSKtask_t*) – An optimization task. (input)
- **str** (*MSKstring_t*) – String that should be copied. (input)

Return (*MSKstring_t*) – A copy of the given string.

Groups *System, memory and debugging*

MSK_strtoconetype

```
MSKrescodee (MSKAPI MSK_strtoconetype) (
    MSKtask_t task,
    const char * str,
    MSKconetypeee * conetype)
```

Obtains cone type code corresponding to a cone type string.

Parameters

- **task** (*MSKtask_t*) – An optimization task. (input)
- **str** (*MSKstring_t*) – String corresponding to the cone type code **conetype**. (input)
- **conetype** (*MSKconetypeee by reference*) – The cone type corresponding to the string **str**. (output)

Return (*MSKrescodee*) – The function response code.

Groups *Names*

MSK_strtosk

```
MSKrescodee (MSKAPI MSK_strtosk) (
    MSKtask_t task,
    const char * str,
    MSKstakeye * sk)
```

Obtains the status key corresponding to an abbreviation string.

Parameters

- **task** (*MSKtask_t*) – An optimization task. (input)
- **str** (*MSKstring_t*) – A status key abbreviation string. (input)
- **sk** (*MSKstakeye by reference*) – Status key corresponding to the string. (output)

Return (*MSKrescodee*) – The function response code.

Groups *Names*

MSK_syeig

```
MSKrescodee (MSKAPI MSK_syeig) (
    MSKenv_t env,
    MSKuploe uplo,
    MSKint32t n,
    const MSKrealt * a,
    MSKrealt * w)
```

Computes all eigenvalues of a real symmetric matrix A . Given a matrix $A \in \mathbb{R}^{n \times n}$ it returns a vector $w \in \mathbb{R}^n$ containing the eigenvalues of A .

Parameters

- `env` (*MSKenv_t*) – The MOSEK environment. (input)
- `uplo` (*MSKuploe*) – Indicates whether the upper or lower triangular part is used. (input)
- `n` (*MSKint32t*) – Dimension of the symmetric input matrix. (input)
- `a` (*MSKrealt**) – A symmetric matrix A stored in column-major order. Only the part indicated by `uplo` is used. (input)
- `w` (*MSKrealt**) – Array of length at least `n` containing the eigenvalues of A . (output)

Return (*MSKrescodee*) – The function response code.

Groups *Linear algebra*

MSK_syevd

```
MSKrescodee (MSKAPI MSK_syevd) (
    MSKenv_t env,
    MSKuploe uplo,
    MSKint32t n,
    MSKrealt * a,
    MSKrealt * w)
```

Computes all the eigenvalues and eigenvectors a real symmetric matrix. Given the input matrix $A \in \mathbb{R}^{n \times n}$, this function returns a vector $w \in \mathbb{R}^n$ containing the eigenvalues of A and it also computes the eigenvectors of A . Therefore, this function computes the eigenvalue decomposition of A as

$$A = UVU^T,$$

where $V = \text{diag}(w)$ and U contains the eigenvectors of A .

Note that the matrix U overwrites the input data A .

Parameters

- `env` (*MSKenv_t*) – The MOSEK environment. (input)
- `uplo` (*MSKuploe*) – Indicates whether the upper or lower triangular part is used. (input)
- `n` (*MSKint32t*) – Dimension of the symmetric input matrix. (input)
- `a` (*MSKrealt**) – A symmetric matrix A stored in column-major order. Only the part indicated by `uplo` is used. On exit it will be overwritten by the matrix U . (input/output)
- `w` (*MSKrealt**) – Array of length at least `n` containing the eigenvalues of A . (output)

Return (*MSKrescodee*) – The function response code.

Groups *Linear algebra*

MSK_symnamtovalue

```
MSKboolean (MSKAPI MSK_symnamtovalue) (
    const char * name,
    char * value)
```

Obtains the value corresponding to a symbolic name defined by **MOSEK**.

Parameters

- **name** (*MSKstring_t*) – Symbolic name. (input)
- **value** (*MSKstring_t*) – The corresponding value. (output)

Return (*MSKboolean*) – Indicates if the symbolic name has been converted.

Groups *Parameters*

MSK_syrk

```
MSKrescodee (MSKAPI MSK_syrk) (
    MSKenv_t env,
    MSKuploe uplo,
    MSKtransposee trans,
    MSKint32t n,
    MSKint32t k,
    MSKrealt alpha,
    const MSKrealt * a,
    MSKrealt beta,
    MSKrealt * c)
```

Performs a symmetric rank- k update for a symmetric matrix.

Given a symmetric matrix $C \in \mathbb{R}^{n \times n}$, two scalars α, β and a matrix A of rank $k \leq n$, it computes either

$$C := \alpha AA^T + \beta C,$$

when **trans** is set to *MSK_TRANSPOSE_NO* and $A \in \mathbb{R}^{n \times k}$, or

$$C := \alpha A^T A + \beta C,$$

when **trans** is set to *MSK_TRANSPOSE_YES* and $A \in \mathbb{R}^{k \times n}$.

Only the part of C indicated by **uplo** is used and only that part is updated with the result.

Parameters

- **env** (*MSKenv_t*) – The MOSEK environment. (input)
- **uplo** (*MSKuploe*) – Indicates whether the upper or lower triangular part of C is used. (input)
- **trans** (*MSKtransposee*) – Indicates whether the matrix A must be transposed. (input)
- **n** (*MSKint32t*) – Specifies the order of C . (input)
- **k** (*MSKint32t*) – Indicates the number of rows or columns of A , depending on whether or not it is transposed, and its rank. (input)
- **alpha** (*MSKrealt*) – A scalar value multiplying the result of the matrix multiplication. (input)
- **a** (*MSKrealt**) – The pointer to the array storing matrix A in a column-major format. (input)
- **beta** (*MSKrealt*) – A scalar value that multiplies C . (input)
- **c** (*MSKrealt**) – The pointer to the array storing matrix C in a column-major format. (input/output)

Return (*MSKrescodee*) – The function response code.

Groups *Linear algebra*

MSK_toconic

```
MSKrescodee (MSKAPI MSK_toconic) (  
    MSKtask_t task)
```

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.

Parameters *task* (*MSKtask_t*) – An optimization task. (input)

Return (*MSKrescodee*) – The function response code.

Groups *Problem data - quadratic part*

MSK_unlinkfuncfromenvstream

```
MSKrescodee (MSKAPI MSK_unlinkfuncfromenvstream) (  
    MSKenv_t env,  
    MSKstreamtypee whichstream)
```

Disconnects a user-defined function from a stream.

Parameters

- *env* (*MSKenv_t*) – The MOSEK environment. (input)
- *whichstream* (*MSKstreamtypee*) – Index of the stream. (input)

Return (*MSKrescodee*) – The function response code.

Groups *Logging, Callback*

MSK_unlinkfuncfromtaskstream

```
MSKrescodee (MSKAPI MSK_unlinkfuncfromtaskstream) (  
    MSKtask_t task,  
    MSKstreamtypee whichstream)
```

Disconnects a user-defined function from a task stream.

Parameters

- *task* (*MSKtask_t*) – An optimization task. (input)
- *whichstream* (*MSKstreamtypee*) – Index of the stream. (input)

Return (*MSKrescodee*) – The function response code.

Groups *Logging, Callback*

MSK_updatesolutioninfo

```
MSKrescodee (MSKAPI MSK_updatesolutioninfo) (  
    MSKtask_t task,  
    MSKsoltypee whichsol)
```

Update the information items related to the solution.

Parameters

- `task` (*MSKtask_t*) – An optimization task. (input)
- `whichsol` (*MSKsoltypee*) – Selects a solution. (input)

Return (*MSKrescodee*) – The function response code.

Groups *Information items and statistics*

MSK_utf8towchar

```
MSKrescodee (MSKAPI MSK_utf8towchar) (  
    const size_t outputlen,  
    size_t * len,  
    size_t * conv,  
    MSKwchart * output,  
    const char * input)
```

Converts an UTF8 string to a *MSKwchart* string.

Parameters

- `outputlen` (*size_t*) – The length of the output buffer. (input)
- `len` (*size_t**) – The length of the string contained in the output buffer. (output)
- `conv` (*size_t**) – Returns the number of characters converted, i.e. `input[conv]` is the first character which was not converted. If the whole string was converted, then `input[conv]=0`. (output)
- `output` (*MSKwchart**) – The input string converted to a *MSKwchart* string. (output)
- `input` (*MSKstring_t*) – The UTF8 input string. (input)

Return (*MSKrescodee*) – The function response code.

Groups *System, memory and debugging*

MSK_wchartoutf8

```
MSKrescodee (MSKAPI MSK_wchartoutf8) (  
    const size_t outputlen,  
    size_t * len,  
    size_t * conv,  
    char * output,  
    const MSKwchart * input)
```

Converts a *MSKwchart* string to an UTF8 string.

Parameters

- `outputlen` (*size_t*) – The length of the output buffer. (input)
- `len` (*size_t**) – The length of the string contained in the output buffer. (output)
- `conv` (*size_t**) – Returns the number of characters from converted, i.e. `input[conv]` is the first char which was not converted. If the whole string was converted, then `input[conv]=0`. (output)
- `output` (*MSKstring_t*) – The input string converted to a UTF8 string. (output)
- `input` (*MSKwchart**) – The *MSKwchart* input string. (input)

Return (*MSKrescodee*) – The function response code.

Groups *System, memory and debugging*

MSK_whichparam

```
MSKrescodee (MSKAPI MSK_whichparam) (
    MSKtask_t task,
    const char * parname,
    MSKparametertypee * partye,
    MSKint32t * param)
```

Checks if `parname` is a valid parameter name. If yes then `partye` and `param` denote the type and the index of the parameter, respectively.

Parameters

- `task` (*MSKtask_t*) – An optimization task. (input)
- `parname` (*MSKstring_t*) – Parameter name. (input)
- `partye` (*MSKparametertypee by reference*) – Parameter type. (output)
- `param` (*MSKint32t by reference*) – Which parameter. (output)

Return (*MSKrescodee*) – The function response code.

Groups *Parameters, Names*

MSK_writedata

```
MSKrescodee (MSKAPI MSK_writedata) (
    MSKtask_t task,
    const char * 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 *MSK_IPAR_WRITE_GENERIC_NAMES* parameter to *MSK_ON*.

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

Parameters

- `task` (*MSKtask_t*) – An optimization task. (input)
- `filename` (*MSKstring_t*) – A valid file name. (input)

Return (*MSKrescodee*) – The function response code.

Groups *Input/Output*

MSK_writejsonsol

```
MSKrescodee (MSKAPI MSK_writejsonsol) (
    MSKtask_t task,
    const char * filename)
```

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

Parameters

- `task` (*MSKtask_t*) – An optimization task. (input)
- `filename` (*MSKstring_t*) – A valid file name. (input)

Return (*MSKrescodee*) – The function response code.

Groups *Input/Output*

MSK_writeparamfile

```
MSKrescodee (MSKAPI MSK_writeparamfile) (
    MSKtask_t task,
    const char * filename)
```

Writes all the parameters to a parameter file.

Parameters

- **task** (*MSKtask_t*) – An optimization task. (input)
- **filename** (*MSKstring_t*) – A valid file name. (input)

Return (*MSKrescodee*) – The function response code.

Groups *Input/Output, Parameters*

MSK_writesolution

```
MSKrescodee (MSKAPI MSK_writesolution) (
    MSKtask_t task,
    MSKsoltypee whichsol,
    const char * filename)
```

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

Parameters

- **task** (*MSKtask_t*) – An optimization task. (input)
- **whichsol** (*MSKsoltypee*) – Selects a solution. (input)
- **filename** (*MSKstring_t*) – A valid file name. (input)

Return (*MSKrescodee*) – The function response code.

Groups *Input/Output*

MSK_writetask

```
MSKrescodee (MSKAPI MSK_writetask) (
    MSKtask_t task,
    const char * 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

- **task** (*MSKtask_t*) – An optimization task. (input)
- **filename** (*MSKstring_t*) – A valid file name. (input)

Return (*MSKrescodee*) – The function response code.

Groups *Input/Output*

15.4 Parameters grouped by topic

Analysis

- *MSK_DPAR_ANA_SOL_INFEAS_TOL*
- *MSK_IPAR_ANA_SOL_BASIS*
- *MSK_IPAR_ANA_SOL_PRINT_VIOLATED*
- *MSK_IPAR_LOG_ANA_PRO*

Basis identification

- *MSK_DPAR_SIM_LU_TOL_REL_PIV*
- *MSK_IPAR_BI_CLEAN_OPTIMIZER*
- *MSK_IPAR_BI_IGNORE_MAX_ITER*
- *MSK_IPAR_BI_IGNORE_NUM_ERROR*
- *MSK_IPAR_BI_MAX_ITERATIONS*
- *MSK_IPAR_INTPNT_BASIS*
- *MSK_IPAR_LOG_BI*
- *MSK_IPAR_LOG_BI_FREQ*

Conic interior-point method

- *MSK_DPAR_INTPNT_CO_TOL_DFEAS*
- *MSK_DPAR_INTPNT_CO_TOL_INFEAS*
- *MSK_DPAR_INTPNT_CO_TOL_MU_RED*
- *MSK_DPAR_INTPNT_CO_TOL_NEAR_REL*
- *MSK_DPAR_INTPNT_CO_TOL_PFEAS*
- *MSK_DPAR_INTPNT_CO_TOL_REL_GAP*

Data check

- *MSK_DPAR_DATA_SYM_MAT_TOL*
- *MSK_DPAR_DATA_SYM_MAT_TOL_HUGE*
- *MSK_DPAR_DATA_SYM_MAT_TOL_LARGE*
- *MSK_DPAR_DATA_TOL_AIJ_HUGE*
- *MSK_DPAR_DATA_TOL_AIJ_LARGE*
- *MSK_DPAR_DATA_TOL_BOUND_INF*
- *MSK_DPAR_DATA_TOL_BOUND_WRN*
- *MSK_DPAR_DATA_TOL_C_HUGE*
- *MSK_DPAR_DATA_TOL_CJ_LARGE*
- *MSK_DPAR_DATA_TOL_QIJ*
- *MSK_DPAR_DATA_TOL_X*
- *MSK_DPAR_SEMIDEFINITE_TOL_APPROX*
- *MSK_IPAR_CHECK_CONVEXITY*
- *MSK_IPAR_LOG_CHECK_CONVEXITY*

Data input/output

- *MSK_IPAR_INFEAS_REPORT_AUTO*
- *MSK_IPAR_LOG_FILE*
- *MSK_IPAR_OPF_WRITE_HEADER*
- *MSK_IPAR_OPF_WRITE_HINTS*
- *MSK_IPAR_OPF_WRITE_LINE_LENGTH*
- *MSK_IPAR_OPF_WRITE_PARAMETERS*
- *MSK_IPAR_OPF_WRITE_PROBLEM*
- *MSK_IPAR_OPF_WRITE_SOL_BAS*
- *MSK_IPAR_OPF_WRITE_SOL_ITG*
- *MSK_IPAR_OPF_WRITE_SOL_ITR*
- *MSK_IPAR_OPF_WRITE_SOLUTIONS*
- *MSK_IPAR_PARAM_READ_CASE_NAME*
- *MSK_IPAR_PARAM_READ_IGN_ERROR*
- *MSK_IPAR_PTF_WRITE_TRANSFORM*
- *MSK_IPAR_READ_DEBUG*
- *MSK_IPAR_READ_KEEP_FREE_CON*
- *MSK_IPAR_READ_LP_DROP_NEW_VARS_IN_BOU*
- *MSK_IPAR_READ_LP_QUOTED_NAMES*
- *MSK_IPAR_READ_MPS_FORMAT*
- *MSK_IPAR_READ_MPS_WIDTH*
- *MSK_IPAR_READ_TASK_IGNORE_PARAM*
- *MSK_IPAR_SOL_READ_NAME_WIDTH*
- *MSK_IPAR_SOL_READ_WIDTH*
- *MSK_IPAR_WRITE_BAS_CONSTRAINTS*
- *MSK_IPAR_WRITE_BAS_HEAD*
- *MSK_IPAR_WRITE_BAS_VARIABLES*
- *MSK_IPAR_WRITE_COMPRESSION*
- *MSK_IPAR_WRITE_DATA_PARAM*
- *MSK_IPAR_WRITE_FREE_CON*
- *MSK_IPAR_WRITE_GENERIC_NAMES*
- *MSK_IPAR_WRITE_GENERIC_NAMES_IO*
- *MSK_IPAR_WRITE_IGNORE_INCOMPATIBLE_ITEMS*
- *MSK_IPAR_WRITE_INT_CONSTRAINTS*
- *MSK_IPAR_WRITE_INT_HEAD*
- *MSK_IPAR_WRITE_INT_VARIABLES*

- *MSK_IPAR_WRITE_LP_FULL_OBJ*
- *MSK_IPAR_WRITE_LP_LINE_WIDTH*
- *MSK_IPAR_WRITE_LP_QUOTED_NAMES*
- *MSK_IPAR_WRITE_LP_STRICT_FORMAT*
- *MSK_IPAR_WRITE_LP_TERMS_PER_LINE*
- *MSK_IPAR_WRITE_MPS_FORMAT*
- *MSK_IPAR_WRITE_MPS_INT*
- *MSK_IPAR_WRITE_PRECISION*
- *MSK_IPAR_WRITE_SOL_BARVARIABLES*
- *MSK_IPAR_WRITE_SOL_CONSTRAINTS*
- *MSK_IPAR_WRITE_SOL_HEAD*
- *MSK_IPAR_WRITE_SOL_IGNORE_INVALID_NAMES*
- *MSK_IPAR_WRITE_SOL_VARIABLES*
- *MSK_IPAR_WRITE_TASK_INC_SOL*
- *MSK_IPAR_WRITE_XML_MODE*
- *MSK_SPAR_BAS_SOL_FILE_NAME*
- *MSK_SPAR_DATA_FILE_NAME*
- *MSK_SPAR_DEBUG_FILE_NAME*
- *MSK_SPAR_INT_SOL_FILE_NAME*
- *MSK_SPAR_ITR_SOL_FILE_NAME*
- *MSK_SPAR_MIO_DEBUG_STRING*
- *MSK_SPAR_PARAM_COMMENT_SIGN*
- *MSK_SPAR_PARAM_READ_FILE_NAME*
- *MSK_SPAR_PARAM_WRITE_FILE_NAME*
- *MSK_SPAR_READ_MPS_BOU_NAME*
- *MSK_SPAR_READ_MPS_OBJ_NAME*
- *MSK_SPAR_READ_MPS_RAN_NAME*
- *MSK_SPAR_READ_MPS_RHS_NAME*
- *MSK_SPAR_SENSITIVITY_FILE_NAME*
- *MSK_SPAR_SENSITIVITY_RES_FILE_NAME*
- *MSK_SPAR_SOL_FILTER_XC_LOW*
- *MSK_SPAR_SOL_FILTER_XC_UPR*
- *MSK_SPAR_SOL_FILTER_XX_LOW*
- *MSK_SPAR_SOL_FILTER_XX_UPR*
- *MSK_SPAR_STAT_FILE_NAME*
- *MSK_SPAR_STAT_KEY*
- *MSK_SPAR_STAT_NAME*
- *MSK_SPAR_WRITE_LP_GEN_VAR_NAME*

Debugging

- *MSK_IPAR_AUTO_SORT_A_BEFORE_OPT*

Dual simplex

- *MSK_IPAR_SIM_DUAL_CRASH*
- *MSK_IPAR_SIM_DUAL_RESTRICT_SELECTION*
- *MSK_IPAR_SIM_DUAL_SELECTION*

Infeasibility report

- *MSK_IPAR_INFEAS_GENERIC_NAMES*
- *MSK_IPAR_INFEAS_REPORT_LEVEL*
- *MSK_IPAR_LOG_INFEAS_ANA*

Interior-point method

- *MSK_DPAR_CHECK_CONVEXITY_REL_TOL*
- *MSK_DPAR_INTPNT_CO_TOL_DFEAS*
- *MSK_DPAR_INTPNT_CO_TOL_INFEAS*
- *MSK_DPAR_INTPNT_CO_TOL_MU_RED*
- *MSK_DPAR_INTPNT_CO_TOL_NEAR_REL*
- *MSK_DPAR_INTPNT_CO_TOL_PFEAS*
- *MSK_DPAR_INTPNT_CO_TOL_REL_GAP*
- *MSK_DPAR_INTPNT_QO_TOL_DFEAS*
- *MSK_DPAR_INTPNT_QO_TOL_INFEAS*
- *MSK_DPAR_INTPNT_QO_TOL_MU_RED*
- *MSK_DPAR_INTPNT_QO_TOL_NEAR_REL*
- *MSK_DPAR_INTPNT_QO_TOL_PFEAS*
- *MSK_DPAR_INTPNT_QO_TOL_REL_GAP*
- *MSK_DPAR_INTPNT_TOL_DFEAS*
- *MSK_DPAR_INTPNT_TOL_DSAFE*
- *MSK_DPAR_INTPNT_TOL_INFEAS*
- *MSK_DPAR_INTPNT_TOL_MU_RED*
- *MSK_DPAR_INTPNT_TOL_PATH*
- *MSK_DPAR_INTPNT_TOL_PFEAS*
- *MSK_DPAR_INTPNT_TOL_PSAFE*
- *MSK_DPAR_INTPNT_TOL_REL_GAP*
- *MSK_DPAR_INTPNT_TOL_REL_STEP*

- *MSK_DPAR_INTPNT_TOL_STEP_SIZE*
- *MSK_DPAR_QCQO_REFORMULATE_REL_DROP_TOL*
- *MSK_IPAR_BI_IGNORE_MAX_ITER*
- *MSK_IPAR_BI_IGNORE_NUM_ERROR*
- *MSK_IPAR_INTPNT_BASIS*
- *MSK_IPAR_INTPNT_DIFF_STEP*
- *MSK_IPAR_INTPNT_HOTSTART*
- *MSK_IPAR_INTPNT_MAX_ITERATIONS*
- *MSK_IPAR_INTPNT_MAX_NUM_COR*
- *MSK_IPAR_INTPNT_MAX_NUM_REFINEMENT_STEPS*
- *MSK_IPAR_INTPNT_OFF_COL_TRH*
- *MSK_IPAR_INTPNT_ORDER_GP_NUM_SEEDS*
- *MSK_IPAR_INTPNT_ORDER_METHOD*
- *MSK_IPAR_INTPNT_PURIFY*
- *MSK_IPAR_INTPNT_REGULARIZATION_USE*
- *MSK_IPAR_INTPNT_SCALING*
- *MSK_IPAR_INTPNT_SOLVE_FORM*
- *MSK_IPAR_INTPNT_STARTING_POINT*
- *MSK_IPAR_LOG_INTPNT*

License manager

- *MSK_IPAR_CACHE_LICENSE*
- *MSK_IPAR_LICENSE_DEBUG*
- *MSK_IPAR_LICENSE_PAUSE_TIME*
- *MSK_IPAR_LICENSE_SUPPRESS_EXPIRE_WRNS*
- *MSK_IPAR_LICENSE_TRH_EXPIRY_WRN*
- *MSK_IPAR_LICENSE_WAIT*

Logging

- *MSK_IPAR_LOG*
- *MSK_IPAR_LOG_ANA_PRO*
- *MSK_IPAR_LOG_BI*
- *MSK_IPAR_LOG_BI_FREQ*
- *MSK_IPAR_LOG_CUT_SECOND_OPT*
- *MSK_IPAR_LOG_EXPAND*
- *MSK_IPAR_LOG_FEAS_REPAIR*

- *MSK_IPAR_LOG_FILE*
- *MSK_IPAR_LOG_INCLUDE_SUMMARY*
- *MSK_IPAR_LOG_INFEAS_ANA*
- *MSK_IPAR_LOG_INTPNT*
- *MSK_IPAR_LOG_LOCAL_INFO*
- *MSK_IPAR_LOG_MIO*
- *MSK_IPAR_LOG_MIO_FREQ*
- *MSK_IPAR_LOG_ORDER*
- *MSK_IPAR_LOG_PRESOLVE*
- *MSK_IPAR_LOG_RESPONSE*
- *MSK_IPAR_LOG_SENSITIVITY*
- *MSK_IPAR_LOG_SENSITIVITY_OPT*
- *MSK_IPAR_LOG_SIM*
- *MSK_IPAR_LOG_SIM_FREQ*
- *MSK_IPAR_LOG_STORAGE*

Mixed-integer optimization

- *MSK_DPAR_MIO_MAX_TIME*
- *MSK_DPAR_MIO_REL_GAP_CONST*
- *MSK_DPAR_MIO_TOL_ABS_GAP*
- *MSK_DPAR_MIO_TOL_ABS_RELAX_INT*
- *MSK_DPAR_MIO_TOL_FEAS*
- *MSK_DPAR_MIO_TOL_REL_DUAL_BOUND_IMPROVEMENT*
- *MSK_DPAR_MIO_TOL_REL_GAP*
- *MSK_IPAR_LOG_MIO*
- *MSK_IPAR_LOG_MIO_FREQ*
- *MSK_IPAR_MIO_BRANCH_DIR*
- *MSK_IPAR_MIO_CONIC_OUTER_APPROXIMATION*
- *MSK_IPAR_MIO_CUT_CLIQUE*
- *MSK_IPAR_MIO_CUT_CMIR*
- *MSK_IPAR_MIO_CUT_GMI*
- *MSK_IPAR_MIO_CUT_IMPLIED_BOUND*
- *MSK_IPAR_MIO_CUT_KNAPSACK_COVER*
- *MSK_IPAR_MIO_CUT_SELECTION_LEVEL*
- *MSK_IPAR_MIO_FEASPUMP_LEVEL*
- *MSK_IPAR_MIO_HEURISTIC_LEVEL*

- *MSK_IPAR_MIO_MAX_NUM_BRANCHES*
- *MSK_IPAR_MIO_MAX_NUM_RELAXS*
- *MSK_IPAR_MIO_MAX_NUM_ROOT_CUT_ROUNDS*
- *MSK_IPAR_MIO_MAX_NUM_SOLUTIONS*
- *MSK_IPAR_MIO_NODE_OPTIMIZER*
- *MSK_IPAR_MIO_NODE_SELECTION*
- *MSK_IPAR_MIO_PERSPECTIVE_REFORMULATE*
- *MSK_IPAR_MIO_PROBING_LEVEL*
- *MSK_IPAR_MIO_PROPAGATE_OBJECTIVE_CONSTRAINT*
- *MSK_IPAR_MIO_RINS_MAX_NODES*
- *MSK_IPAR_MIO_ROOT_OPTIMIZER*
- *MSK_IPAR_MIO_ROOT_REPEAT_PRESOLVE_LEVEL*
- *MSK_IPAR_MIO_SEED*
- *MSK_IPAR_MIO_VB_DETECTION_LEVEL*

Output information

- *MSK_IPAR_INFEAS_REPORT_LEVEL*
- *MSK_IPAR_LICENSE_SUPPRESS_EXPIRE_WRNS*
- *MSK_IPAR_LICENSE_TRH_EXPIRY_WRN*
- *MSK_IPAR_LOG*
- *MSK_IPAR_LOG_BI*
- *MSK_IPAR_LOG_BI_FREQ*
- *MSK_IPAR_LOG_CUT_SECOND_OPT*
- *MSK_IPAR_LOG_EXPAND*
- *MSK_IPAR_LOG_FEAS_REPAIR*
- *MSK_IPAR_LOG_FILE*
- *MSK_IPAR_LOG_INCLUDE_SUMMARY*
- *MSK_IPAR_LOG_INFEAS_ANA*
- *MSK_IPAR_LOG_INTPNT*
- *MSK_IPAR_LOG_LOCAL_INFO*
- *MSK_IPAR_LOG_MIO*
- *MSK_IPAR_LOG_MIO_FREQ*
- *MSK_IPAR_LOG_ORDER*
- *MSK_IPAR_LOG_RESPONSE*
- *MSK_IPAR_LOG_SENSITIVITY*
- *MSK_IPAR_LOG_SENSITIVITY_OPT*

- *MSK_IPAR_LOG_SIM*
- *MSK_IPAR_LOG_SIM_FREQ*
- *MSK_IPAR_LOG_SIM_MINOR*
- *MSK_IPAR_LOG_STORAGE*
- *MSK_IPAR_MAX_NUM_WARNINGS*

Overall solver

- *MSK_IPAR_BI_CLEAN_OPTIMIZER*
- *MSK_IPAR_INFEAS_PREFER_PRIMAL*
- *MSK_IPAR_LICENSE_WAIT*
- *MSK_IPAR_MIO_MODE*
- *MSK_IPAR_OPTIMIZER*
- *MSK_IPAR_PRESOLVE_LEVEL*
- *MSK_IPAR_PRESOLVE_MAX_NUM_REDUCTIONS*
- *MSK_IPAR_PRESOLVE_USE*
- *MSK_IPAR_PRIMAL_REPAIR_OPTIMIZER*
- *MSK_IPAR_SENSITIVITY_ALL*
- *MSK_IPAR_SENSITIVITY_OPTIMIZER*
- *MSK_IPAR_SENSITIVITY_TYPE*
- *MSK_IPAR_SOLUTION_CALLBACK*

Overall system

- *MSK_IPAR_AUTO_UPDATE_SOL_INFO*
- *MSK_IPAR_INTPNT_MULTI_THREAD*
- *MSK_IPAR_LICENSE_WAIT*
- *MSK_IPAR_LOG_STORAGE*
- *MSK_IPAR_MT_SPINCOUNT*
- *MSK_IPAR_NUM_THREADS*
- *MSK_IPAR_REMOVE_UNUSED_SOLUTIONS*
- *MSK_IPAR_TIMING_LEVEL*
- *MSK_SPAR_REMOTE_ACCESS_TOKEN*

Presolve

- *MSK_DPAR_PRESOLVE_TOL_ABS_LINDEP*
- *MSK_DPAR_PRESOLVE_TOL_AIJ*
- *MSK_DPAR_PRESOLVE_TOL_REL_LINDEP*
- *MSK_DPAR_PRESOLVE_TOL_S*
- *MSK_DPAR_PRESOLVE_TOL_X*
- *MSK_IPAR_PRESOLVE_ELIMINATOR_MAX_FILL*
- *MSK_IPAR_PRESOLVE_ELIMINATOR_MAX_NUM_TRIES*
- *MSK_IPAR_PRESOLVE_LEVEL*
- *MSK_IPAR_PRESOLVE_LINDEP_ABS_WORK_TRH*
- *MSK_IPAR_PRESOLVE_LINDEP_REL_WORK_TRH*
- *MSK_IPAR_PRESOLVE_LINDEP_USE*
- *MSK_IPAR_PRESOLVE_MAX_NUM_PASS*
- *MSK_IPAR_PRESOLVE_MAX_NUM_REDUCTIONS*
- *MSK_IPAR_PRESOLVE_USE*

Primal simplex

- *MSK_IPAR_SIM_PRIMAL_CRASH*
- *MSK_IPAR_SIM_PRIMAL_RESTRICT_SELECTION*
- *MSK_IPAR_SIM_PRIMAL_SELECTION*

Progress callback

- *MSK_IPAR_SOLUTION_CALLBACK*

Simplex optimizer

- *MSK_DPAR_BASIS_REL_TOL_S*
- *MSK_DPAR_BASIS_TOL_S*
- *MSK_DPAR_BASIS_TOL_X*
- *MSK_DPAR_SIM_LU_TOL_REL_PIV*
- *MSK_DPAR_SIMPLEX_ABS_TOL_PIV*
- *MSK_IPAR_BASIS_SOLVE_USE_PLUS_ONE*
- *MSK_IPAR_LOG_SIM*
- *MSK_IPAR_LOG_SIM_FREQ*
- *MSK_IPAR_LOG_SIM_MINOR*
- *MSK_IPAR_SENSITIVITY_OPTIMIZER*
- *MSK_IPAR_SIM_BASIS_FACTOR_USE*

- *MSK_IPAR_SIM_DEGEN*
- *MSK_IPAR_SIM_DUAL_PHASEONE_METHOD*
- *MSK_IPAR_SIM_EXPLOIT_DUPVEC*
- *MSK_IPAR_SIM_HOTSTART*
- *MSK_IPAR_SIM_HOTSTART_LU*
- *MSK_IPAR_SIM_MAX_ITERATIONS*
- *MSK_IPAR_SIM_MAX_NUM_SETBACKS*
- *MSK_IPAR_SIM_NON_SINGULAR*
- *MSK_IPAR_SIM_PRIMAL_PHASEONE_METHOD*
- *MSK_IPAR_SIM_REFACTOR_FREQ*
- *MSK_IPAR_SIM_REFORMULATION*
- *MSK_IPAR_SIM_SAVE_LU*
- *MSK_IPAR_SIM_SCALING*
- *MSK_IPAR_SIM_SCALING_METHOD*
- *MSK_IPAR_SIM_SEED*
- *MSK_IPAR_SIM_SOLVE_FORM*
- *MSK_IPAR_SIM_STABILITY_PRIORITY*
- *MSK_IPAR_SIM_SWITCH_OPTIMIZER*

Solution input/output

- *MSK_IPAR_INFEAS_REPORT_AUTO*
- *MSK_IPAR_SOL_FILTER_KEEP_BASIC*
- *MSK_IPAR_SOL_FILTER_KEEP_RANGED*
- *MSK_IPAR_SOL_READ_NAME_WIDTH*
- *MSK_IPAR_SOL_READ_WIDTH*
- *MSK_IPAR_WRITE_BAS_CONSTRAINTS*
- *MSK_IPAR_WRITE_BAS_HEAD*
- *MSK_IPAR_WRITE_BAS_VARIABLES*
- *MSK_IPAR_WRITE_INT_CONSTRAINTS*
- *MSK_IPAR_WRITE_INT_HEAD*
- *MSK_IPAR_WRITE_INT_VARIABLES*
- *MSK_IPAR_WRITE_SOL_BARVARIABLES*
- *MSK_IPAR_WRITE_SOL_CONSTRAINTS*
- *MSK_IPAR_WRITE_SOL_HEAD*
- *MSK_IPAR_WRITE_SOL_IGNORE_INVALID_NAMES*
- *MSK_IPAR_WRITE_SOL_VARIABLES*

- *MSK_SPAR_BAS_SOL_FILE_NAME*
- *MSK_SPAR_INT_SOL_FILE_NAME*
- *MSK_SPAR_ITR_SOL_FILE_NAME*
- *MSK_SPAR_SOL_FILTER_XC_LOW*
- *MSK_SPAR_SOL_FILTER_XC_UPR*
- *MSK_SPAR_SOL_FILTER_XX_LOW*
- *MSK_SPAR_SOL_FILTER_XX_UPR*

Termination criteria

- *MSK_DPAR_BASIS_REL_TOL_S*
- *MSK_DPAR_BASIS_TOL_S*
- *MSK_DPAR_BASIS_TOL_X*
- *MSK_DPAR_INTPNT_CO_TOL_DFEAS*
- *MSK_DPAR_INTPNT_CO_TOL_INFEAS*
- *MSK_DPAR_INTPNT_CO_TOL_MU_RED*
- *MSK_DPAR_INTPNT_CO_TOL_NEAR_REL*
- *MSK_DPAR_INTPNT_CO_TOL_PFEAS*
- *MSK_DPAR_INTPNT_CO_TOL_REL_GAP*
- *MSK_DPAR_INTPNT_QO_TOL_DFEAS*
- *MSK_DPAR_INTPNT_QO_TOL_INFEAS*
- *MSK_DPAR_INTPNT_QO_TOL_MU_RED*
- *MSK_DPAR_INTPNT_QO_TOL_NEAR_REL*
- *MSK_DPAR_INTPNT_QO_TOL_PFEAS*
- *MSK_DPAR_INTPNT_QO_TOL_REL_GAP*
- *MSK_DPAR_INTPNT_TOL_DFEAS*
- *MSK_DPAR_INTPNT_TOL_INFEAS*
- *MSK_DPAR_INTPNT_TOL_MU_RED*
- *MSK_DPAR_INTPNT_TOL_PFEAS*
- *MSK_DPAR_INTPNT_TOL_REL_GAP*
- *MSK_DPAR_LOWER_OBJ_CUT*
- *MSK_DPAR_LOWER_OBJ_CUT_FINITE_TRH*
- *MSK_DPAR_MIO_MAX_TIME*
- *MSK_DPAR_MIO_REL_GAP_CONST*
- *MSK_DPAR_MIO_TOL_REL_GAP*
- *MSK_DPAR_OPTIMIZER_MAX_TIME*
- *MSK_DPAR_UPPER_OBJ_CUT*

- *MSK_DPAR_UPPER_OBJ_CUT_FINITE_TRH*
- *MSK_IPAR_BI_MAX_ITERATIONS*
- *MSK_IPAR_INTPNT_MAX_ITERATIONS*
- *MSK_IPAR_MIO_MAX_NUM_BRANCHES*
- *MSK_IPAR_MIO_MAX_NUM_ROOT_CUT_ROUNDS*
- *MSK_IPAR_MIO_MAX_NUM_SOLUTIONS*
- *MSK_IPAR_SIM_MAX_ITERATIONS*

Other

- *MSK_IPAR_COMPRESS_STATFILE*

15.5 Parameters (alphabetical list sorted by type)

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

15.5.1 Double parameters

MSKdparame

The enumeration type containing all double parameters.

MSK_DPAR_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 `MSK_putdoupam(task, MSK_DPAR_ANA_SOL_INFEAS_TOL, 1e-6)`

Groups *Analysis*

MSK_DPAR_BASIS_REL_TOL_S

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

Default 1.0e-12

Accepted [0.0; +inf]

Example `MSK_putdoupam(task, MSK_DPAR_BASIS_REL_TOL_S, 1.0e-12)`

Groups *Simplex optimizer, Termination criteria*

MSK_DPAR_BASIS_TOL_S

Maximum absolute dual bound violation in an optimal basic solution.

Default 1.0e-6

Accepted [1.0e-9; +inf]

Example `MSK_putdoupam(task, MSK_DPAR_BASIS_TOL_S, 1.0e-6)`

Groups *Simplex optimizer, Termination criteria*

MSK_DPAR_BASIS_TOL_X

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

Default 1.0e-6

Accepted [1.0e-9; +inf]

Example `MSK_putdoupam(task, MSK_DPAR_BASIS_TOL_X, 1.0e-6)`

Groups *Simplex optimizer, Termination criteria*

MSK_DPAR_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 MSK_putdoupparam(task, MSK_DPAR_CHECK_CONVEXITY_REL_TOL, 1e-10)

Groups *Interior-point method*

MSK_DPAR_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 MSK_putdoupparam(task, MSK_DPAR_DATA_SYM_MAT_TOL, 1.0e-12)

Groups *Data check*

MSK_DPAR_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 MSK_putdoupparam(task, MSK_DPAR_DATA_SYM_MAT_TOL_HUGE, 1.0e20)

Groups *Data check*

MSK_DPAR_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 MSK_putdoupparam(task, MSK_DPAR_DATA_SYM_MAT_TOL_LARGE, 1.0e10)

Groups *Data check*

MSK_DPAR_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 MSK_putdoupparam(task, MSK_DPAR_DATA_TOL_AIJ_HUGE, 1.0e20)

Groups *Data check*

MSK_DPAR_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 MSK_putdoupparam(task, MSK_DPAR_DATA_TOL_AIJ_LARGE, 1.0e10)

Groups *Data check*

MSK_DPAR_DATA_TOL_BOUND_INF

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

Default 1.0e16

Accepted [0.0; +inf]

Example MSK_putdoupparam(task, MSK_DPAR_DATA_TOL_BOUND_INF, 1.0e16)

Groups *Data check*

MSK_DPAR_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 MSK_putdoupparam(task, MSK_DPAR_DATA_TOL_BOUND_WRN, 1.0e8)

Groups *Data check*

MSK_DPAR_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 MSK_putdoupparam(task, MSK_DPAR_DATA_TOL_C_HUGE, 1.0e16)

Groups *Data check*

MSK_DPAR_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 MSK_putdoupparam(task, MSK_DPAR_DATA_TOL_CJ_LARGE, 1.0e8)

Groups *Data check*

MSK_DPAR_DATA_TOL_QIJ

Absolute zero tolerance for elements in Q matrices.

Default 1.0e-16

Accepted [0.0; +inf]

Example MSK_putdoupparam(task, MSK_DPAR_DATA_TOL_QIJ, 1.0e-16)

Groups *Data check*

MSK_DPAR_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 MSK_putdoupparam(task, MSK_DPAR_DATA_TOL_X, 1.0e-8)

Groups *Data check*

MSK_DPAR_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 MSK_putdoupparam(task, MSK_DPAR_INTPNT_CO_TOL_DFEAS, 1.0e-8)

See also *MSK_DPAR_INTPNT_CO_TOL_NEAR_REL*

Groups *Interior-point method, Termination criteria, Conic interior-point method*

MSK_DPAR_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 `MSK_putdouparam(task, MSK_DPAR_INTPNT_CO_TOL_INFEAS, 1.0e-12)`

Groups *Interior-point method, Termination criteria, Conic interior-point method*

MSK_DPAR_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 `MSK_putdouparam(task, MSK_DPAR_INTPNT_CO_TOL_MU_RED, 1.0e-8)`

Groups *Interior-point method, Termination criteria, Conic interior-point method*

MSK_DPAR_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 `MSK_putdouparam(task, MSK_DPAR_INTPNT_CO_TOL_NEAR_REL, 1000)`

Groups *Interior-point method, Termination criteria, Conic interior-point method*

MSK_DPAR_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 `MSK_putdouparam(task, MSK_DPAR_INTPNT_CO_TOL_PFEAS, 1.0e-8)`

See also [*MSK_DPAR_INTPNT_CO_TOL_NEAR_REL*](#)

Groups *Interior-point method, Termination criteria, Conic interior-point method*

MSK_DPAR_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 `MSK_putdouparam(task, MSK_DPAR_INTPNT_CO_TOL_REL_GAP, 1.0e-8)`

See also [*MSK_DPAR_INTPNT_CO_TOL_NEAR_REL*](#)

Groups *Interior-point method, Termination criteria, Conic interior-point method*

MSK_DPAR_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 `MSK_putdouparam(task, MSK_DPAR_INTPNT_QO_TOL_DFEAS, 1.0e-8)`

See also [*MSK_DPAR_INTPNT_QO_TOL_NEAR_REL*](#)

Groups *Interior-point method, Termination criteria*

MSK_DPAR_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 `MSK_putdouparam(task, MSK_DPAR_INTPNT_QO_TOL_INFEAS, 1.0e-12)`

Groups *Interior-point method, Termination criteria*

MSK_DPAR_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 `MSK_putdouparam(task, MSK_DPAR_INTPNT_QO_TOL_MU_RED, 1.0e-8)`

Groups *Interior-point method, Termination criteria*

MSK_DPAR_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 `MSK_putdouparam(task, MSK_DPAR_INTPNT_QO_TOL_NEAR_REL, 1000)`

Groups *Interior-point method, Termination criteria*

MSK_DPAR_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 `MSK_putdouparam(task, MSK_DPAR_INTPNT_QO_TOL_PFEAS, 1.0e-8)`

See also [*MSK_DPAR_INTPNT_QO_TOL_NEAR_REL*](#)

Groups *Interior-point method, Termination criteria*

MSK_DPAR_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 `MSK_putdouparam(task, MSK_DPAR_INTPNT_QO_TOL_REL_GAP, 1.0e-8)`

See also [*MSK_DPAR_INTPNT_QO_TOL_NEAR_REL*](#)

Groups *Interior-point method, Termination criteria*

MSK_DPAR_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 `MSK_putdouparam(task, MSK_DPAR_INTPNT_TOL_DFEAS, 1.0e-8)`

Groups *Interior-point method, Termination criteria*

MSK_DPAR_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 MSK_putdoupparam(task, MSK_DPAR_INTPNT_TOL_DSAFE, 1.0)
Groups *Interior-point method*

MSK_DPAR_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 MSK_putdoupparam(task, MSK_DPAR_INTPNT_TOL_INFEAS, 1.0e-10)
Groups *Interior-point method, Termination criteria*

MSK_DPAR_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 MSK_putdoupparam(task, MSK_DPAR_INTPNT_TOL_MU_RED, 1.0e-16)
Groups *Interior-point method, Termination criteria*

MSK_DPAR_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 MSK_putdoupparam(task, MSK_DPAR_INTPNT_TOL_PATH, 1.0e-8)
Groups *Interior-point method*

MSK_DPAR_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 MSK_putdoupparam(task, MSK_DPAR_INTPNT_TOL_PFEAS, 1.0e-8)
Groups *Interior-point method, Termination criteria*

MSK_DPAR_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 MSK_putdoupparam(task, MSK_DPAR_INTPNT_TOL_PSAFE, 1.0)
Groups *Interior-point method*

MSK_DPAR_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 MSK_putdoupparam(task, MSK_DPAR_INTPNT_TOL_REL_GAP, 1.0e-8)
Groups *Termination criteria, Interior-point method*

MSK_DPAR_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 `MSK_putdouparam(task, MSK_DPAR_INTPNT_TOL_REL_STEP, 0.9999)`

Groups *Interior-point method*

MSK_DPAR_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 `MSK_putdouparam(task, MSK_DPAR_INTPNT_TOL_STEP_SIZE, 1.0e-6)`

Groups *Interior-point method*

MSK_DPAR_LOWER_OBJ_CUT

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

Default -1.0e30

Accepted [-inf; +inf]

Example `MSK_putdouparam(task, MSK_DPAR_LOWER_OBJ_CUT, -1.0e30)`

See also *MSK_DPAR_LOWER_OBJ_CUT_FINITE_TRH*

Groups *Termination criteria*

MSK_DPAR_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. *MSK_DPAR_LOWER_OBJ_CUT* is treated as $-\infty$.

Default -0.5e30

Accepted [-inf; +inf]

Example `MSK_putdouparam(task, MSK_DPAR_LOWER_OBJ_CUT_FINITE_TRH, -0.5e30)`

Groups *Termination criteria*

MSK_DPAR_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 `MSK_putdouparam(task, MSK_DPAR_MIO_MAX_TIME, -1.0)`

Groups *Mixed-integer optimization, Termination criteria*

MSK_DPAR_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 `MSK_putdouparam(task, MSK_DPAR_MIO_REL_GAP_CONST, 1.0e-10)`

Groups *Mixed-integer optimization, Termination criteria*

MSK_DPAR_MIO_TOL_ABS_GAP

Absolute optimality tolerance employed by the mixed-integer optimizer.

Default 0.0

Accepted [0.0; +inf]

Example MSK_putdouparam(task, MSK_DPAR_MIO_TOL_ABS_GAP, 0.0)

Groups *Mixed-integer optimization*

MSK_DPAR_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 MSK_putdouparam(task, MSK_DPAR_MIO_TOL_ABS_RELAX_INT, 1.0e-5)

Groups *Mixed-integer optimization*

MSK_DPAR_MIO_TOL_FEAS

Feasibility tolerance for mixed integer solver.

Default 1.0e-6

Accepted [1e-9; 1e-3]

Example MSK_putdouparam(task, MSK_DPAR_MIO_TOL_FEAS, 1.0e-6)

Groups *Mixed-integer optimization*

MSK_DPAR_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 MSK_putdouparam(task, MSK_DPAR_MIO_TOL_REL_DUAL_BOUND_IMPROVEMENT, 0.0)

Groups *Mixed-integer optimization*

MSK_DPAR_MIO_TOL_REL_GAP

Relative optimality tolerance employed by the mixed-integer optimizer.

Default 1.0e-4

Accepted [0.0; +inf]

Example MSK_putdouparam(task, MSK_DPAR_MIO_TOL_REL_GAP, 1.0e-4)

Groups *Mixed-integer optimization, Termination criteria*

MSK_DPAR_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 MSK_putdouparam(task, MSK_DPAR_OPTIMIZER_MAX_TIME, -1.0)

Groups *Termination criteria*

MSK_DPAR_PREOLVE_TOL_ABS_LINDEP

Absolute tolerance employed by the linear dependency checker.

Default 1.0e-6

Accepted [0.0; +inf]

Example MSK_putdouparam(task, MSK_DPAR_PREOLVE_TOL_ABS_LINDEP, 1.0e-6)

Groups *Presolve*

MSK_DPAR_PREOLVE_TOL_AIJ

Absolute zero tolerance employed for a_{ij} in the presolve.

Default 1.0e-12

Accepted [1.0e-15; +inf]

Example `MSK_putdoupparam(task, MSK_DPAR_PREOLVE_TOL_AIJ, 1.0e-12)`

Groups *Presolve*

MSK_DPAR_PREOLVE_TOL_REL_LINDEP

Relative tolerance employed by the linear dependency checker.

Default 1.0e-10

Accepted [0.0; +inf]

Example `MSK_putdoupparam(task, MSK_DPAR_PREOLVE_TOL_REL_LINDEP, 1.0e-10)`

Groups *Presolve*

MSK_DPAR_PREOLVE_TOL_S

Absolute zero tolerance employed for s_i in the presolve.

Default 1.0e-8

Accepted [0.0; +inf]

Example `MSK_putdoupparam(task, MSK_DPAR_PREOLVE_TOL_S, 1.0e-8)`

Groups *Presolve*

MSK_DPAR_PREOLVE_TOL_X

Absolute zero tolerance employed for x_j in the presolve.

Default 1.0e-8

Accepted [0.0; +inf]

Example `MSK_putdoupparam(task, MSK_DPAR_PREOLVE_TOL_X, 1.0e-8)`

Groups *Presolve*

MSK_DPAR_QCQO_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 `MSK_putdoupparam(task, MSK_DPAR_QCQO_REFORMULATE_REL_DROP_TOL, 1e-15)`

Groups *Interior-point method*

MSK_DPAR_SEMIDEFINITE_TOL_APPROX

Tolerance to define a matrix to be positive semidefinite.

Default 1.0e-10

Accepted [1.0e-15; +inf]

Example `MSK_putdoupparam(task, MSK_DPAR_SEMIDEFINITE_TOL_APPROX, 1.0e-10)`

Groups *Data check*

MSK_DPAR_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 `MSK_putdoupparam(task, MSK_DPAR_SIM_LU_TOL_REL_PIV, 0.01)`

Groups *Basis identification, Simplex optimizer*

MSK_DPAR_SIMPLEX_ABS_TOL_PIV

Absolute pivot tolerance employed by the simplex optimizers.

Default 1.0e-7

Accepted [1.0e-12; +inf]

Example MSK_putdoupparam(task, MSK_DPAR_SIMPLEX_ABS_TOL_PIV, 1.0e-7)

Groups *Simplex optimizer*

MSK_DPAR_UPPER_OBJ_CUT

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

Default 1.0e30

Accepted [-inf; +inf]

Example MSK_putdoupparam(task, MSK_DPAR_UPPER_OBJ_CUT, 1.0e30)

See also *MSK_DPAR_UPPER_OBJ_CUT_FINITE_TRH*

Groups *Termination criteria*

MSK_DPAR_UPPER_OBJ_CUT_FINITE_TRH

If the upper objective cut is greater than the value of this parameter, then the upper objective cut *MSK_DPAR_UPPER_OBJ_CUT* is treated as ∞ .

Default 0.5e30

Accepted [-inf; +inf]

Example MSK_putdoupparam(task, MSK_DPAR_UPPER_OBJ_CUT_FINITE_TRH, 0.5e30)

Groups *Termination criteria*

15.5.2 Integer parameters

MSKiparame

The enumeration type containing all integer parameters.

MSK_IPAR_ANA_SOL_BASIS

Controls whether the basis matrix is analyzed in solution analyzer.

Default *ON*

Accepted *ON*, *OFF* (see *MSKonoffkeye*)

Example MSK_putintparam(task, MSK_IPAR_ANA_SOL_BASIS, MSK_ON)

Groups *Analysis*

MSK_IPAR_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 *MSK_DPAR_ANA_SOL_INFEAS_TOL* will be printed.

Default *OFF*

Accepted *ON*, *OFF* (see *MSKonoffkeye*)

Example MSK_putintparam(task, MSK_IPAR_ANA_SOL_PRINT_VIOLATED, MSK_OFF)

Groups *Analysis*

MSK_IPAR_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 *MSKonoffkeye*)

Example MSK_putintparam(task, MSK_IPAR_AUTO_SORT_A_BEFORE_OPT, MSK_OFF)

Groups *Debugging*

MSK_IPAR_AUTO_UPDATE_SOL_INFO

Controls whether the solution information items are automatically updated after an optimization is performed.

Default *OFF*

Accepted *ON, OFF* (see *MSKonoffkeye*)

Example `MSK_putintparam(task, MSK_IPAR_AUTO_UPDATE_SOL_INFO, MSK_OFF)`

Groups *Overall system*

MSK_IPAR_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 *MSK_ON*, -1 is replaced by 1.

This has significance for the results returned by the *MSK_solvewithbasis* function.

Default *OFF*

Accepted *ON, OFF* (see *MSKonoffkeye*)

Example `MSK_putintparam(task, MSK_IPAR_BASIS_SOLVE_USE_PLUS_ONE, MSK_OFF)`

Groups *Simplex optimizer*

MSK_IPAR_BI_CLEAN_OPTIMIZER

Controls which simplex optimizer is used in the clean-up phase. Anything else than *MSK_OPTIMIZER_PRIMAL_SIMPLEX* or *MSK_OPTIMIZER_DUAL_SIMPLEX* is equivalent to *MSK_OPTIMIZER_FREE_SIMPLEX*.

Default *FREE*

Accepted *FREE, INTPNT, CONIC, PRIMAL_SIMPLEX, DUAL_SIMPLEX, FREE_SIMPLEX, MIXED_INT* (see *MSKoptimizertypee*)

Example `MSK_putintparam(task, MSK_IPAR_BI_CLEAN_OPTIMIZER, MSK_OPTIMIZER_FREE)`

Groups *Basis identification, Overall solver*

MSK_IPAR_BI_IGNORE_MAX_ITER

If the parameter *MSK_IPAR_INTPNT_BASIS* has the value *MSK_BI_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 *MSK_ON*.

Default *OFF*

Accepted *ON, OFF* (see *MSKonoffkeye*)

Example `MSK_putintparam(task, MSK_IPAR_BI_IGNORE_MAX_ITER, MSK_OFF)`

Groups *Interior-point method, Basis identification*

MSK_IPAR_BI_IGNORE_NUM_ERROR

If the parameter *MSK_IPAR_INTPNT_BASIS* has the value *MSK_BI_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 *MSK_ON*.

Default *OFF*

Accepted *ON, OFF* (see *MSKonoffkeye*)

Example `MSK_putintparam(task, MSK_IPAR_BI_IGNORE_NUM_ERROR, MSK_OFF)`

Groups *Interior-point method, Basis identification*

MSK_IPAR_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 `MSK_putintparam(task, MSK_IPAR_BI_MAX_ITERATIONS, 1000000)`

Groups *Basis identification, Termination criteria*

MSK_IPAR_CACHE_LICENSE

Specifies if the license is kept checked out for the lifetime of the **MOSEK** environment/model/process (*MSK_ON*) or returned to the server immediately after the optimization (*MSK_OFF*).

By default the license is checked out for the lifetime of the **MOSEK** environment by the first call to *MSK_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 *MSK_onoffkeye*)

Example *MSK_putintparam(task, MSK_IPAR_CACHE_LICENSE, MSK_ON)*

Groups *License manager*

MSK_IPAR_CHECK_CONVEXITY

Specify the level of convexity check on quadratic problems.

Default *FULL*

Accepted *NONE, SIMPLE, FULL* (see *MSK_checkconvexitytypee*)

Example *MSK_putintparam(task, MSK_IPAR_CHECK_CONVEXITY, MSK_CHECK_CONVEXITY_FULL)*

Groups *Data check*

MSK_IPAR_COMPRESS_STATFILE

Control compression of stat files.

Default *ON*

Accepted *ON, OFF* (see *MSK_onoffkeye*)

Example *MSK_putintparam(task, MSK_IPAR_COMPRESS_STATFILE, MSK_ON)*

MSK_IPAR_INFEAS_GENERIC_NAMES

Controls whether generic names are used when an infeasible subproblem is created.

Default *OFF*

Accepted *ON, OFF* (see *MSK_onoffkeye*)

Example *MSK_putintparam(task, MSK_IPAR_INFEAS_GENERIC_NAMES, MSK_OFF)*

Groups *Infeasibility report*

MSK_IPAR_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 *MSK_onoffkeye*)

Example *MSK_putintparam(task, MSK_IPAR_INFEAS_PREFER_PRIMAL, MSK_ON)*

Groups *Overall solver*

MSK_IPAR_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 *MSK_onoffkeye*)

Example *MSK_putintparam(task, MSK_IPAR_INFEAS_REPORT_AUTO, MSK_OFF)*

Groups *Data input/output, Solution input/output*

MSK_IPAR_INFEAS_REPORT_LEVEL

Controls the amount of information presented in an infeasibility report. Higher values imply more information.

Default 1

Accepted [0; +inf]

Example MSK_putintparam(task, MSK_IPAR_INFEAS_REPORT_LEVEL, 1)

Groups *Infeasibility report, Output information*

MSK_IPAR_INTPNT_BASIS

Controls whether the interior-point optimizer also computes an optimal basis.

Default *ALWAYS*

Accepted *NEVER, ALWAYS, NO_ERROR, IF_FEASIBLE, RESERVED* (see *MSKbasindtypee*)

Example MSK_putintparam(task, MSK_IPAR_INTPNT_BASIS, MSK_BI_ALWAYS)

See also *MSK_IPAR_BI_IGNORE_MAX_ITER, MSK_IPAR_BI_IGNORE_NUM_ERROR, MSK_IPAR_BI_MAX_ITERATIONS, MSK_IPAR_BI_CLEAN_OPTIMIZER*

Groups *Interior-point method, Basis identification*

MSK_IPAR_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 MSK_putintparam(task, MSK_IPAR_INTPNT_DIFF_STEP, MSK_ON)

Groups *Interior-point method*

MSK_IPAR_INTPNT_HOTSTART

Currently not in use.

Default *NONE*

Accepted *NONE, PRIMAL, DUAL, PRIMAL_DUAL* (see *MSKintpnthotstarte*)

Example MSK_putintparam(task, MSK_IPAR_INTPNT_HOTSTART, MSK_INTPNT_HOTSTART_NONE)

Groups *Interior-point method*

MSK_IPAR_INTPNT_MAX_ITERATIONS

Controls the maximum number of iterations allowed in the interior-point optimizer.

Default 400

Accepted [0; +inf]

Example MSK_putintparam(task, MSK_IPAR_INTPNT_MAX_ITERATIONS, 400)

Groups *Interior-point method, Termination criteria*

MSK_IPAR_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 MSK_putintparam(task, MSK_IPAR_INTPNT_MAX_NUM_COR, -1)

Groups *Interior-point method*

MSK_IPAR_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 MSK_putintparam(task, MSK_IPAR_INTPNT_MAX_NUM_REFINEMENT_STEPS, -1)

Groups *Interior-point method*

MSK_IPAR_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 *MSK_onoffkeye*)

Example MSK_putintparam(task, MSK_IPAR_INTPNT_MULTI_THREAD, MSK_ON)

Groups *Overall system*

MSK_IPAR_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 MSK_putintparam(task, MSK_IPAR_INTPNT_OFF_COL_TRH, 40)

Groups *Interior-point method*

MSK_IPAR_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 MSK_putintparam(task, MSK_IPAR_INTPNT_ORDER_GP_NUM_SEEDS, 0)

Groups *Interior-point method*

MSK_IPAR_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 *MSK_orderingtypee*)

Example MSK_putintparam(task, MSK_IPAR_INTPNT_ORDER_METHOD, MSK_ORDER_METHOD_FREE)

Groups *Interior-point method*

MSK_IPAR_INTPNT_PURIFY

Currently not in use.

Default *NONE*

Accepted *NONE, PRIMAL, DUAL, PRIMAL_DUAL, AUTO* (see *MSK_purifye*)

Example MSK_putintparam(task, MSK_IPAR_INTPNT_PURIFY, MSK_PURIFY_NONE)

Groups *Interior-point method*

MSK_IPAR_INTPNT_REGULARIZATION_USE

Controls whether regularization is allowed.

Default *ON*

Accepted *ON, OFF* (see *MSKonoffkeye*)

Example `MSK_putintparam(task, MSK_IPAR_INTPNT_REGULARIZATION_USE, MSK_ON)`

Groups *Interior-point method*

MSK_IPAR_INTPNT_SCALING

Controls how the problem is scaled before the interior-point optimizer is used.

Default *FREE*

Accepted *FREE, NONE, MODERATE, AGGRESSIVE* (see *MSKscalingtypee*)

Example `MSK_putintparam(task, MSK_IPAR_INTPNT_SCALING, MSK_SCALING_FREE)`

Groups *Interior-point method*

MSK_IPAR_INTPNT_SOLVE_FORM

Controls whether the primal or the dual problem is solved.

Default *FREE*

Accepted *FREE, PRIMAL, DUAL* (see *MSKsolveforme*)

Example `MSK_putintparam(task, MSK_IPAR_INTPNT_SOLVE_FORM, MSK_SOLVE_FREE)`

Groups *Interior-point method*

MSK_IPAR_INTPNT_STARTING_POINT

Starting point used by the interior-point optimizer.

Default *FREE*

Accepted *FREE, GUESS, CONSTANT, SATISFY_BOUNDS* (see *MSKstartpointtypee*)

Example `MSK_putintparam(task, MSK_IPAR_INTPNT_STARTING_POINT, MSK_STARTING_POINT_FREE)`

Groups *Interior-point method*

MSK_IPAR_LICENSE_DEBUG

This option is used to turn on debugging of the license manager.

Default *OFF*

Accepted *ON, OFF* (see *MSKonoffkeye*)

Example `MSK_putintparam(task, MSK_IPAR_LICENSE_DEBUG, MSK_OFF)`

Groups *License manager*

MSK_IPAR_LICENSE_PAUSE_TIME

If *MSK_IPAR_LICENSE_WAIT* is *MSK_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 `MSK_putintparam(task, MSK_IPAR_LICENSE_PAUSE_TIME, 100)`

Groups *License manager*

MSK_IPAR_LICENSE_SUPPRESS_EXPIRE_WRNS

Controls whether license features expire warnings are suppressed.

Default *OFF*

Accepted *ON, OFF* (see *MSKonoffkeye*)

Example `MSK_putintparam(task, MSK_IPAR_LICENSE_SUPPRESS_EXPIRE_WRNS, MSK_OFF)`

Groups *License manager, Output information*

MSK_IPAR_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 `MSK_putintparam(task, MSK_IPAR_LICENSE_TRH_EXPIRY_WRN, 7)`

Groups *License manager, Output information*

MSK_IPAR_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 *MSK_onoffkey*)

Example `MSK_putintparam(task, MSK_IPAR_LICENSE_WAIT, MSK_OFF)`

Groups *Overall solver, Overall system, License manager*

MSK_IPAR_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 *MSK_IPAR_LOG_CUT_SECOND_OPT* for the second and any subsequent optimizations.

Default 10

Accepted [0; +inf]

Example `MSK_putintparam(task, MSK_IPAR_LOG, 10)`

See also *MSK_IPAR_LOG_CUT_SECOND_OPT*

Groups *Output information, Logging*

MSK_IPAR_LOG_ANA_PRO

Controls amount of output from the problem analyzer.

Default 1

Accepted [0; +inf]

Example `MSK_putintparam(task, MSK_IPAR_LOG_ANA_PRO, 1)`

Groups *Analysis, Logging*

MSK_IPAR_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 `MSK_putintparam(task, MSK_IPAR_LOG_BI, 1)`

Groups *Basis identification, Output information, Logging*

MSK_IPAR_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 `MSK_putintparam(task, MSK_IPAR_LOG_BI_FREQ, 2500)`

Groups *Basis identification, Output information, Logging*

MSK_IPAR_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 `MSK_putintparam(task, MSK_IPAR_LOG_CHECK_CONVEXITY, 0)`

Groups *Data check*

MSK_IPAR_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 *MSK_IPAR_LOG* and *MSK_IPAR_LOG_SIM* are reduced by the value of this parameter for the second and any subsequent optimizations.

Default 1

Accepted [0; +inf]

Example `MSK_putintparam(task, MSK_IPAR_LOG_CUT_SECOND_OPT, 1)`

See also *MSK_IPAR_LOG*, *MSK_IPAR_LOG_INTPNT*, *MSK_IPAR_LOG_MIO*,
MSK_IPAR_LOG_SIM

Groups *Output information, Logging*

MSK_IPAR_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 `MSK_putintparam(task, MSK_IPAR_LOG_EXPAND, 0)`

Groups *Output information, Logging*

MSK_IPAR_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 `MSK_putintparam(task, MSK_IPAR_LOG_FEAS_REPAIR, 1)`

Groups *Output information, Logging*

MSK_IPAR_LOG_FILE

If turned on, then some log info is printed when a file is written or read.

Default 1

Accepted [0; +inf]

Example `MSK_putintparam(task, MSK_IPAR_LOG_FILE, 1)`

Groups *Data input/output, Output information, Logging*

MSK_IPAR_LOG_INCLUDE_SUMMARY

If on, then the solution summary will be printed by *MSK_optimize*, so a separate call to *MSK_solutionsummary* is not necessary.

Default *OFF*

Accepted *ON*, *OFF* (see *MSKonoffkeye*)

Example `MSK_putintparam(task, MSK_IPAR_LOG_INCLUDE_SUMMARY, MSK_OFF)`

Groups *Output information, Logging*

MSK_IPAR_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 `MSK_putintparam(task, MSK_IPAR_LOG_INFEAS_ANA, 1)`

Groups *Infeasibility report, Output information, Logging*

MSK_IPAR_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 `MSK_putintparam(task, MSK_IPAR_LOG_INTPNT, 1)`

Groups *Interior-point method, Output information, Logging*

MSK_IPAR_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 *MSK_onoffkeye*)

Example `MSK_putintparam(task, MSK_IPAR_LOG_LOCAL_INFO, MSK_ON)`

Groups *Output information, Logging*

MSK_IPAR_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 `MSK_putintparam(task, MSK_IPAR_LOG_MIO, 4)`

Groups *Mixed-integer optimization, Output information, Logging*

MSK_IPAR_LOG_MIO_FREQ

Controls how frequent the mixed-integer optimizer prints the log line. It will print line every time *MSK_IPAR_LOG_MIO_FREQ* relaxations have been solved.

Default 10

Accepted [-inf; +inf]

Example `MSK_putintparam(task, MSK_IPAR_LOG_MIO_FREQ, 10)`

Groups *Mixed-integer optimization, Output information, Logging*

MSK_IPAR_LOG_ORDER

If turned on, then factor lines are added to the log.

Default 1

Accepted [0; +inf]

Example `MSK_putintparam(task, MSK_IPAR_LOG_ORDER, 1)`

Groups *Output information, Logging*

MSK_IPAR_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 `MSK_putintparam(task, MSK_IPAR_LOG_PRESOLVE, 1)`

Groups *Logging*

MSK_IPAR_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 `MSK_putintparam(task, MSK_IPAR_LOG_RESPONSE, 0)`

Groups *Output information, Logging*

MSK_IPAR_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 `MSK_putintparam(task, MSK_IPAR_LOG_SENSITIVITY, 1)`

Groups *Output information, Logging*

MSK_IPAR_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 `MSK_putintparam(task, MSK_IPAR_LOG_SENSITIVITY_OPT, 0)`

Groups *Output information, Logging*

MSK_IPAR_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 `MSK_putintparam(task, MSK_IPAR_LOG_SIM, 4)`

Groups *Simplex optimizer, Output information, Logging*

MSK_IPAR_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 `MSK_putintparam(task, MSK_IPAR_LOG_SIM_FREQ, 1000)`

Groups *Simplex optimizer, Output information, Logging*

MSK_IPAR_LOG_SIM_MINOR

Currently not in use.

Default 1

Accepted [0; +inf]

Example `MSK_putintparam(task, MSK_IPAR_LOG_SIM_MINOR, 1)`

Groups *Simplex optimizer, Output information*

MSK_IPAR_LOG_STORAGE

When turned on, **MOSEK** prints messages regarding the storage usage and allocation.

Default 0

Accepted [0; +inf]

Example MSK_putintparam(task, MSK_IPAR_LOG_STORAGE, 0)

Groups *Output information, Overall system, Logging*

MSK_IPAR_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 MSK_putintparam(task, MSK_IPAR_MAX_NUM_WARNINGS, 10)

Groups *Output information*

MSK_IPAR_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 *MSKbranchdire*)

Example MSK_putintparam(task, MSK_IPAR_MIO_BRANCH_DIR, MSK_BRANCH_DIR_FREE)

Groups *Mixed-integer optimization*

MSK_IPAR_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 *MSKonoffkeye*)

Example MSK_putintparam(task, MSK_IPAR_MIO_CONIC_OUTER_APPROXIMATION, MSK_OFF)

Groups *Mixed-integer optimization*

MSK_IPAR_MIO_CUT_CLIQUE

Controls whether clique cuts should be generated.

Default *ON*

Accepted *ON, OFF* (see *MSKonoffkeye*)

Example MSK_putintparam(task, MSK_IPAR_MIO_CUT_CLIQUE, MSK_ON)

Groups *Mixed-integer optimization*

MSK_IPAR_MIO_CUT_CMIR

Controls whether mixed integer rounding cuts should be generated.

Default *ON*

Accepted *ON, OFF* (see *MSKonoffkeye*)

Example MSK_putintparam(task, MSK_IPAR_MIO_CUT_CMIR, MSK_ON)

Groups *Mixed-integer optimization*

MSK_IPAR_MIO_CUT_GMI

Controls whether GMI cuts should be generated.

Default *ON*

Accepted *ON, OFF* (see *MSKonoffkeye*)

Example MSK_putintparam(task, MSK_IPAR_MIO_CUT_GMI, MSK_ON)

Groups *Mixed-integer optimization*

MSK_IPAR_MIO_CUT_IMPLIED_BOUND

Controls whether implied bound cuts should be generated.

Default *OFF*

Accepted *ON, OFF* (see *MSKOnoffkeye*)

Example `MSK_putintparam(task, MSK_IPAR_MIO_CUT_IMPLIED_BOUND, MSK_OFF)`

Groups *Mixed-integer optimization*

MSK_IPAR_MIO_CUT_KNAPSACK_COVER

Controls whether knapsack cover cuts should be generated.

Default *OFF*

Accepted *ON, OFF* (see *MSKOnoffkeye*)

Example `MSK_putintparam(task, MSK_IPAR_MIO_CUT_KNAPSACK_COVER, MSK_OFF)`

Groups *Mixed-integer optimization*

MSK_IPAR_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 `MSK_putintparam(task, MSK_IPAR_MIO_CUT_SELECTION_LEVEL, -1)`

Groups *Mixed-integer optimization*

MSK_IPAR_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 `MSK_putintparam(task, MSK_IPAR_MIO_FEASPUMP_LEVEL, -1)`

Groups *Mixed-integer optimization*

MSK_IPAR_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 `MSK_putintparam(task, MSK_IPAR_MIO_HEURISTIC_LEVEL, -1)`

Groups *Mixed-integer optimization*

MSK_IPAR_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 `MSK_putintparam(task, MSK_IPAR_MIO_MAX_NUM_BRANCHES, -1)`

Groups *Mixed-integer optimization, Termination criteria*

MSK_IPAR_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 `MSK_putintparam(task, MSK_IPAR_MIO_MAX_NUM_RELAXS, -1)`

Groups *Mixed-integer optimization*

MSK_IPAR_MIO_MAX_NUM_ROOT_CUT_ROUNDS

Maximum number of cut separation rounds at the root node.

Default 100

Accepted [0; +inf]

Example `MSK_putintparam(task, MSK_IPAR_MIO_MAX_NUM_ROOT_CUT_ROUNDS, 100)`

Groups *Mixed-integer optimization, Termination criteria*

MSK_IPAR_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 `MSK_putintparam(task, MSK_IPAR_MIO_MAX_NUM_SOLUTIONS, -1)`

Groups *Mixed-integer optimization, Termination criteria*

MSK_IPAR_MIO_MODE

Controls whether the optimizer includes the integer restrictions when solving a (mixed) integer optimization problem.

Default *SATISFIED*

Accepted *IGNORED, SATISFIED* (see *MSKmiomodee*)

Example `MSK_putintparam(task, MSK_IPAR_MIO_MODE, MSK_MIO_MODE_SATISFIED)`

Groups *Overall solver*

MSK_IPAR_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 *MSKoptimizertypee*)

Example `MSK_putintparam(task, MSK_IPAR_MIO_NODE_OPTIMIZER, MSK_OPTIMIZER_FREE)`

Groups *Mixed-integer optimization*

MSK_IPAR_MIO_NODE_SELECTION

Controls the node selection strategy employed by the mixed-integer optimizer.

Default *FREE*

Accepted *FREE, FIRST, BEST, PSEUDO* (see *MSKmionodeseltypee*)

Example `MSK_putintparam(task, MSK_IPAR_MIO_NODE_SELECTION, MSK_MIO_NODE_SELECTION_FREE)`

Groups *Mixed-integer optimization*

MSK_IPAR_MIO_PERSPECTIVE_REFORMULATE

Enables or disables perspective reformulation in presolve.

Default *ON*

Accepted *ON, OFF* (see *MSKOnoffkeye*)

Example `MSK_putintparam(task, MSK_IPAR_MIO_PERSPECTIVE_REFORMULATE, MSK_ON)`

Groups *Mixed-integer optimization*

MSK_IPAR_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 `MSK_putintparam(task, MSK_IPAR_MIO_PROBING_LEVEL, -1)`

Groups *Mixed-integer optimization*

MSK_IPAR_MIO_PROPAGATE_OBJECTIVE_CONSTRAINT

Use objective domain propagation.

Default *OFF*

Accepted *ON, OFF* (see *MSKOnoffkeye*)

Example `MSK_putintparam(task, MSK_IPAR_MIO_PROPAGATE_OBJECTIVE_CONSTRAINT, MSK_OFF)`

Groups *Mixed-integer optimization*

MSK_IPAR_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 `MSK_putintparam(task, MSK_IPAR_MIO_RINS_MAX_NODES, -1)`

Groups *Mixed-integer optimization*

MSK_IPAR_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 *MSKoptimizertypee*)

Example `MSK_putintparam(task, MSK_IPAR_MIO_ROOT_OPTIMIZER, MSK_OPTIMIZER_FREE)`

Groups *Mixed-integer optimization*

MSK_IPAR_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 `MSK_putintparam(task, MSK_IPAR_MIO_ROOT_REPEAT_PREOLVE_LEVEL, -1)`

Groups *Mixed-integer optimization*

MSK_IPAR_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 `MSK_putintparam(task, MSK_IPAR_MIO_SEED, 42)`

Groups *Mixed-integer optimization*

MSK_IPAR_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 `MSK_putintparam(task, MSK_IPAR_MIO_VB_DETECTION_LEVEL, -1)`

Groups *Mixed-integer optimization*

MSK_IPAR_MT_SPINCOUNT

Set the number of iterations to spin before sleeping.

Default 0

Accepted [0; 1000000000]

Example `MSK_putintparam(task, MSK_IPAR_MT_SPINCOUNT, 0)`

Groups *Overall system*

MSK_IPAR_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 *MSK_IPAR_INTPNT_MULTI_THREAD*.

For the mixed-integer optimizer and interior-point linear optimizer there is no such restriction.

Default 0

Accepted [0; +inf]

Example `MSK_putintparam(task, MSK_IPAR_NUM_THREADS, 0)`

Groups *Overall system*

MSK_IPAR_OPF_WRITE_HEADER

Write a text header with date and **MOSEK** version in an OPF file.

Default *ON*

Accepted *ON*, *OFF* (see *MSKOnoffkeye*)

Example `MSK_putintparam(task, MSK_IPAR_OPF_WRITE_HEADER, MSK_ON)`

Groups *Data input/output*

MSK_IPAR_OPF_WRITE_HINTS

Write a hint section with problem dimensions in the beginning of an OPF file.

Default *ON*

Accepted *ON, OFF* (see *MSKonoffkeye*)

Example `MSK_putintparam(task, MSK_IPAR_OPF_WRITE_HINTS, MSK_ON)`

Groups *Data input/output*

MSK_IPAR_OPF_WRITE_LINE_LENGTH

Aim to keep lines in OPF files not much longer than this.

Default 80

Accepted `[0; +inf]`

Example `MSK_putintparam(task, MSK_IPAR_OPF_WRITE_LINE_LENGTH, 80)`

Groups *Data input/output*

MSK_IPAR_OPF_WRITE_PARAMETERS

Write a parameter section in an OPF file.

Default *OFF*

Accepted *ON, OFF* (see *MSKonoffkeye*)

Example `MSK_putintparam(task, MSK_IPAR_OPF_WRITE_PARAMETERS, MSK_OFF)`

Groups *Data input/output*

MSK_IPAR_OPF_WRITE_PROBLEM

Write objective, constraints, bounds etc. to an OPF file.

Default *ON*

Accepted *ON, OFF* (see *MSKonoffkeye*)

Example `MSK_putintparam(task, MSK_IPAR_OPF_WRITE_PROBLEM, MSK_ON)`

Groups *Data input/output*

MSK_IPAR_OPF_WRITE_SOL_BAS

If *MSK_IPAR_OPF_WRITE_SOLUTIONS* is *MSK_ON* and a basic solution is defined, include the basic solution in OPF files.

Default *ON*

Accepted *ON, OFF* (see *MSKonoffkeye*)

Example `MSK_putintparam(task, MSK_IPAR_OPF_WRITE_SOL_BAS, MSK_ON)`

Groups *Data input/output*

MSK_IPAR_OPF_WRITE_SOL_ITG

If *MSK_IPAR_OPF_WRITE_SOLUTIONS* is *MSK_ON* and an integer solution is defined, write the integer solution in OPF files.

Default *ON*

Accepted *ON, OFF* (see *MSKonoffkeye*)

Example `MSK_putintparam(task, MSK_IPAR_OPF_WRITE_SOL_ITG, MSK_ON)`

Groups *Data input/output*

MSK_IPAR_OPF_WRITE_SOL_ITR

If *MSK_IPAR_OPF_WRITE_SOLUTIONS* is *MSK_ON* and an interior solution is defined, write the interior solution in OPF files.

Default *ON*

Accepted *ON, OFF* (see *MSKonoffkeye*)

Example `MSK_putintparam(task, MSK_IPAR_OPF_WRITE_SOL_ITR, MSK_ON)`

Groups *Data input/output*

MSK_IPAR_OPF_WRITE_SOLUTIONS

Enable inclusion of solutions in the OPF files.

Default *OFF*

Accepted *ON, OFF* (see *MSKonoffkeye*)

Example `MSK_putintparam(task, MSK_IPAR_OPF_WRITE_SOLUTIONS, MSK_OFF)`

Groups *Data input/output*

MSK_IPAR_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 *MSKoptimizertypee*)

Example `MSK_putintparam(task, MSK_IPAR_OPTIMIZER, MSK_OPTIMIZER_FREE)`

Groups *Overall solver*

MSK_IPAR_PARAM_READ_CASE_NAME

If turned on, then names in the parameter file are case sensitive.

Default *ON*

Accepted *ON, OFF* (see *MSKonoffkeye*)

Example `MSK_putintparam(task, MSK_IPAR_PARAM_READ_CASE_NAME, MSK_ON)`

Groups *Data input/output*

MSK_IPAR_PARAM_READ_IGN_ERROR

If turned on, then errors in parameter settings is ignored.

Default *OFF*

Accepted *ON, OFF* (see *MSKonoffkeye*)

Example `MSK_putintparam(task, MSK_IPAR_PARAM_READ_IGN_ERROR, MSK_OFF)`

Groups *Data input/output*

MSK_IPAR_PREOLVE_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 `MSK_putintparam(task, MSK_IPAR_PREOLVE_ELIMINATOR_MAX_FILL, -1)`

Groups *Presolve*

MSK_IPAR_PREOLVE_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 `MSK_putintparam(task, MSK_IPAR_PREOLVE_ELIMINATOR_MAX_NUM_TRIES, -1)`

Groups *Presolve*

MSK_IPAR_PREOLVE_LEVEL

Currently not used.

Default *-1*

Accepted *[-inf; +inf]*

Example `MSK_putintparam(task, MSK_IPAR_PREOLVE_LEVEL, -1)`

Groups *Overall solver, Presolve*

MSK_IPAR_PREOLVE_LINDEP_ABS_WORK_TRH

Controls linear dependency check in presolve. The linear dependency check is potentially computationally expensive.

Default 100

Accepted [-inf; +inf]

Example MSK_putintparam(task, MSK_IPAR_PREOLVE_LINDEP_ABS_WORK_TRH, 100)

Groups *Presolve*

MSK_IPAR_PREOLVE_LINDEP_REL_WORK_TRH

Controls linear dependency check in presolve. The linear dependency check is potentially computationally expensive.

Default 100

Accepted [-inf; +inf]

Example MSK_putintparam(task, MSK_IPAR_PREOLVE_LINDEP_REL_WORK_TRH, 100)

Groups *Presolve*

MSK_IPAR_PREOLVE_LINDEP_USE

Controls whether the linear constraints are checked for linear dependencies.

Default *ON*

Accepted *ON, OFF* (see *MSK_onoffkeye*)

Example MSK_putintparam(task, MSK_IPAR_PREOLVE_LINDEP_USE, MSK_ON)

Groups *Presolve*

MSK_IPAR_PREOLVE_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 MSK_putintparam(task, MSK_IPAR_PREOLVE_MAX_NUM_PASS, -1)

Groups *Presolve*

MSK_IPAR_PREOLVE_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 MSK_putintparam(task, MSK_IPAR_PREOLVE_MAX_NUM_REDUCTIONS, -1)

Groups *Overall solver, Presolve*

MSK_IPAR_PREOLVE_USE

Controls whether the presolve is applied to a problem before it is optimized.

Default *FREE*

Accepted *OFF, ON, FREE* (see *MSK_presolvemodee*)

Example MSK_putintparam(task, MSK_IPAR_PREOLVE_USE, MSK_PREOLVE_MODE_FREE)

Groups *Overall solver, Presolve*

MSK_IPAR_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 *MSKoptimizertypee*)

Example `MSK_putintparam(task, MSK_IPAR_PRIMAL_REPAIR_OPTIMIZER, MSK_OPTIMIZER_FREE)`

Groups *Overall solver*

MSK_IPAR_PTF_WRITE_TRANSFORM

If *MSK_IPAR_PTF_WRITE_TRANSFORM* is *MSK_ON*, constraint blocks with identifiable conic slacks are transformed into conic constraints and the slacks are eliminated.

Default *ON*

Accepted *ON, OFF* (see *MSKonoffkeye*)

Example `MSK_putintparam(task, MSK_IPAR_PTF_WRITE_TRANSFORM, MSK_ON)`

Groups *Data input/output*

MSK_IPAR_READ_DEBUG

Turns on additional debugging information when reading files.

Default *OFF*

Accepted *ON, OFF* (see *MSKonoffkeye*)

Example `MSK_putintparam(task, MSK_IPAR_READ_DEBUG, MSK_OFF)`

Groups *Data input/output*

MSK_IPAR_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 `MSK_putintparam(task, MSK_IPAR_READ_KEEP_FREE_CON, MSK_OFF)`

Groups *Data input/output*

MSK_IPAR_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 *MSKonoffkeye*)

Example `MSK_putintparam(task, MSK_IPAR_READ_LP_DROP_NEW_VARS_IN_BOU, MSK_OFF)`

Groups *Data input/output*

MSK_IPAR_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 *MSKonoffkeye*)

Example `MSK_putintparam(task, MSK_IPAR_READ_LP_QUOTED_NAMES, MSK_ON)`

Groups *Data input/output*

MSK_IPAR_READ_MPS_FORMAT

Controls how strictly the MPS file reader interprets the MPS format.

Default *FREE*

Accepted *STRICT, RELAXED, FREE, CPLEX* (see *MSKmpsformat*)

Example `MSK_putintparam(task, MSK_IPAR_READ_MPS_FORMAT, MSK_MPS_FORMAT_FREE)`

Groups *Data input/output*

MSK_IPAR_READ_MPS_WIDTH

Controls the maximal number of characters allowed in one line of the MPS file.

Default 1024

Accepted [80; +inf]

Example MSK_putintparam(task, MSK_IPAR_READ_MPS_WIDTH, 1024)

Groups *Data input/output*

MSK_IPAR_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 *MSKonoffkeye*)

Example MSK_putintparam(task, MSK_IPAR_READ_TASK_IGNORE_PARAM, MSK_OFF)

Groups *Data input/output*

MSK_IPAR_REMOVE_UNUSED_SOLUTIONS

Removes unused solutions before the optimization is performed.

Default *OFF*

Accepted *ON*, *OFF* (see *MSKonoffkeye*)

Example MSK_putintparam(task, MSK_IPAR_REMOVE_UNUSED_SOLUTIONS, MSK_OFF)

Groups *Overall system*

MSK_IPAR_SENSITIVITY_ALL

If set to *MSK_ON*, then *MSK_sensitivityreport* analyzes all bounds and variables instead of reading a specification from the file.

Default *OFF*

Accepted *ON*, *OFF* (see *MSKonoffkeye*)

Example MSK_putintparam(task, MSK_IPAR_SENSITIVITY_ALL, MSK_OFF)

Groups *Overall solver*

MSK_IPAR_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 *MSKoptimizertypee*)

Example MSK_putintparam(task, MSK_IPAR_SENSITIVITY_OPTIMIZER, MSK_OPTIMIZER_FREE_SIMPLEX)

Groups *Overall solver*, *Simplex optimizer*

MSK_IPAR_SENSITIVITY_TYPE

Controls which type of sensitivity analysis is to be performed.

Default *BASIS*

Accepted *BASIS* (see *MSKsensitivitytypee*)

Example MSK_putintparam(task, MSK_IPAR_SENSITIVITY_TYPE, MSK_SENSITIVITY_TYPE_BASIS)

Groups *Overall solver*

MSK_IPAR_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 *MSK_{onoffkeye}*)

Example `MSK_putintparam(task, MSK_IPAR_SIM_BASIS_FACTOR_USE, MSK_ON)`

Groups *Simplex optimizer*

MSK_IPAR_SIM_DEGEN

Controls how aggressively degeneration is handled.

Default *FREE*

Accepted *NONE, FREE, AGGRESSIVE, MODERATE, MINIMUM* (see *MSK_{simdegene}*)

Example `MSK_putintparam(task, MSK_IPAR_SIM_DEGEN, MSK_SIM_DEGEN_FREE)`

Groups *Simplex optimizer*

MSK_IPAR_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 `MSK_putintparam(task, MSK_IPAR_SIM_DUAL_CRASH, 90)`

Groups *Dual simplex*

MSK_IPAR_SIM_DUAL_PHASEONE_METHOD

An experimental feature.

Default 0

Accepted $[0; 10]$

Example `MSK_putintparam(task, MSK_IPAR_SIM_DUAL_PHASEONE_METHOD, 0)`

Groups *Simplex optimizer*

MSK_IPAR_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 `MSK_putintparam(task, MSK_IPAR_SIM_DUAL_RESTRICT_SELECTION, 50)`

Groups *Dual simplex*

MSK_IPAR_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 *MSK_{simseltypee}*)

Example `MSK_putintparam(task, MSK_IPAR_SIM_DUAL_SELECTION, MSK_SIM_SELECTION_FREE)`

Groups *Dual simplex*

MSK_IPAR_SIM_EXPLOIT_DUPVEC

Controls if the simplex optimizers are allowed to exploit duplicated columns.

Default *OFF*

Accepted *ON, OFF, FREE* (see *MSK_{simdupvece}*)

Example `MSK_putintparam(task, MSK_IPAR_SIM_EXPLOIT_DUPVEC, MSK_SIM_EXPLOIT_DUPVEC_OFF)`

Groups *Simplex optimizer*

MSK_IPAR_SIM_HOTSTART

Controls the type of hot-start that the simplex optimizer perform.

Default *FREE*

Accepted *NONE, FREE, STATUS_KEYS* (see *MSKsimhotstarte*)

Example `MSK_putintparam(task, MSK_IPAR_SIM_HOTSTART, MSK_SIM_HOTSTART_FREE)`

Groups *Simplex optimizer*

MSK_IPAR_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 `MSK_putintparam(task, MSK_IPAR_SIM_HOTSTART_LU, MSK_ON)`

Groups *Simplex optimizer*

MSK_IPAR_SIM_MAX_ITERATIONS

Maximum number of iterations that can be used by a simplex optimizer.

Default 10000000

Accepted [0; +inf]

Example `MSK_putintparam(task, MSK_IPAR_SIM_MAX_ITERATIONS, 10000000)`

Groups *Simplex optimizer, Termination criteria*

MSK_IPAR_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 `MSK_putintparam(task, MSK_IPAR_SIM_MAX_NUM_SETBACKS, 250)`

Groups *Simplex optimizer*

MSK_IPAR_SIM_NON_SINGULAR

Controls if the simplex optimizer ensures a non-singular basis, if possible.

Default *ON*

Accepted *ON, OFF* (see *MSKonoffkeye*)

Example `MSK_putintparam(task, MSK_IPAR_SIM_NON_SINGULAR, MSK_ON)`

Groups *Simplex optimizer*

MSK_IPAR_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 `MSK_putintparam(task, MSK_IPAR_SIM_PRIMAL_CRASH, 90)`

Groups *Primal simplex*

MSK_IPAR_SIM_PRIMAL_PHASEONE_METHOD

An experimental feature.

Default 0

Accepted [0; 10]

Example `MSK_putintparam(task, MSK_IPAR_SIM_PRIMAL_PHASEONE_METHOD, 0)`

Groups *Simplex optimizer*

MSK_IPAR_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 `MSK_putintparam(task, MSK_IPAR_SIM_PRIMAL_RESTRICT_SELECTION, 50)`

Groups *Primal simplex*

MSK_IPAR_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 *MSKsimseltypee*)

Example `MSK_putintparam(task, MSK_IPAR_SIM_PRIMAL_SELECTION, MSK_SIM_SELECTION_FREE)`

Groups *Primal simplex*

MSK_IPAR_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 `MSK_putintparam(task, MSK_IPAR_SIM_REFACTOR_FREQ, 0)`

Groups *Simplex optimizer*

MSK_IPAR_SIM_REFORMULATION

Controls if the simplex optimizers are allowed to reformulate the problem.

Default *OFF*

Accepted *ON, OFF, FREE, AGGRESSIVE* (see *MSKsimreforme*)

Example `MSK_putintparam(task, MSK_IPAR_SIM_REFORMULATION, MSK_SIM_REFORMULATION_OFF)`

Groups *Simplex optimizer*

MSK_IPAR_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 *MSKnonoffkeye*)

Example `MSK_putintparam(task, MSK_IPAR_SIM_SAVE_LU, MSK_OFF)`

Groups *Simplex optimizer*

MSK_IPAR_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 *MSKscalingtypee*)

Example `MSK_putintparam(task, MSK_IPAR_SIM_SCALING, MSK_SCALING_FREE)`

Groups *Simplex optimizer*

MSK_IPAR_SIM_SCALING_METHOD

Controls how the problem is scaled before a simplex optimizer is used.

Default *POW2*

Accepted *POW2, FREE* (see *MSKscalingmethode*)

Example `MSK_putintparam(task, MSK_IPAR_SIM_SCALING_METHOD, MSK_SCALING_METHOD_POW2)`

Groups *Simplex optimizer*

MSK_IPAR_SIM_SEED

Sets the random seed used for randomization in the simplex optimizers.

Default 23456

Accepted [0; 32749]

Example `MSK_putintparam(task, MSK_IPAR_SIM_SEED, 23456)`

Groups *Simplex optimizer*

MSK_IPAR_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 *MSKsolveforme*)

Example `MSK_putintparam(task, MSK_IPAR_SIM_SOLVE_FORM, MSK_SOLVE_FREE)`

Groups *Simplex optimizer*

MSK_IPAR_SIM_STABILITY_PRIORITY

Controls how high priority the numerical stability should be given.

Default 50

Accepted [0; 100]

Example `MSK_putintparam(task, MSK_IPAR_SIM_STABILITY_PRIORITY, 50)`

Groups *Simplex optimizer*

MSK_IPAR_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 *MSKonoffkeye*)

Example `MSK_putintparam(task, MSK_IPAR_SIM_SWITCH_OPTIMIZER, MSK_OFF)`

Groups *Simplex optimizer*

MSK_IPAR_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 *MSKonoffkeye*)

Example `MSK_putintparam(task, MSK_IPAR_SOL_FILTER_KEEP_BASIC, MSK_OFF)`

Groups *Solution input/output*

MSK_IPAR_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 *MSKOnoffkeye*)

Example `MSK_putintparam(task, MSK_IPAR_SOL_FILTER_KEEP_RANGED, MSK_OFF)`

Groups *Solution input/output*

MSK_IPAR_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 contains blanks.

Default *-1*

Accepted *[-inf; +inf]*

Example `MSK_putintparam(task, MSK_IPAR_SOL_READ_NAME_WIDTH, -1)`

Groups *Data input/output, Solution input/output*

MSK_IPAR_SOL_READ_WIDTH

Controls the maximal acceptable width of line in the solutions when read by **MOSEK**.

Default *1024*

Accepted *[80; +inf]*

Example `MSK_putintparam(task, MSK_IPAR_SOL_READ_WIDTH, 1024)`

Groups *Data input/output, Solution input/output*

MSK_IPAR_SOLUTION_CALLBACK

Indicates whether solution callbacks will be performed during the optimization.

Default *OFF*

Accepted *ON, OFF* (see *MSKOnoffkeye*)

Example `MSK_putintparam(task, MSK_IPAR_SOLUTION_CALLBACK, MSK_OFF)`

Groups *Progress callback, Overall solver*

MSK_IPAR_TIMING_LEVEL

Controls the amount of timing performed inside **MOSEK**.

Default *1*

Accepted *[0; +inf]*

Example `MSK_putintparam(task, MSK_IPAR_TIMING_LEVEL, 1)`

Groups *Overall system*

MSK_IPAR_WRITE_BAS_CONSTRAINTS

Controls whether the constraint section is written to the basic solution file.

Default *ON*

Accepted *ON, OFF* (see *MSKOnoffkeye*)

Example `MSK_putintparam(task, MSK_IPAR_WRITE_BAS_CONSTRAINTS, MSK_ON)`

Groups *Data input/output, Solution input/output*

MSK_IPAR_WRITE_BAS_HEAD

Controls whether the header section is written to the basic solution file.

Default *ON*

Accepted *ON, OFF* (see *MSKOnoffkeye*)

Example `MSK_putintparam(task, MSK_IPAR_WRITE_BAS_HEAD, MSK_ON)`

Groups *Data input/output, Solution input/output*

MSK_IPAR_WRITE_BAS_VARIABLES

Controls whether the variables section is written to the basic solution file.

Default *ON*

Accepted *ON, OFF* (see *MSKOnoffkeye*)

Example `MSK_putintparam(task, MSK_IPAR_WRITE_BAS_VARIABLES, MSK_ON)`

Groups *Data input/output, Solution input/output*

MSK_IPAR_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 `MSK_putintparam(task, MSK_IPAR_WRITE_COMPRESSION, 9)`

Groups *Data input/output*

MSK_IPAR_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 *MSKonoffkeye*)

Example `MSK_putintparam(task, MSK_IPAR_WRITE_DATA_PARAM, MSK_OFF)`

Groups *Data input/output*

MSK_IPAR_WRITE_FREE_CON

Controls whether the free constraints are written to the data file.

Default *ON*

Accepted *ON, OFF* (see *MSKonoffkeye*)

Example `MSK_putintparam(task, MSK_IPAR_WRITE_FREE_CON, MSK_ON)`

Groups *Data input/output*

MSK_IPAR_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 *MSKonoffkeye*)

Example `MSK_putintparam(task, MSK_IPAR_WRITE_GENERIC_NAMES, MSK_OFF)`

Groups *Data input/output*

MSK_IPAR_WRITE_GENERIC_NAMES_IO

Index origin used in generic names.

Default 1

Accepted [0; +inf]

Example `MSK_putintparam(task, MSK_IPAR_WRITE_GENERIC_NAMES_IO, 1)`

Groups *Data input/output*

MSK_IPAR_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 `MSK_putintparam(task, MSK_IPAR_WRITE_IGNORE_INCOMPATIBLE_ITEMS, MSK_OFF)`

Groups *Data input/output*

MSK_IPAR_WRITE_INT_CONSTRAINTS

Controls whether the constraint section is written to the integer solution file.

Default *ON*

Accepted *ON, OFF* (see *MSKOnoffkeye*)

Example `MSK_putintparam(task, MSK_IPAR_WRITE_INT_CONSTRAINTS, MSK_ON)`

Groups *Data input/output, Solution input/output*

MSK_IPAR_WRITE_INT_HEAD

Controls whether the header section is written to the integer solution file.

Default *ON*

Accepted *ON, OFF* (see *MSKOnoffkeye*)

Example `MSK_putintparam(task, MSK_IPAR_WRITE_INT_HEAD, MSK_ON)`

Groups *Data input/output, Solution input/output*

MSK_IPAR_WRITE_INT_VARIABLES

Controls whether the variables section is written to the integer solution file.

Default *ON*

Accepted *ON, OFF* (see *MSKOnoffkeye*)

Example `MSK_putintparam(task, MSK_IPAR_WRITE_INT_VARIABLES, MSK_ON)`

Groups *Data input/output, Solution input/output*

MSK_IPAR_WRITE_LP_FULL_OBJ

Write all variables, including the ones with 0-coefficients, in the objective.

Default *ON*

Accepted *ON, OFF* (see *MSKOnoffkeye*)

Example `MSK_putintparam(task, MSK_IPAR_WRITE_LP_FULL_OBJ, MSK_ON)`

Groups *Data input/output*

MSK_IPAR_WRITE_LP_LINE_WIDTH

Maximum width of line in an LP file written by **MOSEK**.

Default 80

Accepted [40; +inf]

Example `MSK_putintparam(task, MSK_IPAR_WRITE_LP_LINE_WIDTH, 80)`

Groups *Data input/output*

MSK_IPAR_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 *MSKOnoffkeye*)

Example `MSK_putintparam(task, MSK_IPAR_WRITE_LP_QUOTED_NAMES, MSK_ON)`

Groups *Data input/output*

MSK_IPAR_WRITE_LP_STRICT_FORMAT

Controls whether LP output files satisfy the LP format strictly.

Default *OFF*

Accepted *ON, OFF* (see *MSKOnoffkeye*)

Example `MSK_putintparam(task, MSK_IPAR_WRITE_LP_STRICT_FORMAT, MSK_OFF)`

Groups *Data input/output*

MSK_IPAR_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 `MSK_putintparam(task, MSK_IPAR_WRITE_LP_TERMS_PER_LINE, 10)`

Groups *Data input/output*

MSK_IPAR_WRITE_MPS_FORMAT

Controls in which format the MPS is written.

Default *FREE*

Accepted *STRICT, RELAXED, FREE, CPLEX* (see *MSKmpsformat*)

Example `MSK_putintparam(task, MSK_IPAR_WRITE_MPS_FORMAT, MSK_MPS_FORMAT_FREE)`

Groups *Data input/output*

MSK_IPAR_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 *MSKonoffkeye*)

Example `MSK_putintparam(task, MSK_IPAR_WRITE_MPS_INT, MSK_ON)`

Groups *Data input/output*

MSK_IPAR_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 `MSK_putintparam(task, MSK_IPAR_WRITE_PRECISION, 15)`

Groups *Data input/output*

MSK_IPAR_WRITE_SOL_BARVARIABLES

Controls whether the symmetric matrix variables section is written to the solution file.

Default *ON*

Accepted *ON, OFF* (see *MSKonoffkeye*)

Example `MSK_putintparam(task, MSK_IPAR_WRITE_SOL_BARVARIABLES, MSK_ON)`

Groups *Data input/output, Solution input/output*

MSK_IPAR_WRITE_SOL_CONSTRAINTS

Controls whether the constraint section is written to the solution file.

Default *ON*

Accepted *ON, OFF* (see *MSKonoffkeye*)

Example `MSK_putintparam(task, MSK_IPAR_WRITE_SOL_CONSTRAINTS, MSK_ON)`

Groups *Data input/output, Solution input/output*

MSK_IPAR_WRITE_SOL_HEAD

Controls whether the header section is written to the solution file.

Default *ON*

Accepted *ON, OFF* (see *MSKonoffkeye*)

Example `MSK_putintparam(task, MSK_IPAR_WRITE_SOL_HEAD, MSK_ON)`

Groups *Data input/output, Solution input/output*

MSK_IPAR_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 *MSKonoffkeye*)

Example `MSK_putintparam(task, MSK_IPAR_WRITE_SOL_IGNORE_INVALID_NAMES, MSK_OFF)`

Groups *Data input/output, Solution input/output*

MSK_IPAR_WRITE_SOL_VARIABLES

Controls whether the variables section is written to the solution file.

Default *ON*

Accepted *ON, OFF* (see *MSKonoffkeye*)

Example `MSK_putintparam(task, MSK_IPAR_WRITE_SOL_VARIABLES, MSK_ON)`

Groups *Data input/output, Solution input/output*

MSK_IPAR_WRITE_TASK_INC_SOL

Controls whether the solutions are stored in the task file too.

Default *ON*

Accepted *ON, OFF* (see *MSKonoffkeye*)

Example `MSK_putintparam(task, MSK_IPAR_WRITE_TASK_INC_SOL, MSK_ON)`

Groups *Data input/output*

MSK_IPAR_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 *MSKxmlwriteroutputtypee*)

Example `MSK_putintparam(task, MSK_IPAR_WRITE_XML_MODE, MSK_WRITE_XML_MODE_ROW)`

Groups *Data input/output*

15.5.3 String parameters

MSKsparame

The enumeration type containing all string parameters.

MSK_SPAR_BAS_SOL_FILE_NAME

Name of the bas solution file.

Accepted Any valid file name.

Example `MSK_putstrparam(task, MSK_SPAR_BAS_SOL_FILE_NAME, "somevalue")`

Groups *Data input/output, Solution input/output*

MSK_SPAR_DATA_FILE_NAME

Data are read and written to this file.

Accepted Any valid file name.

Example `MSK_putstrparam(task, MSK_SPAR_DATA_FILE_NAME, "somevalue")`

Groups *Data input/output*

MSK_SPAR_DEBUG_FILE_NAME

MOSEK debug file.

Accepted Any valid file name.

Example `MSK_putstrparam(task, MSK_SPAR_DEBUG_FILE_NAME, "somevalue")`

Groups *Data input/output*

MSK_SPAR_INT_SOL_FILE_NAME

Name of the int solution file.

Accepted Any valid file name.

Example `MSK_putstrparam(task, MSK_SPAR_INT_SOL_FILE_NAME, "somevalue")`

Groups *Data input/output, Solution input/output*

MSK_SPAR_ITR_SOL_FILE_NAME

Name of the itr solution file.

Accepted Any valid file name.

Example MSK_putstrparam(task, MSK_SPAR_ITR_SOL_FILE_NAME, "somevalue")

Groups *Data input/output, Solution input/output*

MSK_SPAR_MIO_DEBUG_STRING

For internal debugging purposes.

Accepted Any valid string.

Example MSK_putstrparam(task, MSK_SPAR_MIO_DEBUG_STRING, "somevalue")

Groups *Data input/output*

MSK_SPAR_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 MSK_putstrparam(task, MSK_SPAR_PARAM_COMMENT_SIGN, "%")

Groups *Data input/output*

MSK_SPAR_PARAM_READ_FILE_NAME

Modifications to the parameter database is read from this file.

Accepted Any valid file name.

Example MSK_putstrparam(task, MSK_SPAR_PARAM_READ_FILE_NAME, "somevalue")

Groups *Data input/output*

MSK_SPAR_PARAM_WRITE_FILE_NAME

The parameter database is written to this file.

Accepted Any valid file name.

Example MSK_putstrparam(task, MSK_SPAR_PARAM_WRITE_FILE_NAME, "somevalue")

Groups *Data input/output*

MSK_SPAR_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 MSK_putstrparam(task, MSK_SPAR_READ_MPS_BOU_NAME, "somevalue")

Groups *Data input/output*

MSK_SPAR_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 MSK_putstrparam(task, MSK_SPAR_READ_MPS_OBJ_NAME, "somevalue")

Groups *Data input/output*

MSK_SPAR_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 MSK_putstrparam(task, MSK_SPAR_READ_MPS_RAN_NAME, "somevalue")

Groups *Data input/output*

MSK_SPAR_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 `MSK_putstrparam(task, MSK_SPAR_READ_MPS_RHS_NAME, "somevalue")`

Groups *Data input/output*

MSK_SPAR_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 `MSK_putstrparam(task, MSK_SPAR_REMOTE_ACCESS_TOKEN, "somevalue")`

Groups *Overall system*

MSK_SPAR_SENSITIVITY_FILE_NAME

If defined *MSK_sensitivityreport* reads this file as a sensitivity analysis data file specifying the type of analysis to be done.

Accepted Any valid string.

Example `MSK_putstrparam(task, MSK_SPAR_SENSITIVITY_FILE_NAME, "somevalue")`

Groups *Data input/output*

MSK_SPAR_SENSITIVITY_RES_FILE_NAME

If this is a nonempty string, then *MSK_sensitivityreport* writes results to this file.

Accepted Any valid string.

Example `MSK_putstrparam(task, MSK_SPAR_SENSITIVITY_RES_FILE_NAME, "somevalue")`

Groups *Data input/output*

MSK_SPAR_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 `MSK_putstrparam(task, MSK_SPAR_SOL_FILTER_XC_LOW, "somevalue")`

Groups *Data input/output, Solution input/output*

MSK_SPAR_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 `MSK_putstrparam(task, MSK_SPAR_SOL_FILTER_XC_UPR, "somevalue")`

Groups *Data input/output, Solution input/output*

MSK_SPAR_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 `MSK_putstrparam(task, MSK_SPAR_SOL_FILTER_XX_LOW, "somevalue")`

Groups *Data input/output, Solution input/output*

MSK_SPAR_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 `MSK_putstrparam(task, MSK_SPAR_SOL_FILTER_XX_UPR, "somevalue")`

Groups *Data input/output, Solution input/output*

MSK_SPAR_STAT_FILE_NAME

Statistics file name.

Accepted Any valid file name.

Example `MSK_putstrparam(task, MSK_SPAR_STAT_FILE_NAME, "somevalue")`

Groups *Data input/output*

MSK_SPAR_STAT_KEY

Key used when writing the summary file.

Accepted Any valid string.

Example `MSK_putstrparam(task, MSK_SPAR_STAT_KEY, "somevalue")`

Groups *Data input/output*

MSK_SPAR_STAT_NAME

Name used when writing the statistics file.

Accepted Any valid XML string.

Example `MSK_putstrparam(task, MSK_SPAR_STAT_NAME, "somevalue")`

Groups *Data input/output*

MSK_SPAR_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 `MSK_putstrparam(task, MSK_SPAR_WRITE_LP_GEN_VAR_NAME, "xmskgen")`

Groups *Data input/output*

15.6 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.

MSKrescodee

The enumeration type containing all response codes.

15.6.1 Termination

MSK_RES_OK (0)

No error occurred.

MSK_RES_TRM_MAX_ITERATIONS (10000)

The optimizer terminated at the maximum number of iterations.

MSK_RES_TRM_MAX_TIME (10001)

The optimizer terminated at the maximum amount of time.

MSK_RES_TRM_OBJECTIVE_RANGE (10002)

The optimizer terminated with an objective value outside the objective range.

MSK_RES_TRM_MIO_NUM_RELAXS (10008)

The mixed-integer optimizer terminated as the maximum number of relaxations was reached.

MSK_RES_TRM_MIO_NUM_BRANCHES (10009)

The mixed-integer optimizer terminated as the maximum number of branches was reached.

MSK_RES_TRM_NUM_MAX_NUM_INT_SOLUTIONS (10015)

The mixed-integer optimizer terminated as the maximum number of feasible solutions was reached.

MSK_RES_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.

MSK_RES_TRM_USER_CALLBACK (10007)

The optimizer terminated due to the return of the user-defined callback function.

MSK_RES_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.

MSK_RES_TRM_NUMERICAL_PROBLEM (10025)

The optimizer terminated due to numerical problems.

MSK_RES_TRM_INTERNAL (10030)

The optimizer terminated due to some internal reason. Please contact **MOSEK** support.

MSK_RES_TRM_INTERNAL_STOP (10031)

The optimizer terminated for internal reasons. Please contact **MOSEK** support.

15.6.2 Warnings

MSK_RES_WRN_OPEN_PARAM_FILE (50)

The parameter file could not be opened.

MSK_RES_WRN_LARGE_BOUND (51)

A numerically large bound value is specified.

MSK_RES_WRN_LARGE_LO_BOUND (52)

A numerically large lower bound value is specified.

MSK_RES_WRN_LARGE_UP_BOUND (53)

A numerically large upper bound value is specified.

MSK_RES_WRN_LARGE_CON_FX (54)

An equality constraint is fixed to a numerically large value. This can cause numerical problems.

MSK_RES_WRN_LARGE_CJ (57)

A numerically large value is specified for one c_j .

MSK_RES_WRN_LARGE_AIJ (62)

A numerically large value is specified for an $a_{i,j}$ element in A . The parameter *MSK_DPAR_DATA_TOL_AIJ_LARGE* controls when an $a_{i,j}$ is considered large.

MSK_RES_WRN_ZERO_AIJ (63)
One or more zero elements are specified in A.

MSK_RES_WRN_NAME_MAX_LEN (65)
A name is longer than the buffer that is supposed to hold it.

MSK_RES_WRN_SPAR_MAX_LEN (66)
A value for a string parameter is longer than the buffer that is supposed to hold it.

MSK_RES_WRN_MPS_SPLIT_RHS_VECTOR (70)
An RHS vector is split into several nonadjacent parts in an MPS file.

MSK_RES_WRN_MPS_SPLIT_RAN_VECTOR (71)
A RANGE vector is split into several nonadjacent parts in an MPS file.

MSK_RES_WRN_MPS_SPLIT_BOU_VECTOR (72)
A BOUNDS vector is split into several nonadjacent parts in an MPS file.

MSK_RES_WRN_LP_OLD_QUAD_FORMAT (80)
Missing $\sqrt{2}$ after quadratic expressions in bound or objective.

MSK_RES_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.

MSK_RES_WRN_NZ_IN_UPR_TRI (200)
Non-zero elements specified in the upper triangle of a matrix were ignored.

MSK_RES_WRN_DROPPED_NZ_QOBJ (201)
One or more non-zero elements were dropped in the Q matrix in the objective.

MSK_RES_WRN_IGNORE_INTEGER (250)
Ignored integer constraints.

MSK_RES_WRN_NO_GLOBAL_OPTIMIZER (251)
No global optimizer is available.

MSK_RES_WRN_MIO_INFEASIBLE_FINAL (270)
The final mixed-integer problem with all the integer variables fixed at their optimal values is infeasible.

MSK_RES_WRN_SOL_FILTER (300)
Invalid solution filter is specified.

MSK_RES_WRN_UNDEF_SOL_FILE_NAME (350)
Undefined name occurred in a solution.

MSK_RES_WRN_SOL_FILE_IGNORED_CON (351)
One or more lines in the constraint section were ignored when reading a solution file.

MSK_RES_WRN_SOL_FILE_IGNORED_VAR (352)
One or more lines in the variable section were ignored when reading a solution file.

MSK_RES_WRN_TOO_FEW_BASIS_VARS (400)
An incomplete basis has been specified. Too few basis variables are specified.

MSK_RES_WRN_TOO_MANY_BASIS_VARS (405)
A basis with too many variables has been specified.

MSK_RES_WRN_LICENSE_EXPIRE (500)
The license expires.

MSK_RES_WRN_LICENSE_SERVER (501)
The license server is not responding.

MSK_RES_WRN_EMPTY_NAME (502)
A variable or constraint name is empty. The output file may be invalid.

MSK_RES_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.

MSK_RES_WRN_LICENSE_FEATURE_EXPIRE (505)
The license expires.

MSK_RES_WRN_PARAM_NAME_DOUB (510)
The parameter name is not recognized as a double parameter.

MSK_RES_WRN_PARAM_NAME_INT (511)
The parameter name is not recognized as an integer parameter.

MSK_RES_WRN_PARAM_NAME_STR (512)
The parameter name is not recognized as a string parameter.

MSK_RES_WRN_PARAM_STR_VALUE (515)
The string is not recognized as a symbolic value for the parameter.

MSK_RES_WRN_PARAM_IGNORED_CMIO (516)
A parameter was ignored by the conic mixed integer optimizer.

MSK_RES_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.

MSK_RES_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.

MSK_RES_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.

MSK_RES_WRN_ELIMINATOR_SPACE (801)
The eliminator is skipped at least once due to lack of space.

MSK_RES_WRN_PRESOLVE_OUTOFSPACE (802)
The presolve is incomplete due to lack of space.

MSK_RES_WRN_WRITE_CHANGED_NAMES (803)
Some names were changed because they were invalid for the output file format.

MSK_RES_WRN_WRITE_DISCARDED_CFIX (804)
The fixed objective term could not be converted to a variable and was discarded in the output file.

MSK_RES_WRN_DUPLICATE_CONSTRAINT_NAMES (850)
Two constraint names are identical.

MSK_RES_WRN_DUPLICATE_VARIABLE_NAMES (851)
Two variable names are identical.

MSK_RES_WRN_DUPLICATE_BARVARIABLE_NAMES (852)
Two barvariable names are identical.

MSK_RES_WRN_DUPLICATE_CONE_NAMES (853)
Two cone names are identical.

MSK_RES_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 $-\text{inf}$.

MSK_RES_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.

MSK_RES_WRN_ANA_EMPTY_COLS (902)
This warning is issued by the problem analyzer, if columns, in which all coefficients are zero, are found.

MSK_RES_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.

MSK_RES_WRN_ANA_ALMOST_INT_BOUNDS (904)
This warning is issued by the problem analyzer if a constraint is bound nearly integral.

MSK_RES_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.

MSK_RES_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.

MSK_RES_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.

MSK_RES_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.

MSK_RES_WRN_NO_DUALIZER (950)

No automatic dualizer is available for the specified problem. The primal problem is solved.

MSK_RES_WRN_SYM_MAT_LARGE (960)

A numerically large value is specified for an $e_{i,j}$ element in E . The parameter *MSK_DPAR_DATA_SYM_MAT_TOL_LARGE* controls when an $e_{i,j}$ is considered large.

15.6.3 Errors

MSK_RES_ERR_LICENSE (1000)

Invalid license.

MSK_RES_ERR_LICENSE_EXPIRED (1001)

The license has expired.

MSK_RES_ERR_LICENSE_VERSION (1002)

The license is valid for another version of **MOSEK**.

MSK_RES_ERR_SIZE_LICENSE (1005)

The problem is bigger than the license.

MSK_RES_ERR_PROB_LICENSE (1006)

The software is not licensed to solve the problem.

MSK_RES_ERR_FILE_LICENSE (1007)

Invalid license file.

MSK_RES_ERR_MISSING_LICENSE_FILE (1008)

MOSEK cannot find license file or a token server. See the **MOSEK** licensing manual for details.

MSK_RES_ERR_SIZE_LICENSE_CON (1010)

The problem has too many constraints to be solved with the available license.

MSK_RES_ERR_SIZE_LICENSE_VAR (1011)

The problem has too many variables to be solved with the available license.

MSK_RES_ERR_SIZE_LICENSE_INTVAR (1012)

The problem contains too many integer variables to be solved with the available license.

MSK_RES_ERR_OPTIMIZER_LICENSE (1013)

The optimizer required is not licensed.

MSK_RES_ERR_FLEXLM (1014)

The FLEXlm license manager reported an error.

MSK_RES_ERR_LICENSE_SERVER (1015)

The license server is not responding.

MSK_RES_ERR_LICENSE_MAX (1016)

Maximum number of licenses is reached.

MSK_RES_ERR_LICENSE_MOSEKLM_DAEMON (1017)

The MOSEKLM license manager daemon is not up and running.

MSK_RES_ERR_LICENSE_FEATURE (1018)

A requested feature is not available in the license file(s). Most likely due to an incorrect license system setup.

MSK_RES_ERR_PLATFORM_NOT_LICENSED (1019)

A requested license feature is not available for the required platform.

MSK_RES_ERR_LICENSE_CANNOT_ALLOCATE (1020)

The license system cannot allocate the memory required.

MSK_RES_ERR_LICENSE_CANNOT_CONNECT (1021)

MOSEK cannot connect to the license server. Most likely the license server is not up and running.

MSK_RES_ERR_LICENSE_INVALID_HOSTID (1025)

The host ID specified in the license file does not match the host ID of the computer.

MSK_RES_ERR_LICENSE_SERVER_VERSION (1026)

The version specified in the checkout request is greater than the highest version number the daemon supports.

MSK_RES_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`.

MSK_RES_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.

MSK_RES_ERR_OLDER_DLL (1035)

The dynamic link library is older than the specified version.

MSK_RES_ERR_NEWER_DLL (1036)

The dynamic link library is newer than the specified version.

MSK_RES_ERR_LINK_FILE_DLL (1040)

A file cannot be linked to a stream in the DLL version.

MSK_RES_ERR_THREAD_MUTEX_INIT (1045)

Could not initialize a mutex.

MSK_RES_ERR_THREAD_MUTEX_LOCK (1046)

Could not lock a mutex.

MSK_RES_ERR_THREAD_MUTEX_UNLOCK (1047)

Could not unlock a mutex.

MSK_RES_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.

MSK_RES_ERR_THREAD_COND_INIT (1049)

Could not initialize a condition.

MSK_RES_ERR_UNKNOWN (1050)

Unknown error.

MSK_RES_ERR_SPACE (1051)

Out of space.

MSK_RES_ERR_FILE_OPEN (1052)

Error while opening a file.

MSK_RES_ERR_FILE_READ (1053)

File read error.

MSK_RES_ERR_FILE_WRITE (1054)

File write error.

MSK_RES_ERR_DATA_FILE_EXT (1055)

The data file format cannot be determined from the file name.

MSK_RES_ERR_INVALID_FILE_NAME (1056)

An invalid file name has been specified.

MSK_RES_ERR_INVALID_SOL_FILE_NAME (1057)

An invalid file name has been specified.

MSK_RES_ERR_END_OF_FILE (1059)

End of file reached.

MSK_RES_ERR_NULL_ENV (1060)

`env` is a **NULL** pointer.

MSK_RES_ERR_NULL_TASK (1061)

`task` is a **NULL** pointer.

MSK_RES_ERR_INVALID_STREAM (1062)

An invalid stream is referenced.

MSK_RES_ERR_NO_INIT_ENV (1063)

`env` is not initialized.

MSK_RES_ERR_INVALID_TASK (1064)

The `task` is invalid.

MSK_RES_ERR_NULL_POINTER (1065)

An argument to a function is unexpectedly a **NULL** pointer.

MSK_RES_ERR_LIVING_TASKS (1066)
 All tasks associated with an environment must be deleted before the environment is deleted. There are still some undeleted tasks.

MSK_RES_ERR_BLANK_NAME (1070)
 An all blank name has been specified.

MSK_RES_ERR_DUP_NAME (1071)
 The same name was used multiple times for the same problem item type.

MSK_RES_ERR_FORMAT_STRING (1072)
 The name format string is invalid.

MSK_RES_ERR_INVALID_OBJ_NAME (1075)
 An invalid objective name is specified.

MSK_RES_ERR_INVALID_CON_NAME (1076)
 An invalid constraint name is used.

MSK_RES_ERR_INVALID_VAR_NAME (1077)
 An invalid variable name is used.

MSK_RES_ERR_INVALID_CONE_NAME (1078)
 An invalid cone name is used.

MSK_RES_ERR_INVALID_BARVAR_NAME (1079)
 An invalid symmetric matrix variable name is used.

MSK_RES_ERR_SPACE_LEAKING (1080)
MOSEK is leaking memory. This can be due to either an incorrect use of **MOSEK** or a bug.

MSK_RES_ERR_SPACE_NO_INFO (1081)
 No available information about the space usage.

MSK_RES_ERR_READ_FORMAT (1090)
 The specified format cannot be read.

MSK_RES_ERR_MPS_FILE (1100)
 An error occurred while reading an MPS file.

MSK_RES_ERR_MPS_INV_FIELD (1101)
 A field in the MPS file is invalid. Probably it is too wide.

MSK_RES_ERR_MPS_INV_MARKER (1102)
 An invalid marker has been specified in the MPS file.

MSK_RES_ERR_MPS_NULL_CON_NAME (1103)
 An empty constraint name is used in an MPS file.

MSK_RES_ERR_MPS_NULL_VAR_NAME (1104)
 An empty variable name is used in an MPS file.

MSK_RES_ERR_MPS_UNDEF_CON_NAME (1105)
 An undefined constraint name occurred in an MPS file.

MSK_RES_ERR_MPS_UNDEF_VAR_NAME (1106)
 An undefined variable name occurred in an MPS file.

MSK_RES_ERR_MPS_INV_CON_KEY (1107)
 An invalid constraint key occurred in an MPS file.

MSK_RES_ERR_MPS_INV_BOUND_KEY (1108)
 An invalid bound key occurred in an MPS file.

MSK_RES_ERR_MPS_INV_SEC_NAME (1109)
 An invalid section name occurred in an MPS file.

MSK_RES_ERR_MPS_NO_OBJECTIVE (1110)
 No objective is defined in an MPS file.

MSK_RES_ERR_MPS_SPLITTED_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.

MSK_RES_ERR_MPS_MUL_CON_NAME (1112)
 A constraint name was specified multiple times in the ROWS section.

MSK_RES_ERR_MPS_MUL_QSEC (1113)
 Multiple QSECTIONS are specified for a constraint in the MPS data file.

MSK_RES_ERR_MPS_MUL_QOBJ (1114)
 The Q term in the objective is specified multiple times in the MPS data file.

MSK_RES_ERR_MPS_INV_SEC_ORDER (1115)
 The sections in the MPS data file are not in the correct order.

MSK_RES_ERR_MPS_MUL_CSEC (1116)
Multiple CSECTIONs are given the same name.

MSK_RES_ERR_MPS_CONE_TYPE (1117)
Invalid cone type specified in a CSECTION.

MSK_RES_ERR_MPS_CONE_OVERLAP (1118)
A variable is specified to be a member of several cones.

MSK_RES_ERR_MPS_CONE_REPEAT (1119)
A variable is repeated within the CSECTION.

MSK_RES_ERR_MPS_NON_SYMMETRIC_Q (1120)
A non symmetric matrix has been specified.

MSK_RES_ERR_MPS_DUPLICATE_Q_ELEMENT (1121)
Duplicate elements is specified in a Q matrix.

MSK_RES_ERR_MPS_INVALID_OBJSENSE (1122)
An invalid objective sense is specified.

MSK_RES_ERR_MPS_TAB_IN_FIELD2 (1125)
A tab char occurred in field 2.

MSK_RES_ERR_MPS_TAB_IN_FIELD3 (1126)
A tab char occurred in field 3.

MSK_RES_ERR_MPS_TAB_IN_FIELD5 (1127)
A tab char occurred in field 5.

MSK_RES_ERR_MPS_INVALID_OBJ_NAME (1128)
An invalid objective name is specified.

MSK_RES_ERR_LP_INCOMPATIBLE (1150)
The problem cannot be written to an LP formatted file.

MSK_RES_ERR_LP_EMPTY (1151)
The problem cannot be written to an LP formatted file.

MSK_RES_ERR_LP_DUP_SLACK_NAME (1152)
The name of the slack variable added to a ranged constraint already exists.

MSK_RES_ERR_WRITE_MPS_INVALID_NAME (1153)
An invalid name is created while writing an MPS file. Usually this will make the MPS file unreadable.

MSK_RES_ERR_LP_INVALID_VAR_NAME (1154)
A variable name is invalid when used in an LP formatted file.

MSK_RES_ERR_LP_FREE_CONSTRAINT (1155)
Free constraints cannot be written in LP file format.

MSK_RES_ERR_WRITE_OPF_INVALID_VAR_NAME (1156)
Empty variable names cannot be written to OPF files.

MSK_RES_ERR_LP_FILE_FORMAT (1157)
Syntax error in an LP file.

MSK_RES_ERR_WRITE_LP_FORMAT (1158)
Problem cannot be written as an LP file.

MSK_RES_ERR_READ_LP_MISSING_END_TAG (1159)
Syntax error in LP file. Possibly missing End tag.

MSK_RES_ERR_LP_FORMAT (1160)
Syntax error in an LP file.

MSK_RES_ERR_WRITE_LP_NON_UNIQUE_NAME (1161)
An auto-generated name is not unique.

MSK_RES_ERR_READ_LP_NONEXISTING_NAME (1162)
A variable never occurred in objective or constraints.

MSK_RES_ERR_LP_WRITE_CONIC_PROBLEM (1163)
The problem contains cones that cannot be written to an LP formatted file.

MSK_RES_ERR_LP_WRITE_GECO_PROBLEM (1164)
The problem contains general convex terms that cannot be written to an LP formatted file.

MSK_RES_ERR_WRITING_FILE (1166)
An error occurred while writing file

MSK_RES_ERR_PTF_FORMAT (1167)
Syntax error in an PTF file

MSK_RES_ERR_OPF_FORMAT (1168)
 Syntax error in an OPF file

MSK_RES_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.

MSK_RES_ERR_INVALID_NAME_IN_SOL_FILE (1170)
 An invalid name occurred in a solution file.

MSK_RES_ERR_LP_INVALID_CON_NAME (1171)
 A constraint name is invalid when used in an LP formatted file.

MSK_RES_ERR_OPF_PREMATURE_EOF (1172)
 Premature end of file in an OPF file.

MSK_RES_ERR_JSON_SYNTAX (1175)
 Syntax error in an JSON data

MSK_RES_ERR_JSON_STRING (1176)
 Error in JSON string.

MSK_RES_ERR_JSON_NUMBER_OVERFLOW (1177)
 Invalid number entry - wrong type or value overflow.

MSK_RES_ERR_JSON_FORMAT (1178)
 Error in an JSON Task file

MSK_RES_ERR_JSON_DATA (1179)
 Inconsistent data in JSON Task file

MSK_RES_ERR_JSON_MISSING_DATA (1180)
 Missing data section in JSON task file.

MSK_RES_ERR_ARGUMENT_LENNEQ (1197)
 Incorrect length of arguments.

MSK_RES_ERR_ARGUMENT_TYPE (1198)
 Incorrect argument type.

MSK_RES_ERR_NUM_ARGUMENTS (1199)
 Incorrect number of function arguments.

MSK_RES_ERR_IN_ARGUMENT (1200)
 A function argument is incorrect.

MSK_RES_ERR_ARGUMENT_DIMENSION (1201)
 A function argument is of incorrect dimension.

MSK_RES_ERR_SHAPE_IS_TOO_LARGE (1202)
 The size of the n-dimensional shape is too large.

MSK_RES_ERR_INDEX_IS_TOO_SMALL (1203)
 An index in an argument is too small.

MSK_RES_ERR_INDEX_IS_TOO_LARGE (1204)
 An index in an argument is too large.

MSK_RES_ERR_PARAM_NAME (1205)
 The parameter name is not correct.

MSK_RES_ERR_PARAM_NAME_DOUB (1206)
 The parameter name is not correct for a double parameter.

MSK_RES_ERR_PARAM_NAME_INT (1207)
 The parameter name is not correct for an integer parameter.

MSK_RES_ERR_PARAM_NAME_STR (1208)
 The parameter name is not correct for a string parameter.

MSK_RES_ERR_PARAM_INDEX (1210)
 Parameter index is out of range.

MSK_RES_ERR_PARAM_IS_TOO_LARGE (1215)
 The parameter value is too large.

MSK_RES_ERR_PARAM_IS_TOO_SMALL (1216)
 The parameter value is too small.

MSK_RES_ERR_PARAM_VALUE_STR (1217)
 The parameter value string is incorrect.

MSK_RES_ERR_PARAM_TYPE (1218)
 The parameter type is invalid.

MSK_RES_ERR_INF_DOU_INDEX (1219)
 A double information index is out of range for the specified type.

MSK_RES_ERR_INF_INT_INDEX (1220)
 An integer information index is out of range for the specified type.

MSK_RES_ERR_INDEX_ARR_IS_TOO_SMALL (1221)
 An index in an array argument is too small.

MSK_RES_ERR_INDEX_ARR_IS_TOO_LARGE (1222)
 An index in an array argument is too large.

MSK_RES_ERR_INF_LINT_INDEX (1225)
 A long integer information index is out of range for the specified type.

MSK_RES_ERR_ARG_IS_TOO_SMALL (1226)
 The value of a argument is too small.

MSK_RES_ERR_ARG_IS_TOO_LARGE (1227)
 The value of a argument is too large.

MSK_RES_ERR_INVALID_WHICHSOL (1228)
`whichsol` is invalid.

MSK_RES_ERR_INF_DOU_NAME (1230)
 A double information name is invalid.

MSK_RES_ERR_INF_INT_NAME (1231)
 An integer information name is invalid.

MSK_RES_ERR_INF_TYPE (1232)
 The information type is invalid.

MSK_RES_ERR_INF_LINT_NAME (1234)
 A long integer information name is invalid.

MSK_RES_ERR_INDEX (1235)
 An index is out of range.

MSK_RES_ERR_WHICHSOL (1236)
 The solution defined by `whichsol` does not exists.

MSK_RES_ERR_SOLITEM (1237)
 The solution item number `solitem` is invalid. Please note that `MSK_SOL_ITEM_SNX` is invalid for the basic solution.

MSK_RES_ERR_WHICHITEM_NOT_ALLOWED (1238)
`whichitem` is unacceptable.

MSK_RES_ERR_MAXNUMCON (1240)
 The maximum number of constraints specified is smaller than the number of constraints in the task.

MSK_RES_ERR_MAXNUMVAR (1241)
 The maximum number of variables specified is smaller than the number of variables in the task.

MSK_RES_ERR_MAXNUMBARVAR (1242)
 The maximum number of semidefinite variables specified is smaller than the number of semidefinite variables in the task.

MSK_RES_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.

MSK_RES_ERR_TOO_SMALL_MAX_NUM_NZ (1245)
 The maximum number of non-zeros specified is too small.

MSK_RES_ERR_INVALID_IDX (1246)
 A specified index is invalid.

MSK_RES_ERR_INVALID_MAX_NUM (1247)
 A specified index is invalid.

MSK_RES_ERR_NUMCONLIM (1250)
 Maximum number of constraints limit is exceeded.

MSK_RES_ERR_NUMVARLIM (1251)
 Maximum number of variables limit is exceeded.

MSK_RES_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 .

MSK_RES_ERR_INV_APTRE (1253)
 `aptre[j]` is strictly smaller than `aptrb[j]` for some `j`.

MSK_RES_ERR_MUL_A_ELEMENT (1254)
 An element in A is defined multiple times.

MSK_RES_ERR_INV_BK (1255)
 Invalid bound key.

MSK_RES_ERR_INV_BKC (1256)
 Invalid bound key is specified for a constraint.

MSK_RES_ERR_INV_BKX (1257)
 An invalid bound key is specified for a variable.

MSK_RES_ERR_INV_VAR_TYPE (1258)
 An invalid variable type is specified for a variable.

MSK_RES_ERR_SOLVER_PROBTYPE (1259)
 Problem type does not match the chosen optimizer.

MSK_RES_ERR_OBJECTIVE_RANGE (1260)
 Empty objective range.

MSK_RES_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.

MSK_RES_ERR_BASIS (1266)
 An invalid basis is specified. Either too many or too few basis variables are specified.

MSK_RES_ERR_INV_SKC (1267)
 Invalid value in `skc`.

MSK_RES_ERR_INV_SKX (1268)
 Invalid value in `skx`.

MSK_RES_ERR_INV_SKN (1274)
 Invalid value in `skn`.

MSK_RES_ERR_INV_SK_STR (1269)
 Invalid status key string encountered.

MSK_RES_ERR_INV_SK (1270)
 Invalid status key code.

MSK_RES_ERR_INV_CONE_TYPE_STR (1271)
 Invalid cone type string encountered.

MSK_RES_ERR_INV_CONE_TYPE (1272)
 Invalid cone type code is encountered.

MSK_RES_ERR_INVALID_SURPLUS (1275)
 Invalid surplus.

MSK_RES_ERR_INV_NAME_ITEM (1280)
 An invalid name item code is used.

MSK_RES_ERR_PRO_ITEM (1281)
 An invalid problem is used.

MSK_RES_ERR_INVALID_FORMAT_TYPE (1283)
 Invalid format type.

MSK_RES_ERR_FIRSTI (1285)
 Invalid `firsti`.

MSK_RES_ERR_LASTI (1286)
 Invalid `lasti`.

MSK_RES_ERR_FIRSTJ (1287)
 Invalid `firstj`.

MSK_RES_ERR_LASTJ (1288)
Invalid lastj.

MSK_RES_ERR_MAX_LEN_IS_TOO_SMALL (1289)
A maximum length that is too small has been specified.

MSK_RES_ERR_NONLINEAR_EQUALITY (1290)
The model contains a nonlinear equality which defines a nonconvex set.

MSK_RES_ERR_NONCONVEX (1291)
The optimization problem is nonconvex.

MSK_RES_ERR_NONLINEAR_RANGED (1292)
Nonlinear constraints with finite lower and upper bound always define a nonconvex feasible set.

MSK_RES_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 `MSK_DPAR_CHECK_CONVEXITY_REL_TOL` can be used to relax the convexity check.

MSK_RES_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 `MSK_DPAR_CHECK_CONVEXITY_REL_TOL` can be used to relax the convexity check.

MSK_RES_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 `MSK_DPAR_CHECK_CONVEXITY_REL_TOL` can be used to relax the convexity check.

MSK_RES_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 `MSK_DPAR_CHECK_CONVEXITY_REL_TOL` can be used to relax the convexity check.

MSK_RES_ERR_ARGUMENT_PERM_ARRAY (1299)
An invalid permutation array is specified.

MSK_RES_ERR_CONE_INDEX (1300)
An index of a non-existing cone has been specified.

MSK_RES_ERR_CONE_SIZE (1301)
A cone with incorrect number of members is specified.

MSK_RES_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.

MSK_RES_ERR_CONE_REP_VAR (1303)
A variable is included multiple times in the cone.

MSK_RES_ERR_MAXNUMCONE (1304)
The value specified for `maxnumcone` is too small.

MSK_RES_ERR_CONE_TYPE (1305)
Invalid cone type specified.

MSK_RES_ERR_CONE_TYPE_STR (1306)
Invalid cone type specified.

MSK_RES_ERR_CONE_OVERLAP_APPEND (1307)
The cone to be appended has one variable which is already member of another cone.

MSK_RES_ERR_REMOVE_CONE_VARIABLE (1310)
A variable cannot be removed because it will make a cone invalid.

MSK_RES_ERR_APPENDING_TOO_BIG_CONE (1311)
Trying to append a too big cone.

MSK_RES_ERR_CONE_PARAMETER (1320)
An invalid cone parameter.

MSK_RES_ERR_SOL_FILE_INVALID_NUMBER (1350)
An invalid number is specified in a solution file.

MSK_RES_ERR_HUGE_C (1375)
A huge value in absolute size is specified for one c_j .

MSK_RES_ERR_HUGE_AIJ (1380)
 A numerically huge value is specified for an $a_{i,j}$ element in A . The parameter `MSK_DPAR_DATA_TOL_AIJ_HUGE` controls when an $a_{i,j}$ is considered huge.

MSK_RES_ERR_DUPLICATE_AIJ (1385)
 An element in the A matrix is specified twice.

MSK_RES_ERR_LOWER_BOUND_IS_A_NAN (1390)
 The lower bound specified is not a number (nan).

MSK_RES_ERR_UPPER_BOUND_IS_A_NAN (1391)
 The upper bound specified is not a number (nan).

MSK_RES_ERR_INFINITE_BOUND (1400)
 A numerically huge bound value is specified.

MSK_RES_ERR_INV_QOBJ_SUBI (1401)
 Invalid value in `qosubi`.

MSK_RES_ERR_INV_QOBJ_SUBJ (1402)
 Invalid value in `qosubj`.

MSK_RES_ERR_INV_QOBJ_VAL (1403)
 Invalid value in `qoval`.

MSK_RES_ERR_INV_QCON_SUBK (1404)
 Invalid value in `qcsubk`.

MSK_RES_ERR_INV_QCON_SUBI (1405)
 Invalid value in `qcsubi`.

MSK_RES_ERR_INV_QCON_SUBJ (1406)
 Invalid value in `qcsbj`.

MSK_RES_ERR_INV_QCON_VAL (1407)
 Invalid value in `qcval`.

MSK_RES_ERR_QCON_SUBI_TOO_SMALL (1408)
 Invalid value in `qcsubi`.

MSK_RES_ERR_QCON_SUBI_TOO_LARGE (1409)
 Invalid value in `qcsubi`.

MSK_RES_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.

MSK_RES_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.

MSK_RES_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.

MSK_RES_ERR_TOO_SMALL_A_TRUNCATION_VALUE (1421)
 A too small value for the A truncation value is specified.

MSK_RES_ERR_INVALID_OBJECTIVE_SENSE (1445)
 An invalid objective sense is specified.

MSK_RES_ERR_UNDEFINED_OBJECTIVE_SENSE (1446)
 The objective sense has not been specified before the optimization.

MSK_RES_ERR_Y_IS_UNDEFINED (1449)
 The solution item y is undefined.

MSK_RES_ERR_NAN_IN_DOUBLE_DATA (1450)
 An invalid floating point value was used in some double data.

MSK_RES_ERR_NAN_IN_BLC (1461)
 l^c contains an invalid floating point value, i.e. a NaN.

MSK_RES_ERR_NAN_IN_BUC (1462)
 u^c contains an invalid floating point value, i.e. a NaN.

MSK_RES_ERR_NAN_IN_C (1470)
 c contains an invalid floating point value, i.e. a NaN.

MSK_RES_ERR_NAN_IN_BLX (1471)
 l^x contains an invalid floating point value, i.e. a NaN.

MSK_RES_ERR_NAN_IN_BUX (1472)
 u^x contains an invalid floating point value, i.e. a NaN.

MSK_RES_ERR_INVALID_AIJ (1473)
 $a_{i,j}$ contains an invalid floating point value, i.e. a NaN or an infinite value.

MSK_RES_ERR_SYM_MAT_INVALID (1480)
A symmetric matrix contains an invalid floating point value, i.e. a NaN or an infinite value.

MSK_RES_ERR_SYM_MAT_HUGE (1482)
A symmetric matrix contains a huge value in absolute size. The parameter `MSK_DPAR_DATA_SYM_MAT_TOL_HUGE` controls when an $e_{i,j}$ is considered huge.

MSK_RES_ERR_INV_PROBLEM (1500)
Invalid problem type. Probably a nonconvex problem has been specified.

MSK_RES_ERR_MIXED_CONIC_AND_NL (1501)
The problem contains nonlinear terms conic constraints. The requested operation cannot be applied to this type of problem.

MSK_RES_ERR_GLOBAL_INV_CONIC_PROBLEM (1503)
The global optimizer can only be applied to problems without semidefinite variables.

MSK_RES_ERR_INV_OPTIMIZER (1550)
An invalid optimizer has been chosen for the problem.

MSK_RES_ERR_MIO_NO_OPTIMIZER (1551)
No optimizer is available for the current class of integer optimization problems.

MSK_RES_ERR_NO_OPTIMIZER_VAR_TYPE (1552)
No optimizer is available for this class of optimization problems.

MSK_RES_ERR_FINAL_SOLUTION (1560)
An error occurred during the solution finalization.

MSK_RES_ERR_FIRST (1570)
Invalid `first`.

MSK_RES_ERR_LAST (1571)
Invalid index `last`. A given index was out of expected range.

MSK_RES_ERR_SLICE_SIZE (1572)
Invalid slice size specified.

MSK_RES_ERR_NEGATIVE_SURPLUS (1573)
Negative surplus.

MSK_RES_ERR_NEGATIVE_APPEND (1578)
Cannot append a negative number.

MSK_RES_ERR_POSTSOLVE (1580)
An error occurred during the postsolve. Please contact **MOSEK** support.

MSK_RES_ERR_OVERFLOW (1590)
A computation produced an overflow i.e. a very large number.

MSK_RES_ERR_NO_BASIS_SOL (1600)
No basic solution is defined.

MSK_RES_ERR_BASIS_FACTOR (1610)
The factorization of the basis is invalid.

MSK_RES_ERR_BASIS_SINGULAR (1615)
The basis is singular and hence cannot be factored.

MSK_RES_ERR_FACTOR (1650)
An error occurred while factorizing a matrix.

MSK_RES_ERR_FEASREPAIR_CANNOT_RELAX (1700)
An optimization problem cannot be relaxed.

MSK_RES_ERR_FEASREPAIR_SOLVING_RELAXED (1701)
The relaxed problem could not be solved to optimality. Please consult the log file for further details.

MSK_RES_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.

MSK_RES_ERR_REPAIR_INVALID_PROBLEM (1710)
The feasibility repair does not support the specified problem type.

MSK_RES_ERR_REPAIR_OPTIMIZATION_FAILED (1711)
Computation the optimal relaxation failed. The cause may have been numerical problems.

MSK_RES_ERR_NAME_MAX_LEN (1750)
A name is longer than the buffer that is supposed to hold it.

MSK_RES_ERR_NAME_IS_NULL (1760)
The name buffer is a NULL pointer.

MSK_RES_ERR_INVALID_COMPRESSION (1800)
Invalid compression type.

MSK_RES_ERR_INVALID_IOMODE (1801)
Invalid io mode.

MSK_RES_ERR_NO_PRIMAL_INFEAS_CER (2000)
A certificate of primal infeasibility is not available.

MSK_RES_ERR_NO_DUAL_INFEAS_CER (2001)
A certificate of infeasibility is not available.

MSK_RES_ERR_NO_SOLUTION_IN_CALLBACK (2500)
The required solution is not available.

MSK_RES_ERR_INV_MARKI (2501)
Invalid value in marki.

MSK_RES_ERR_INV_MARKJ (2502)
Invalid value in markj.

MSK_RES_ERR_INV_NUMI (2503)
Invalid numi.

MSK_RES_ERR_INV_NUMJ (2504)
Invalid numj.

MSK_RES_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.

MSK_RES_ERR_TASK_INVALID (2561)
The Task file is invalid.

MSK_RES_ERR_TASK_WRITE (2562)
Failed to write the task file.

MSK_RES_ERR_LU_MAX_NUM_TRIES (2800)
Could not compute the LU factors of the matrix within the maximum number of allowed tries.

MSK_RES_ERR_INVALID_UTF8 (2900)
An invalid UTF8 string is encountered.

MSK_RES_ERR_INVALID_WCHAR (2901)
An invalid `wchar` string is encountered.

MSK_RES_ERR_NO_DUAL_FOR_ITG_SOL (2950)
No dual information is available for the integer solution.

MSK_RES_ERR_NO_SNX_FOR_BAS_SOL (2953)
 s_n^x is not available for the basis solution.

MSK_RES_ERR_INTERNAL (3000)
An internal error occurred. Please report this problem.

MSK_RES_ERR_API_ARRAY_TOO_SMALL (3001)
An input array was too short.

MSK_RES_ERR_API_CB_CONNECT (3002)
Failed to connect a callback object.

MSK_RES_ERR_API_FATAL_ERROR (3005)
An internal error occurred in the API. Please report this problem.

MSK_RES_ERR_API_INTERNAL (3999)
An internal fatal error occurred in an interface function.

MSK_RES_ERR_SEN_FORMAT (3050)
Syntax error in sensitivity analysis file.

MSK_RES_ERR_SEN_UNDEF_NAME (3051)
An undefined name was encountered in the sensitivity analysis file.

MSK_RES_ERR_SEN_INDEX_RANGE (3052)
Index out of range in the sensitivity analysis file.

MSK_RES_ERR_SEN_BOUND_INVALID_UP (3053)
Analysis of upper bound requested for an index, where no upper bound exists.

MSK_RES_ERR_SEN_BOUND_INVALID_LO (3054)
Analysis of lower bound requested for an index, where no lower bound exists.

MSK_RES_ERR_SEN_INDEX_INVALID (3055)
Invalid range given in the sensitivity file.

MSK_RES_ERR_SEN_INVALID_REGEX (3056)
Syntax error in regexp or regexp longer than 1024.

MSK_RES_ERR_SEN_SOLUTION_STATUS (3057)
No optimal solution found to the original problem given for sensitivity analysis.

MSK_RES_ERR_SEN_NUMERICAL (3058)
Numerical difficulties encountered performing the sensitivity analysis.

MSK_RES_ERR_SEN_UNHANDLED_PROBLEM_TYPE (3080)
Sensitivity analysis cannot be performed for the specified problem. Sensitivity analysis is only possible for linear problems.

MSK_RES_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.

MSK_RES_ERR_IDENTICAL_TASKS (3101)
Some tasks related to this function call were identical. Unique tasks were expected.

MSK_RES_ERR_AD_INVALID_CODELIST (3102)
The code list data was invalid.

MSK_RES_ERR_INTERNAL_TEST_FAILED (3500)
An internal unit test function failed.

MSK_RES_ERR_XML_INVALID_PROBLEM_TYPE (3600)
The problem type is not supported by the XML format.

MSK_RES_ERR_INVALIDAMPL_STUB (3700)
Invalid AMPL stub.

MSK_RES_ERR_INT64_TO_INT32_CAST (3800)
A 64 bit integer could not be cast to a 32 bit integer.

MSK_RES_ERR_SIZE_LICENSE_NUMCORES (3900)
The computer contains more cpu cores than the license allows for.

MSK_RES_ERR_INFEAS_UNDEFINED (3910)
The requested value is not defined for this solution type.

MSK_RES_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.

MSK_RES_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.

MSK_RES_ERR_BAR_VAR_DIM (3920)
The dimension of a symmetric matrix variable has to be greater than 0.

MSK_RES_ERR_SYM_MAT_INVALID_ROW_INDEX (3940)
A row index specified for sparse symmetric matrix is invalid.

MSK_RES_ERR_SYM_MAT_INVALID_COL_INDEX (3941)
A column index specified for sparse symmetric matrix is invalid.

MSK_RES_ERR_SYM_MAT_NOT_LOWER_TRINGULAR (3942)
Only the lower triangular part of sparse symmetric matrix should be specified.

MSK_RES_ERR_SYM_MAT_INVALID_VALUE (3943)
The numerical value specified in a sparse symmetric matrix is not a floating point value.

MSK_RES_ERR_SYM_MAT_DUPLICATE (3944)
A value in a symmetric matrix as been specified more than once.

MSK_RES_ERR_INVALID_SYM_MAT_DIM (3950)
A sparse symmetric matrix of invalid dimension is specified.

MSK_RES_ERR_INVALID_FILE_FORMAT_FOR_SYM_MAT (4000)
The file format does not support a problem with symmetric matrix variables.

MSK_RES_ERR_INVALID_FILE_FORMAT_FOR_CFIX (4001)
The file format does not support a problem with nonzero fixed term in c.

MSK_RES_ERR_INVALID_FILE_FORMAT_FOR_RANGED_CONSTRAINTS (4002)
The file format does not support a problem with ranged constraints.

MSK_RES_ERR_INVALID_FILE_FORMAT_FOR_FREE_CONSTRAINTS (4003)
The file format does not support a problem with free constraints.

MSK_RES_ERR_INVALID_FILE_FORMAT_FOR_CONES (4005)
The file format does not support a problem with conic constraints.

MSK_RES_ERR_INVALID_FILE_FORMAT_FOR_NONLINEAR (4010)
The file format does not support a problem with nonlinear terms.

MSK_RES_ERR_DUPLICATE_CONSTRAINT_NAMES (4500)
Two constraint names are identical.

MSK_RES_ERR_DUPLICATE_VARIABLE_NAMES (4501)
Two variable names are identical.

MSK_RES_ERR_DUPLICATE_BARVARIABLE_NAMES (4502)
Two barvariable names are identical.

MSK_RES_ERR_DUPLICATE_CONE_NAMES (4503)
Two cone names are identical.

MSK_RES_ERR_NON_UNIQUE_ARRAY (5000)
An array does not contain unique elements.

MSK_RES_ERR_ARGUMENT_IS_TOO_LARGE (5005)
The value of a function argument is too large.

MSK_RES_ERR_MIO_INTERNAL (5010)
A fatal error occurred in the mixed integer optimizer. Please contact **MOSEK** support.

MSK_RES_ERR_INVALID_PROBLEM_TYPE (6000)
An invalid problem type.

MSK_RES_ERR_UNHANDLED_SOLUTION_STATUS (6010)
Unhandled solution status.

MSK_RES_ERR_UPPER_TRIANGLE (6020)
An element in the upper triangle of a lower triangular matrix is specified.

MSK_RES_ERR_LAU_SINGULAR_MATRIX (7000)
A matrix is singular.

MSK_RES_ERR_LAU_NOT_POSITIVE_DEFINITE (7001)
A matrix is not positive definite.

MSK_RES_ERR_LAU_INVALID_LOWER_TRIANGULAR_MATRIX (7002)
An invalid lower triangular matrix.

MSK_RES_ERR_LAU_UNKNOWN (7005)
An unknown error.

MSK_RES_ERR_LAU_ARG_M (7010)
Invalid argument m.

MSK_RES_ERR_LAU_ARG_N (7011)
Invalid argument n.

MSK_RES_ERR_LAU_ARG_K (7012)
Invalid argument k.

MSK_RES_ERR_LAU_ARG_TRANSA (7015)
Invalid argument transa.

MSK_RES_ERR_LAU_ARG_TRANSB (7016)
Invalid argument transb.

MSK_RES_ERR_LAU_ARG_UPLO (7017)
Invalid argument uplo.

MSK_RES_ERR_LAU_ARG_TRANS (7018)
Invalid argument trans.

MSK_RES_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.

MSK_RES_ERR_CBF_PARSE (7100)
An error occurred while parsing an CBF file.

MSK_RES_ERR_CBF_OBJ_SENSE (7101)
An invalid objective sense is specified.

MSK_RES_ERR_CBF_NO_VARIABLES (7102)
No variables are specified.

MSK_RES_ERR_CBF_TOO_MANY_CONSTRAINTS (7103)
Too many constraints specified.

MSK_RES_ERR_CBF_TOO_MANY_VARIABLES (7104)
Too many variables specified.

MSK_RES_ERR_CBF_NO_VERSION_SPECIFIED (7105)
No version specified.

MSK_RES_ERR_CBF_SYNTAX (7106)
Invalid syntax.

MSK_RES_ERR_CBF_DUPLICATE_OBJ (7107)
Duplicate OBJ keyword.

MSK_RES_ERR_CBF_DUPLICATE_CON (7108)
Duplicate CON keyword.

MSK_RES_ERR_CBF_DUPLICATE_VAR (7109)
Duplicate VAR keyword.

MSK_RES_ERR_CBF_DUPLICATE_INT (7110)
Duplicate INT keyword.

MSK_RES_ERR_CBF_INVALID_VAR_TYPE (7111)
Invalid variable type.

MSK_RES_ERR_CBF_INVALID_CON_TYPE (7112)
Invalid constraint type.

MSK_RES_ERR_CBF_INVALID_DOMAIN_DIMENSION (7113)
Invalid domain dimension.

MSK_RES_ERR_CBF_DUPLICATE_OBJCOORD (7114)
Duplicate index in OBJCOORD.

MSK_RES_ERR_CBF_DUPLICATE_BCOORD (7115)
Duplicate index in BCOORD.

MSK_RES_ERR_CBF_DUPLICATE_ACOORD (7116)
Duplicate index in ACOORD.

MSK_RES_ERR_CBF_TOO_FEW_VARIABLES (7117)
Too few variables defined.

MSK_RES_ERR_CBF_TOO_FEW_CONSTRAINTS (7118)
Too few constraints defined.

MSK_RES_ERR_CBF_TOO_FEW_INTS (7119)
Too few ints are specified.

MSK_RES_ERR_CBF_TOO_MANY_INTS (7120)
Too many ints are specified.

MSK_RES_ERR_CBF_INVALID_INT_INDEX (7121)
Invalid INT index.

MSK_RES_ERR_CBF_UNSUPPORTED (7122)
Unsupported feature is present.

MSK_RES_ERR_CBF_DUPLICATE_PSDVAR (7123)
Duplicate PSDVAR keyword.

MSK_RES_ERR_CBF_INVALID_PSDVAR_DIMENSION (7124)
Invalid PSDVAR dimension.

MSK_RES_ERR_CBF_TOO_FEW_PSDVAR (7125)
Too few variables defined.

MSK_RES_ERR_CBF_INVALID_EXP_DIMENSION (7126)
Invalid dimension of a exponential cone.

MSK_RES_ERR_CBF_DUPLICATE_POW_CONES (7130)
Multiple POWCONES specified.

MSK_RES_ERR_CBF_DUPLICATE_POW_STAR_CONES (7131)
Multiple POW*CONES specified.

MSK_RES_ERR_CBF_INVALID_POWER (7132)
Invalid power specified.

MSK_RES_ERR_CBF_POWER_CONE_IS_TOO_LONG (7133)
Power cone is too long.

MSK_RES_ERR_CBF_INVALID_POWER_CONE_INDEX (7134)
Invalid power cone index.

MSK_RES_ERR_CBF_INVALID_POWER_STAR_CONE_INDEX (7135)
 Invalid power star cone index.

MSK_RES_ERR_CBF_UNHANDLED_POWER_CONE_TYPE (7136)
 An unhandled power cone type.

MSK_RES_ERR_CBF_UNHANDLED_POWER_STAR_CONE_TYPE (7137)
 An unhandled power star cone type.

MSK_RES_ERR_CBF_POWER_CONE_MISMATCH (7138)
 The power cone does not match with its definition.

MSK_RES_ERR_CBF_POWER_STAR_CONE_MISMATCH (7139)
 The power star cone does not match with its definition.

MSK_RES_ERR_CBF_INVALID_NUMBER_OF_CONES (7740)
 Invalid number of cones.

MSK_RES_ERR_CBF_INVALID_DIMENSION_OF_CONES (7741)
 Invalid dimension of cones.

MSK_RES_ERR_MIO_INVALID_ROOT_OPTIMIZER (7700)
 An invalid root optimizer was selected for the problem type.

MSK_RES_ERR_MIO_INVALID_NODE_OPTIMIZER (7701)
 An invalid node optimizer was selected for the problem type.

MSK_RES_ERR_TOCONIC_CONSTR_Q_NOT_PSD (7800)
 The matrix defining the quadratic part of constraint is not positive semidefinite.

MSK_RES_ERR_TOCONIC_CONSTRAINT_FX (7801)
 The quadratic constraint is an equality, thus not convex.

MSK_RES_ERR_TOCONIC_CONSTRAINT_RA (7802)
 The quadratic constraint has finite lower and upper bound, and therefore it is not convex.

MSK_RES_ERR_TOCONIC_CONSTR_NOT_CONIC (7803)
 The constraint is not conic representable.

MSK_RES_ERR_TOCONIC_OBJECTIVE_NOT_PSD (7804)
 The matrix defining the quadratic part of the objective function is not positive semidefinite.

MSK_RES_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.

MSK_RES_ERR_SERVER_PROTOCOL (8001)
 Unexpected message or data from solver server.

MSK_RES_ERR_SERVER_STATUS (8002)
 Server returned non-ok HTTP status code

MSK_RES_ERR_SERVER_TOKEN (8003)
 The job ID specified is incorrect or invalid

15.7 Enumerations

MSKbasindtypee

Basis identification

MSK_BI_NEVER (0)

Never do basis identification.

MSK_BI_ALWAYS (1)

Basis identification is always performed even if the interior-point optimizer terminates abnormally.

MSK_BI_NO_ERROR (2)

Basis identification is performed if the interior-point optimizer terminates without an error.

MSK_BI_IF_FEASIBLE (3)

Basis identification is not performed if the interior-point optimizer terminates with a problem status saying that the problem is primal or dual infeasible.

MSK_BI_RESERVED (4)

Not currently in use.

MSKboundkeye

Bound keys

MSK_BK_LO (0)
The constraint or variable has a finite lower bound and an infinite upper bound.

MSK_BK_UP (1)
The constraint or variable has an infinite lower bound and an finite upper bound.

MSK_BK_FX (2)
The constraint or variable is fixed.

MSK_BK_FR (3)
The constraint or variable is free.

MSK_BK_RA (4)
The constraint or variable is ranged.

MSKmarke
Mark

MSK_MARK_LO (0)
The lower bound is selected for sensitivity analysis.

MSK_MARK_UP (1)
The upper bound is selected for sensitivity analysis.

MSKsimdegene
Degeneracy strategies

MSK_SIM_DEGEN_NONE (0)
The simplex optimizer should use no degeneration strategy.

MSK_SIM_DEGEN_FREE (1)
The simplex optimizer chooses the degeneration strategy.

MSK_SIM_DEGEN_AGGRESSIVE (2)
The simplex optimizer should use an aggressive degeneration strategy.

MSK_SIM_DEGEN_MODERATE (3)
The simplex optimizer should use a moderate degeneration strategy.

MSK_SIM_DEGEN_MINIMUM (4)
The simplex optimizer should use a minimum degeneration strategy.

MSKtransposee
Transposed matrix.

MSK_TRANSPOSE_NO (0)
No transpose is applied.

MSK_TRANSPOSE_YES (1)
A transpose is applied.

MSKuploe
Triangular part of a symmetric matrix.

MSK_UPLO_LO (0)
Lower part.

MSK_UPLO_UP (1)
Upper part.

MSKsimreforme
Problem reformulation.

MSK_SIM_REFORMULATION_ON (1)
Allow the simplex optimizer to reformulate the problem.

MSK_SIM_REFORMULATION_OFF (0)
Disallow the simplex optimizer to reformulate the problem.

MSK_SIM_REFORMULATION_FREE (2)
The simplex optimizer can choose freely.

MSK_SIM_REFORMULATION_AGGRESSIVE (3)
The simplex optimizer should use an aggressive reformulation strategy.

MSKsimdupvece
Exploit duplicate columns.

MSK_SIM_EXPLOIT_DUPVEC_ON (1)
 Allow the simplex optimizer to exploit duplicated columns.

MSK_SIM_EXPLOIT_DUPVEC_OFF (0)
 Disallow the simplex optimizer to exploit duplicated columns.

MSK_SIM_EXPLOIT_DUPVEC_FREE (2)
 The simplex optimizer can choose freely.

MSKsimhotstarte
 Hot-start type employed by the simplex optimizer

MSK_SIM_HOTSTART_NONE (0)
 The simplex optimizer performs a coldstart.

MSK_SIM_HOTSTART_FREE (1)
 The simplex optimizer chooses the hot-start type.

MSK_SIM_HOTSTART_STATUS_KEYS (2)
 Only the status keys of the constraints and variables are used to choose the type of hot-start.

MSKintpntshotstarte
 Hot-start type employed by the interior-point optimizers.

MSK_INTPNT_HOTSTART_NONE (0)
 The interior-point optimizer performs a coldstart.

MSK_INTPNT_HOTSTART_PRIMAL (1)
 The interior-point optimizer exploits the primal solution only.

MSK_INTPNT_HOTSTART_DUAL (2)
 The interior-point optimizer exploits the dual solution only.

MSK_INTPNT_HOTSTART_PRIMAL_DUAL (3)
 The interior-point optimizer exploits both the primal and dual solution.

MSKpurifye
 Solution purification employed optimizer.

MSK_PURIFY_NONE (0)
 The optimizer performs no solution purification.

MSK_PURIFY_PRIMAL (1)
 The optimizer purifies the primal solution.

MSK_PURIFY_DUAL (2)
 The optimizer purifies the dual solution.

MSK_PURIFY_PRIMAL_DUAL (3)
 The optimizer purifies both the primal and dual solution.

MSK_PURIFY_AUTO (4)
 TBD

MSKcallbackcodee
 Progress callback codes

MSK_CALLBACK_BEGIN_BI (0)
 The basis identification procedure has been started.

MSK_CALLBACK_BEGIN_CONIC (1)
 The callback function is called when the conic optimizer is started.

MSK_CALLBACK_BEGIN_DUAL_BI (2)
 The callback function is called from within the basis identification procedure when the dual phase is started.

MSK_CALLBACK_BEGIN_DUAL_SENSITIVITY (3)
 Dual sensitivity analysis is started.

MSK_CALLBACK_BEGIN_DUAL_SETUP_BI (4)
 The callback function is called when the dual BI phase is started.

MSK_CALLBACK_BEGIN_DUAL_SIMPLEX (5)
 The callback function is called when the dual simplex optimizer started.

MSK_CALLBACK_BEGIN_DUAL_SIMPLEX_BI (6)
The callback function is called from within the basis identification procedure when the dual simplex clean-up phase is started.

MSK_CALLBACK_BEGIN_FULL_CONVEXITY_CHECK (7)
Begin full convexity check.

MSK_CALLBACK_BEGIN_INFEAS_ANA (8)
The callback function is called when the infeasibility analyzer is started.

MSK_CALLBACK_BEGIN_INTPNT (9)
The callback function is called when the interior-point optimizer is started.

MSK_CALLBACK_BEGIN_LICENSE_WAIT (10)
Begin waiting for license.

MSK_CALLBACK_BEGIN_MIO (11)
The callback function is called when the mixed-integer optimizer is started.

MSK_CALLBACK_BEGIN_OPTIMIZER (12)
The callback function is called when the optimizer is started.

MSK_CALLBACK_BEGIN_PRESOLVE (13)
The callback function is called when the presolve is started.

MSK_CALLBACK_BEGIN_PRIMAL_BI (14)
The callback function is called from within the basis identification procedure when the primal phase is started.

MSK_CALLBACK_BEGIN_PRIMAL_REPAIR (15)
Begin primal feasibility repair.

MSK_CALLBACK_BEGIN_PRIMAL_SENSITIVITY (16)
Primal sensitivity analysis is started.

MSK_CALLBACK_BEGIN_PRIMAL_SETUP_BI (17)
The callback function is called when the primal BI setup is started.

MSK_CALLBACK_BEGIN_PRIMAL_SIMPLEX (18)
The callback function is called when the primal simplex optimizer is started.

MSK_CALLBACK_BEGIN_PRIMAL_SIMPLEX_BI (19)
The callback function is called from within the basis identification procedure when the primal simplex clean-up phase is started.

MSK_CALLBACK_BEGIN_QCQO_REFORMULATE (20)
Begin QCQO reformulation.

MSK_CALLBACK_BEGIN_READ (21)
MOSEK has started reading a problem file.

MSK_CALLBACK_BEGIN_ROOT_CUTGEN (22)
The callback function is called when root cut generation is started.

MSK_CALLBACK_BEGIN_SIMPLEX (23)
The callback function is called when the simplex optimizer is started.

MSK_CALLBACK_BEGIN_SIMPLEX_BI (24)
The callback function is called from within the basis identification procedure when the simplex clean-up phase is started.

MSK_CALLBACK_BEGIN_TO_CONIC (25)
Begin conic reformulation.

MSK_CALLBACK_BEGIN_WRITE (26)
MOSEK has started writing a problem file.

MSK_CALLBACK_CONIC (27)
The callback function is called from within the conic optimizer after the information database has been updated.

MSK_CALLBACK_DUAL_SIMPLEX (28)
The callback function is called from within the dual simplex optimizer.

MSK_CALLBACK_END_BI (29)
The callback function is called when the basis identification procedure is terminated.

MSK_CALLBACK_END_CONIC (30)
The callback function is called when the conic optimizer is terminated.

MSK_CALLBACK_END_DUAL_BI (31)
The callback function is called from within the basis identification procedure when the dual phase is terminated.

MSK_CALLBACK_END_DUAL_SENSITIVITY (32)
Dual sensitivity analysis is terminated.

MSK_CALLBACK_END_DUAL_SETUP_BI (33)
The callback function is called when the dual BI phase is terminated.

MSK_CALLBACK_END_DUAL_SIMPLEX (34)
The callback function is called when the dual simplex optimizer is terminated.

MSK_CALLBACK_END_DUAL_SIMPLEX_BI (35)
The callback function is called from within the basis identification procedure when the dual clean-up phase is terminated.

MSK_CALLBACK_END_FULL_CONVEXITY_CHECK (36)
End full convexity check.

MSK_CALLBACK_END_INFEAS_ANA (37)
The callback function is called when the infeasibility analyzer is terminated.

MSK_CALLBACK_END_INTPNT (38)
The callback function is called when the interior-point optimizer is terminated.

MSK_CALLBACK_END_LICENSE_WAIT (39)
End waiting for license.

MSK_CALLBACK_END_MIO (40)
The callback function is called when the mixed-integer optimizer is terminated.

MSK_CALLBACK_END_OPTIMIZER (41)
The callback function is called when the optimizer is terminated.

MSK_CALLBACK_END_PRESOLVE (42)
The callback function is called when the presolve is completed.

MSK_CALLBACK_END_PRIMAL_BI (43)
The callback function is called from within the basis identification procedure when the primal phase is terminated.

MSK_CALLBACK_END_PRIMAL_REPAIR (44)
End primal feasibility repair.

MSK_CALLBACK_END_PRIMAL_SENSITIVITY (45)
Primal sensitivity analysis is terminated.

MSK_CALLBACK_END_PRIMAL_SETUP_BI (46)
The callback function is called when the primal BI setup is terminated.

MSK_CALLBACK_END_PRIMAL_SIMPLEX (47)
The callback function is called when the primal simplex optimizer is terminated.

MSK_CALLBACK_END_PRIMAL_SIMPLEX_BI (48)
The callback function is called from within the basis identification procedure when the primal clean-up phase is terminated.

MSK_CALLBACK_END_QCQO_REFORMULATE (49)
End QCQO reformulation.

MSK_CALLBACK_END_READ (50)
MOSEK has finished reading a problem file.

MSK_CALLBACK_END_ROOT_CUTGEN (51)
The callback function is called when root cut generation is terminated.

MSK_CALLBACK_END_SIMPLEX (52)
The callback function is called when the simplex optimizer is terminated.

MSK_CALLBACK_END_SIMPLEX_BI (53)
The callback function is called from within the basis identification procedure when the simplex clean-up phase is terminated.

MSK_CALLBACK_END_TO_CONIC (54)
End conic reformulation.

MSK_CALLBACK_END_WRITE (55)
MOSEK has finished writing a problem file.

MSK_CALLBACK_IM_BI (56)
The callback function is called from within the basis identification procedure at an intermediate point.

MSK_CALLBACK_IM_CONIC (57)
The callback function is called at an intermediate stage within the conic optimizer where the information database has not been updated.

MSK_CALLBACK_IM_DUAL_BI (58)
The callback function is called from within the basis identification procedure at an intermediate point in the dual phase.

MSK_CALLBACK_IM_DUAL_SENSIVITY (59)
The callback function is called at an intermediate stage of the dual sensitivity analysis.

MSK_CALLBACK_IM_DUAL_SIMPLEX (60)
The callback function is called at an intermediate point in the dual simplex optimizer.

MSK_CALLBACK_IM_FULL_CONVEXITY_CHECK (61)
The callback function is called at an intermediate stage of the full convexity check.

MSK_CALLBACK_IM_INTPNT (62)
The callback function is called at an intermediate stage within the interior-point optimizer where the information database has not been updated.

MSK_CALLBACK_IM_LICENSE_WAIT (63)
MOSEK is waiting for a license.

MSK_CALLBACK_IM_LU (64)
The callback function is called from within the LU factorization procedure at an intermediate point.

MSK_CALLBACK_IM_MIO (65)
The callback function is called at an intermediate point in the mixed-integer optimizer.

MSK_CALLBACK_IM_MIO_DUAL_SIMPLEX (66)
The callback function is called at an intermediate point in the mixed-integer optimizer while running the dual simplex optimizer.

MSK_CALLBACK_IM_MIO_INTPNT (67)
The callback function is called at an intermediate point in the mixed-integer optimizer while running the interior-point optimizer.

MSK_CALLBACK_IM_MIO_PRIMAL_SIMPLEX (68)
The callback function is called at an intermediate point in the mixed-integer optimizer while running the primal simplex optimizer.

MSK_CALLBACK_IM_ORDER (69)
The callback function is called from within the matrix ordering procedure at an intermediate point.

MSK_CALLBACK_IM_PRESOLVE (70)
The callback function is called from within the presolve procedure at an intermediate stage.

MSK_CALLBACK_IM_PRIMAL_BI (71)
The callback function is called from within the basis identification procedure at an intermediate point in the primal phase.

MSK_CALLBACK_IM_PRIMAL_SENSIVITY (72)

The callback function is called at an intermediate stage of the primal sensitivity analysis.

MSK_CALLBACK_IM_PRIMAL_SIMPLEX (73)

The callback function is called at an intermediate point in the primal simplex optimizer.

MSK_CALLBACK_IM_QO_REFORMULATE (74)

The callback function is called at an intermediate stage of the conic quadratic reformulation.

MSK_CALLBACK_IM_READ (75)

Intermediate stage in reading.

MSK_CALLBACK_IM_ROOT_CUTGEN (76)

The callback is called from within root cut generation at an intermediate stage.

MSK_CALLBACK_IM_SIMPLEX (77)

The callback function is called from within the simplex optimizer at an intermediate point.

MSK_CALLBACK_IM_SIMPLEX_BI (78)

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 *MSK_IPAR_LOG_SIM_FREQ* parameter.

MSK_CALLBACK_INTPNT (79)

The callback function is called from within the interior-point optimizer after the information database has been updated.

MSK_CALLBACK_NEW_INT_MIO (80)

The callback function is called after a new integer solution has been located by the mixed-integer optimizer.

MSK_CALLBACK_PRIMAL_SIMPLEX (81)

The callback function is called from within the primal simplex optimizer.

MSK_CALLBACK_READ_OPF (82)

The callback function is called from the OPF reader.

MSK_CALLBACK_READ_OPF_SECTION (83)

A chunk of Q non-zeros has been read from a problem file.

MSK_CALLBACK_SOLVING_REMOTE (84)

The callback function is called while the task is being solved on a remote server.

MSK_CALLBACK_UPDATE_DUAL_BI (85)

The callback function is called from within the basis identification procedure at an intermediate point in the dual phase.

MSK_CALLBACK_UPDATE_DUAL_SIMPLEX (86)

The callback function is called in the dual simplex optimizer.

MSK_CALLBACK_UPDATE_DUAL_SIMPLEX_BI (87)

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 *MSK_IPAR_LOG_SIM_FREQ* parameter.

MSK_CALLBACK_UPDATE_PRESOLVE (88)

The callback function is called from within the presolve procedure.

MSK_CALLBACK_UPDATE_PRIMAL_BI (89)

The callback function is called from within the basis identification procedure at an intermediate point in the primal phase.

MSK_CALLBACK_UPDATE_PRIMAL_SIMPLEX (90)

The callback function is called in the primal simplex optimizer.

MSK_CALLBACK_UPDATE_PRIMAL_SIMPLEX_BI (91)

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 *MSK_IPAR_LOG_SIM_FREQ* parameter.

MSK_CALLBACK_WRITE_OPF (92)

The callback function is called from the OPF writer.

MSKcheckconvexitytypee

Types of convexity checks.

MSK_CHECK_CONVEXITY_NONE (0)

No convexity check.

MSK_CHECK_CONVEXITY_SIMPLE (1)

Perform simple and fast convexity check.

MSK_CHECK_CONVEXITY_FULL (2)

Perform a full convexity check.

MSKcompresstypee

Compression types

MSK_COMPRESS_NONE (0)

No compression is used.

MSK_COMPRESS_FREE (1)

The type of compression used is chosen automatically.

MSK_COMPRESS_GZIP (2)

The type of compression used is gzip compatible.

MSK_COMPRESS_ZSTD (3)

The type of compression used is zstd compatible.

MSKconetypee

Cone types

MSK_CT_QUAD (0)

The cone is a quadratic cone.

MSK_CT_RQUAD (1)

The cone is a rotated quadratic cone.

MSK_CT_PEXP (2)

A primal exponential cone.

MSK_CT_DEXP (3)

A dual exponential cone.

MSK_CT_PPOW (4)

A primal power cone.

MSK_CT_DPOW (5)

A dual power cone.

MSK_CT_ZERO (6)

The zero cone.

MSKnametypee

Name types

MSK_NAME_TYPE_GEN (0)

General names. However, no duplicate and blank names are allowed.

MSK_NAME_TYPE_MPS (1)

MPS type names.

MSK_NAME_TYPE_LP (2)

LP type names.

MSKscope

SCopt operator types

MSK_OPR_ENT (0)

Entropy

MSK_OPR_EXP (1)

Exponential

MSK_OPR_LOG (2)

Logarithm

MSK_OPR_POW (3)

Power

MSK_OPR_SQRT (4)
Square root

MSKsymmattypee
Cone types

MSK_SYMMAT_TYPE_SPARSE (0)
Sparse symmetric matrix.

MSKdataformat
Data format types

MSK_DATA_FORMAT_EXTENSION (0)
The file extension is used to determine the data file format.

MSK_DATA_FORMAT_MPS (1)
The data file is MPS formatted.

MSK_DATA_FORMAT_LP (2)
The data file is LP formatted.

MSK_DATA_FORMAT_OP (3)
The data file is an optimization problem formatted file.

MSK_DATA_FORMAT_FREE_MPS (4)
The data a free MPS formatted file.

MSK_DATA_FORMAT_TASK (5)
Generic task dump file.

MSK_DATA_FORMAT_PTF (6)
(P)retty (T)ext (F)format.

MSK_DATA_FORMAT_CB (7)
Conic benchmark format,

MSK_DATA_FORMAT_JSON_TASK (8)
JSON based task format.

MSKdinfiteme
Double information items

MSK_DINF_BI_CLEAN_DUAL_TIME (0)
Time spent within the dual clean-up optimizer of the basis identification procedure since its invocation.

MSK_DINF_BI_CLEAN_PRIMAL_TIME (1)
Time spent within the primal clean-up optimizer of the basis identification procedure since its invocation.

MSK_DINF_BI_CLEAN_TIME (2)
Time spent within the clean-up phase of the basis identification procedure since its invocation.

MSK_DINF_BI_DUAL_TIME (3)
Time spent within the dual phase basis identification procedure since its invocation.

MSK_DINF_BI_PRIMAL_TIME (4)
Time spent within the primal phase of the basis identification procedure since its invocation.

MSK_DINF_BI_TIME (5)
Time spent within the basis identification procedure since its invocation.

MSK_DINF_INTPNT_DUAL_FEAS (6)
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.)

MSK_DINF_INTPNT_DUAL_OBJ (7)
Dual objective value reported by the interior-point optimizer.

MSK_DINF_INTPNT_FACTOR_NUM_FLOPS (8)
An estimate of the number of flops used in the factorization.

MSK_DINF_INTPNT_OPT_STATUS (9)

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.

MSK_DINF_INTPNT_ORDER_TIME (10)

Order time (in seconds).

MSK_DINF_INTPNT_PRIMAL_FEAS (11)

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).

MSK_DINF_INTPNT_PRIMAL_OBJ (12)

Primal objective value reported by the interior-point optimizer.

MSK_DINF_INTPNT_TIME (13)

Time spent within the interior-point optimizer since its invocation.

MSK_DINF_MIO_CLIQUE_SEPARATION_TIME (14)

Separation time for clique cuts.

MSK_DINF_MIO_CMIR_SEPARATION_TIME (15)

Separation time for CMIR cuts.

MSK_DINF_MIO_CONSTRUCT_SOLUTION_OBJ (16)

If **MOSEK** has successfully constructed an integer feasible solution, then this item contains the optimal objective value corresponding to the feasible solution.

MSK_DINF_MIO_DUAL_BOUND_AFTER_PRESOLVE (17)

Value of the dual bound after presolve but before cut generation.

MSK_DINF_MIO_GMI_SEPARATION_TIME (18)

Separation time for GMI cuts.

MSK_DINF_MIO_IMPLIED_BOUND_TIME (19)

Separation time for implied bound cuts.

MSK_DINF_MIO_KNAPSACK_COVER_SEPARATION_TIME (20)

Separation time for knapsack cover.

MSK_DINF_MIO_OBJ_ABS_GAP (21)

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.

MSK_DINF_MIO_OBJ_BOUND (22)

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 *MSK_IINF_MIO_NUM_RELAX* is strictly positive.

MSK_DINF_MIO_OBJ_INT (23)

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 *MSK_IINF_MIO_NUM_INT_SOLUTIONS*.

MSK_DINF_MIO_OBJ_REL_GAP (24)

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 *MSK_DPAR_MIO_REL_GAP_CONST*. Otherwise it has the value -1.0.

MSK_DINF_MIO_PROBING_TIME (25)
Total time for probing.

MSK_DINF_MIO_ROOT_CUTGEN_TIME (26)
Total time for cut generation.

MSK_DINF_MIO_ROOT_OPTIMIZER_TIME (27)
Time spent in the optimizer while solving the root node relaxation

MSK_DINF_MIO_ROOT_PRESOLVE_TIME (28)
Time spent presolving the problem at the root node.

MSK_DINF_MIO_TIME (29)
Time spent in the mixed-integer optimizer.

MSK_DINF_MIO_USER_OBJ_CUT (30)
If the objective cut is used, then this information item has the value of the cut.

MSK_DINF_OPTIMIZER_TIME (31)
Total time spent in the optimizer since it was invoked.

MSK_DINF_PRESOLVE_ELI_TIME (32)
Total time spent in the eliminator since the presolve was invoked.

MSK_DINF_PRESOLVE_LINDEP_TIME (33)
Total time spent in the linear dependency checker since the presolve was invoked.

MSK_DINF_PRESOLVE_TIME (34)
Total time (in seconds) spent in the presolve since it was invoked.

MSK_DINF_PRIMAL_REPAIR_PENALTY_OBJ (35)
The optimal objective value of the penalty function.

MSK_DINF_QCQO_REFORMULATE_MAX_PERTURBATION (36)
Maximum absolute diagonal perturbation occurring during the QCQO reformulation.

MSK_DINF_QCQO_REFORMULATE_TIME (37)
Time spent with conic quadratic reformulation.

MSK_DINF_QCQO_REFORMULATE_WORST_CHOLESKY_COLUMN_SCALING (38)
Worst Cholesky column scaling.

MSK_DINF_QCQO_REFORMULATE_WORST_CHOLESKY_DIAG_SCALING (39)
Worst Cholesky diagonal scaling.

MSK_DINF_RD_TIME (40)
Time spent reading the data file.

MSK_DINF_SIM_DUAL_TIME (41)
Time spent in the dual simplex optimizer since invoking it.

MSK_DINF_SIM_FEAS (42)
Feasibility measure reported by the simplex optimizer.

MSK_DINF_SIM_OBJ (43)
Objective value reported by the simplex optimizer.

MSK_DINF_SIM_PRIMAL_TIME (44)
Time spent in the primal simplex optimizer since invoking it.

MSK_DINF_SIM_TIME (45)
Time spent in the simplex optimizer since invoking it.

MSK_DINF_SOL_BAS_DUAL_OBJ (46)
Dual objective value of the basic solution. Updated if *MSK_IPAR_AUTO_UPDATE_SOL_INFO* is set or by the method *MSK_updatesolutioninfo*.

MSK_DINF_SOL_BAS_DVIOLCON (47)
Maximal dual bound violation for x^c in the basic solution. Updated if *MSK_IPAR_AUTO_UPDATE_SOL_INFO* is set or by the method *MSK_updatesolutioninfo*.

MSK_DINF_SOL_BAS_DVIOLVAR (48)
Maximal dual bound violation for x^x in the basic solution. Updated if *MSK_IPAR_AUTO_UPDATE_SOL_INFO* is set or by the method *MSK_updatesolutioninfo*.

MSK_DINF_SOL_BAS_NRM_BARX (49)
Infinity norm of \bar{X} in the basic solution.

MSK_DINF_SOL_BAS_NRM_SLC (50)
Infinity norm of s_l^c in the basic solution.

MSK_DINF_SOL_BAS_NRM_SLX (51)
Infinity norm of s_l^x in the basic solution.

MSK_DINF_SOL_BAS_NRM_SUC (52)
Infinity norm of s_u^c in the basic solution.

MSK_DINF_SOL_BAS_NRM_SUX (53)
Infinity norm of s_u^X in the basic solution.

MSK_DINF_SOL_BAS_NRM_XC (54)
Infinity norm of x^c in the basic solution.

MSK_DINF_SOL_BAS_NRM_XX (55)
Infinity norm of x^x in the basic solution.

MSK_DINF_SOL_BAS_NRM_Y (56)
Infinity norm of y in the basic solution.

MSK_DINF_SOL_BAS_PRIMAL_OBJ (57)
Primal objective value of the basic solution. Updated if *MSK_IPAR_AUTO_UPDATE_SOL_INFO* is set or by the method *MSK_updatesolutioninfo*.

MSK_DINF_SOL_BAS_PVIOLCON (58)
Maximal primal bound violation for x^c in the basic solution. Updated if *MSK_IPAR_AUTO_UPDATE_SOL_INFO* is set or by the method *MSK_updatesolutioninfo*.

MSK_DINF_SOL_BAS_PVIOLVAR (59)
Maximal primal bound violation for x^x in the basic solution. Updated if *MSK_IPAR_AUTO_UPDATE_SOL_INFO* is set or by the method *MSK_updatesolutioninfo*.

MSK_DINF_SOL_ITG_NRM_BARX (60)
Infinity norm of \bar{X} in the integer solution.

MSK_DINF_SOL_ITG_NRM_XC (61)
Infinity norm of x^c in the integer solution.

MSK_DINF_SOL_ITG_NRM_XX (62)
Infinity norm of x^x in the integer solution.

MSK_DINF_SOL_ITG_PRIMAL_OBJ (63)
Primal objective value of the integer solution. Updated if *MSK_IPAR_AUTO_UPDATE_SOL_INFO* is set or by the method *MSK_updatesolutioninfo*.

MSK_DINF_SOL_ITG_PVIOLBARVAR (64)
Maximal primal bound violation for \bar{X} in the integer solution. Updated if *MSK_IPAR_AUTO_UPDATE_SOL_INFO* is set or by the method *MSK_updatesolutioninfo*.

MSK_DINF_SOL_ITG_PVIOLCON (65)
Maximal primal bound violation for x^c in the integer solution. Updated if *MSK_IPAR_AUTO_UPDATE_SOL_INFO* is set or by the method *MSK_updatesolutioninfo*.

MSK_DINF_SOL_ITG_PVIOLCONES (66)
Maximal primal violation for primal conic constraints in the integer solution. Updated if *MSK_IPAR_AUTO_UPDATE_SOL_INFO* is set or by the method *MSK_updatesolutioninfo*.

MSK_DINF_SOL_ITG_PVIOLITG (67)
Maximal violation for the integer constraints in the integer solution. Updated if *MSK_IPAR_AUTO_UPDATE_SOL_INFO* is set or by the method *MSK_updatesolutioninfo*.

MSK_DINF_SOL_ITG_PVIOLVAR (68)
Maximal primal bound violation for x^x in the integer solution. Updated if *MSK_IPAR_AUTO_UPDATE_SOL_INFO* is set or by the method *MSK_updatesolutioninfo*.

MSK_DINF_SOL_ITR_DUAL_OBJ (69)
Dual objective value of the interior-point solution. Updated if *MSK_IPAR_AUTO_UPDATE_SOL_INFO* is set or by the method *MSK_updatesolutioninfo*.

MSK_DINF_SOL_ITR_DVIOLBARVAR (70)
Maximal dual bound violation for \bar{X} in the interior-point solution. Updated if *MSK_IPAR_AUTO_UPDATE_SOL_INFO* is set or by the method *MSK_updatesolutioninfo*.

MSK_DINF_SOL_ITR_DVIOLCON (71)
Maximal dual bound violation for x^c in the interior-point solution. Updated if *MSK_IPAR_AUTO_UPDATE_SOL_INFO* is set or by the method *MSK_updatesolutioninfo*.

MSK_DINF_SOL_ITR_DVIOLCONES (72)
Maximal dual violation for dual conic constraints in the interior-point solution. Updated if *MSK_IPAR_AUTO_UPDATE_SOL_INFO* is set or by the method *MSK_updatesolutioninfo*.

MSK_DINF_SOL_ITR_DVIOLVAR (73)
Maximal dual bound violation for x^x in the interior-point solution. Updated if *MSK_IPAR_AUTO_UPDATE_SOL_INFO* is set or by the method *MSK_updatesolutioninfo*.

MSK_DINF_SOL_ITR_NRM_BARS (74)
Infinity norm of \bar{S} in the interior-point solution.

MSK_DINF_SOL_ITR_NRM_BARX (75)
Infinity norm of \bar{X} in the interior-point solution.

MSK_DINF_SOL_ITR_NRM_SLC (76)
Infinity norm of s_l^c in the interior-point solution.

MSK_DINF_SOL_ITR_NRM_SLX (77)
Infinity norm of s_l^x in the interior-point solution.

MSK_DINF_SOL_ITR_NRM_SNX (78)
Infinity norm of s_n^x in the interior-point solution.

MSK_DINF_SOL_ITR_NRM_SUC (79)
Infinity norm of s_u^c in the interior-point solution.

MSK_DINF_SOL_ITR_NRM_SUX (80)
Infinity norm of s_u^X in the interior-point solution.

MSK_DINF_SOL_ITR_NRM_XC (81)
Infinity norm of x^c in the interior-point solution.

MSK_DINF_SOL_ITR_NRM_XX (82)
Infinity norm of x^x in the interior-point solution.

MSK_DINF_SOL_ITR_NRM_Y (83)
Infinity norm of y in the interior-point solution.

MSK_DINF_SOL_ITR_PRIMAL_OBJ (84)
Primal objective value of the interior-point solution. Updated if *MSK_IPAR_AUTO_UPDATE_SOL_INFO* is set or by the method *MSK_updatesolutioninfo*.

MSK_DINF_SOL_ITR_PVIOLBARVAR (85)
Maximal primal bound violation for \bar{X} in the interior-point solution. Updated if *MSK_IPAR_AUTO_UPDATE_SOL_INFO* is set or by the method *MSK_updatesolutioninfo*.

MSK_DINF_SOL_ITR_PVIOLCON (86)
Maximal primal bound violation for x^c in the interior-point solution. Updated if *MSK_IPAR_AUTO_UPDATE_SOL_INFO* is set or by the method *MSK_updatesolutioninfo*.

MSK_DINF_SOL_ITR_PVIOLCONES (87)
Maximal primal violation for primal conic constraints in the interior-point solution. Updated if *MSK_IPAR_AUTO_UPDATE_SOL_INFO* is set or by the method *MSK_updatesolutioninfo*.

MSK_DINF_SOL_ITR_PVIOLVAR (88)
Maximal primal bound violation for x^x in the interior-point solution. Updated if *MSK_IPAR_AUTO_UPDATE_SOL_INFO* is set or by the method *MSK_updatesolutioninfo*.

MSK_DINF_TO_CONIC_TIME (89)
Time spent in the last to conic reformulation.

MSKfeatureee
License feature

MSK_FEATURE_PTS (0)
Base system.

MSK_FEATURE_PTON (1)
Conic extension.

MSKliinfiteme
Long integer information items.

MSK_LIINF_BI_CLEAN_DUAL_DEG_ITER (0)
Number of dual degenerate clean iterations performed in the basis identification.

MSK_LIINF_BI_CLEAN_DUAL_ITER (1)
Number of dual clean iterations performed in the basis identification.

MSK_LIINF_BI_CLEAN_PRIMAL_DEG_ITER (2)
Number of primal degenerate clean iterations performed in the basis identification.

MSK_LIINF_BI_CLEAN_PRIMAL_ITER (3)
Number of primal clean iterations performed in the basis identification.

MSK_LIINF_BI_DUAL_ITER (4)
Number of dual pivots performed in the basis identification.

MSK_LIINF_BI_PRIMAL_ITER (5)
Number of primal pivots performed in the basis identification.

MSK_LIINF_INTPNT_FACTOR_NUM_NZ (6)
Number of non-zeros in factorization.

MSK_LIINF_MIO_ANZ (7)
Number of non-zero entries in the constraint matrix of the problem to be solved by the mixed-integer optimizer.

MSK_LIINF_MIO_INTPNT_ITER (8)
Number of interior-point iterations performed by the mixed-integer optimizer.

MSK_LIINF_MIO_PRESOLVED_ANZ (9)
Number of non-zero entries in the constraint matrix of the problem after the mixed-integer optimizer's presolve.

MSK_LIINF_MIO_SIMPLEX_ITER (10)
Number of simplex iterations performed by the mixed-integer optimizer.

MSK_LIINF_RD_NUMANZ (11)
Number of non-zeros in A that is read.

MSK_LIINF_RD_NUMQNZ (12)
Number of Q non-zeros.

MSKiinfiteme
Integer information items.

MSK_IINF_ANA_PRO_NUM_CON (0)
Number of constraints in the problem. This value is set by *MSK_analyzeproblem*.

MSK_IINF_ANA_PRO_NUM_CON_EQ (1)
Number of equality constraints. This value is set by *MSK_analyzeproblem*.

MSK_IINF_ANA_PRO_NUM_CON_FR (2)
Number of unbounded constraints. This value is set by *MSK_analyzeproblem*.

MSK_IINF_ANA_PRO_NUM_CON_LO (3)
Number of constraints with a lower bound and an infinite upper bound. This value is set by *MSK_analyzeproblem*.

MSK_IINF_ANA_PRO_NUM_CON_RA (4)
Number of constraints with finite lower and upper bounds. This value is set by *MSK_analyzeproblem*.

MSK_IINF_ANA_PRO_NUM_CON_UP (5)
Number of constraints with an upper bound and an infinite lower bound. This value is set by *MSK_analyzeproblem*.

MSK_IINF_ANA_PRO_NUM_VAR (6)
 Number of variables in the problem. This value is set by *MSK_analyzeproblem*.

MSK_IINF_ANA_PRO_NUM_VAR_BIN (7)
 Number of binary (0-1) variables. This value is set by *MSK_analyzeproblem*.

MSK_IINF_ANA_PRO_NUM_VAR_CONT (8)
 Number of continuous variables. This value is set by *MSK_analyzeproblem*.

MSK_IINF_ANA_PRO_NUM_VAR_EQ (9)
 Number of fixed variables. This value is set by *MSK_analyzeproblem*.

MSK_IINF_ANA_PRO_NUM_VAR_FR (10)
 Number of free variables. This value is set by *MSK_analyzeproblem*.

MSK_IINF_ANA_PRO_NUM_VAR_INT (11)
 Number of general integer variables. This value is set by *MSK_analyzeproblem*.

MSK_IINF_ANA_PRO_NUM_VAR_LO (12)
 Number of variables with a lower bound and an infinite upper bound. This value is set by *MSK_analyzeproblem*.

MSK_IINF_ANA_PRO_NUM_VAR_RA (13)
 Number of variables with finite lower and upper bounds. This value is set by *MSK_analyzeproblem*.

MSK_IINF_ANA_PRO_NUM_VAR_UP (14)
 Number of variables with an upper bound and an infinite lower bound. This value is set by *MSK_analyzeproblem*.

MSK_IINF_INTPNT_FACTOR_DIM_DENSE (15)
 Dimension of the dense sub system in factorization.

MSK_IINF_INTPNT_ITER (16)
 Number of interior-point iterations since invoking the interior-point optimizer.

MSK_IINF_INTPNT_NUM_THREADS (17)
 Number of threads that the interior-point optimizer is using.

MSK_IINF_INTPNT_SOLVE_DUAL (18)
 Non-zero if the interior-point optimizer is solving the dual problem.

MSK_IINF_MIO_ABSGAP_SATISFIED (19)
 Non-zero if absolute gap is within tolerances.

MSK_IINF_MIO_CLIQUE_TABLE_SIZE (20)
 Size of the clique table.

MSK_IINF_MIO_CONSTRUCT_SOLUTION (21)
 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.

MSK_IINF_MIO_NODE_DEPTH (22)
 Depth of the last node solved.

MSK_IINF_MIO_NUM_ACTIVE_NODES (23)
 Number of active branch and bound nodes.

MSK_IINF_MIO_NUM_BRANCH (24)
 Number of branches performed during the optimization.

MSK_IINF_MIO_NUM_CLIQUE_CUTS (25)
 Number of clique cuts.

MSK_IINF_MIO_NUM_CMIR_CUTS (26)
 Number of Complemented Mixed Integer Rounding (CMIR) cuts.

MSK_IINF_MIO_NUM_GOMORY_CUTS (27)
 Number of Gomory cuts.

MSK_IINF_MIO_NUM_IMPLIED_BOUND_CUTS (28)
 Number of implied bound cuts.

MSK_IINF_MIO_NUM_INT_SOLUTIONS (29)
 Number of integer feasible solutions that have been found.

MSK_IINF_MIO_NUM_KNAPSACK_COVER_CUTS (30)
 Number of clique cuts.

MSK_IINF_MIO_NUM_RELAX (31)
 Number of relaxations solved during the optimization.

MSK_IINF_MIO_NUM_REPEATED_PRESOLVE (32)
 Number of times presolve was repeated at root.

MSK_IINF_MIO_NUMBIN (33)
 Number of binary variables in the problem to be solved by the mixed-integer optimizer.

MSK_IINF_MIO_NUMBINCONEVAR (34)
 Number of binary cone variables in the problem to be solved by the mixed-integer optimizer.

MSK_IINF_MIO_NUMCON (35)
 Number of constraints in the problem to be solved by the mixed-integer optimizer.

MSK_IINF_MIO_NUMCONE (36)
 Number of cones in the problem to be solved by the mixed-integer optimizer.

MSK_IINF_MIO_NUMCONEVAR (37)
 Number of cone variables in the problem to be solved by the mixed-integer optimizer.

MSK_IINF_MIO_NUMCONT (38)
 Number of continuous variables in the problem to be solved by the mixed-integer optimizer.

MSK_IINF_MIO_NUMCONTCONEVAR (39)
 Number of continuous cone variables in the problem to be solved by the mixed-integer optimizer.

MSK_IINF_MIO_NUMDEXPCONES (40)
 Number of dual exponential cones in the problem to be solved by the mixed-integer optimizer.

MSK_IINF_MIO_NUMDPOWCONES (41)
 Number of dual power cones in the problem to be solved by the mixed-integer optimizer.

MSK_IINF_MIO_NUMINT (42)
 Number of integer variables in the problem to be solved by the mixed-integer optimizer.

MSK_IINF_MIO_NUMINTCONEVAR (43)
 Number of integer cone variables in the problem to be solved by the mixed-integer optimizer.

MSK_IINF_MIO_NUMPEXPONES (44)
 Number of primal exponential cones in the problem to be solved by the mixed-integer optimizer.

MSK_IINF_MIO_NUMPPOWCONES (45)
 Number of primal power cones in the problem to be solved by the mixed-integer optimizer.

MSK_IINF_MIO_NUMQCONES (46)
 Number of quadratic cones in the problem to be solved by the mixed-integer optimizer.

MSK_IINF_MIO_NUMRQCONES (47)
 Number of rotated quadratic cones in the problem to be solved by the mixed-integer optimizer.

MSK_IINF_MIO_NUMVAR (48)
 Number of variables in the problem to be solved by the mixed-integer optimizer.

MSK_IINF_MIO_OBJ_BOUND_DEFINED (49)
 Non-zero if a valid objective bound has been found, otherwise zero.

MSK_IINF_MIO_PRESOLVED_NUMBIN (50)
 Number of binary variables in the problem after the mixed-integer optimizer's presolve.

MSK_IINF_MIO_PRESOLVED_NUMBINCONEVAR (51)
 Number of binary cone variables in the problem after the mixed-integer optimizer's presolve.

MSK_IINF_MIO_PRE SOLVED_NUMCON (52)
Number of constraints in the problem after the mixed-integer optimizer’s presolve.

MSK_IINF_MIO_PRE SOLVED_NUMCONE (53)
Number of cones in the problem after the mixed-integer optimizer’s presolve.

MSK_IINF_MIO_PRE SOLVED_NUMCONEVAR (54)
Number of cone variables in the problem after the mixed-integer optimizer’s presolve.

MSK_IINF_MIO_PRE SOLVED_NUMCONT (55)
Number of continuous variables in the problem after the mixed-integer optimizer’s presolve.

MSK_IINF_MIO_PRE SOLVED_NUMCONTCONEVAR (56)
Number of continuous cone variables in the problem after the mixed-integer optimizer’s presolve.

MSK_IINF_MIO_PRE SOLVED_NUMDEXPCONES (57)
Number of dual exponential cones in the problem after the mixed-integer optimizer’s presolve.

MSK_IINF_MIO_PRE SOLVED_NUMDPOWCONES (58)
Number of dual power cones in the problem after the mixed-integer optimizer’s presolve.

MSK_IINF_MIO_PRE SOLVED_NUMINT (59)
Number of integer variables in the problem after the mixed-integer optimizer’s presolve.

MSK_IINF_MIO_PRE SOLVED_NUMINTCONEVAR (60)
Number of integer cone variables in the problem after the mixed-integer optimizer’s presolve.

MSK_IINF_MIO_PRE SOLVED_NUMPEXP CONES (61)
Number of primal exponential cones in the problem after the mixed-integer optimizer’s presolve.

MSK_IINF_MIO_PRE SOLVED_NUMPPOWCONES (62)
Number of primal power cones in the problem after the mixed-integer optimizer’s presolve.

MSK_IINF_MIO_PRE SOLVED_NUMQCONES (63)
Number of quadratic cones in the problem after the mixed-integer optimizer’s presolve.

MSK_IINF_MIO_PRE SOLVED_NUMRQCONES (64)
Number of rotated quadratic cones in the problem after the mixed-integer optimizer’s presolve.

MSK_IINF_MIO_PRE SOLVED_NUMVAR (65)
Number of variables in the problem after the mixed-integer optimizer’s presolve.

MSK_IINF_MIO_RELGAP_SATISFIED (66)
Non-zero if relative gap is within tolerances.

MSK_IINF_MIO_TOTAL_NUM_CUTS (67)
Total number of cuts generated by the mixed-integer optimizer.

MSK_IINF_MIO_USER_OBJ_CUT (68)
If it is non-zero, then the objective cut is used.

MSK_IINF_OPT_NUMCON (69)
Number of constraints in the problem solved when the optimizer is called.

MSK_IINF_OPT_NUMVAR (70)
Number of variables in the problem solved when the optimizer is called

MSK_IINF_OPTIMIZE_RESPONSE (71)
The response code returned by optimize.

MSK_IINF_PURIFY_DUAL_SUCCESS (72)
Is nonzero if the dual solution is purified.

MSK_IINF_PURIFY_PRIMAL_SUCCESS (73)
Is nonzero if the primal solution is purified.

MSK_IINF_RD_NUMBARVAR (74)
Number of symmetric variables read.

MSK_IINF_RD_NUMCON (75)
Number of constraints read.

MSK_IINF_RD_NUMCONE (76)
Number of conic constraints read.

MSK_IINF_RD_NUMINTVAR (77)
Number of integer-constrained variables read.

MSK_IINF_RD_NUMQ (78)
Number of nonempty Q matrices read.

MSK_IINF_RD_NUMVAR (79)
Number of variables read.

MSK_IINF_RD_PROTOTYPE (80)
Problem type.

MSK_IINF_SIM_DUAL_DEG_ITER (81)
The number of dual degenerate iterations.

MSK_IINF_SIM_DUAL_HOTSTART (82)
If 1 then the dual simplex algorithm is solving from an advanced basis.

MSK_IINF_SIM_DUAL_HOTSTART_LU (83)
If 1 then a valid basis factorization of full rank was located and used by the dual simplex algorithm.

MSK_IINF_SIM_DUAL_INF_ITER (84)
The number of iterations taken with dual infeasibility.

MSK_IINF_SIM_DUAL_ITER (85)
Number of dual simplex iterations during the last optimization.

MSK_IINF_SIM_NUMCON (86)
Number of constraints in the problem solved by the simplex optimizer.

MSK_IINF_SIM_NUMVAR (87)
Number of variables in the problem solved by the simplex optimizer.

MSK_IINF_SIM_PRIMAL_DEG_ITER (88)
The number of primal degenerate iterations.

MSK_IINF_SIM_PRIMAL_HOTSTART (89)
If 1 then the primal simplex algorithm is solving from an advanced basis.

MSK_IINF_SIM_PRIMAL_HOTSTART_LU (90)
If 1 then a valid basis factorization of full rank was located and used by the primal simplex algorithm.

MSK_IINF_SIM_PRIMAL_INF_ITER (91)
The number of iterations taken with primal infeasibility.

MSK_IINF_SIM_PRIMAL_ITER (92)
Number of primal simplex iterations during the last optimization.

MSK_IINF_SIM_SOLVE_DUAL (93)
Is non-zero if dual problem is solved.

MSK_IINF_SOL_BAS_PROSTA (94)
Problem status of the basic solution. Updated after each optimization.

MSK_IINF_SOL_BAS_SOLSTA (95)
Solution status of the basic solution. Updated after each optimization.

MSK_IINF_SOL_ITG_PROSTA (96)
Problem status of the integer solution. Updated after each optimization.

MSK_IINF_SOL_ITG_SOLSTA (97)
Solution status of the integer solution. Updated after each optimization.

MSK_IINF_SOL_ITR_PROSTA (98)
Problem status of the interior-point solution. Updated after each optimization.

MSK_IINF_SOL_ITR_SOLSTA (99)
Solution status of the interior-point solution. Updated after each optimization.

MSK_IINF_STO_NUM_A_REALLOC (100)
 Number of times the storage for storing A has been changed. A large value may indicate that memory fragmentation may occur.

MSKinfypee
 Information item types

MSK_INF_DOU_TYPE (0)
 Is a double information type.

MSK_INF_INT_TYPE (1)
 Is an integer.

MSK_INF_LINT_TYPE (2)
 Is a long integer.

MSKiomodee
 Input/output modes

MSK_IOMODE_READ (0)
 The file is read-only.

MSK_IOMODE_WRITE (1)
 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.

MSK_IOMODE_READWRITE (2)
 The file is to read and write.

MSKbranchdire
 Specifies the branching direction.

MSK_BRANCH_DIR_FREE (0)
 The mixed-integer optimizer decides which branch to choose.

MSK_BRANCH_DIR_UP (1)
 The mixed-integer optimizer always chooses the up branch first.

MSK_BRANCH_DIR_DOWN (2)
 The mixed-integer optimizer always chooses the down branch first.

MSK_BRANCH_DIR_NEAR (3)
 Branch in direction nearest to selected fractional variable.

MSK_BRANCH_DIR_FAR (4)
 Branch in direction farthest from selected fractional variable.

MSK_BRANCH_DIR_ROOT_LP (5)
 Chose direction based on root lp value of selected variable.

MSK_BRANCH_DIR_GUIDED (6)
 Branch in direction of current incumbent.

MSK_BRANCH_DIR_PSEUDOCOST (7)
 Branch based on the pseudocost of the variable.

MSKmiocontsoltypee
 Continuous mixed-integer solution type

MSK_MIO_CONT_SOL_NONE (0)
 No interior-point or basic solution are reported when the mixed-integer optimizer is used.

MSK_MIO_CONT_SOL_ROOT (1)
 The reported interior-point and basic solutions are a solution to the root node problem when mixed-integer optimizer is used.

MSK_MIO_CONT_SOL_ITG (2)
 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.

MSK_MIO_CONT_SOL_ITG_REL (3)
 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.

MSKmiomodee

Integer restrictions

MSK_MIO_MODE_IGNORED (0)

The integer constraints are ignored and the problem is solved as a continuous problem.

MSK_MIO_MODE_SATISFIED (1)

Integer restrictions should be satisfied.

MSKmionodeseltypee

Mixed-integer node selection types

MSK_MIO_NODE_SELECTION_FREE (0)

The optimizer decides the node selection strategy.

MSK_MIO_NODE_SELECTION_FIRST (1)

The optimizer employs a depth first node selection strategy.

MSK_MIO_NODE_SELECTION_BEST (2)

The optimizer employs a best bound node selection strategy.

MSK_MIO_NODE_SELECTION_PSEUDO (3)

The optimizer employs selects the node based on a pseudo cost estimate.

MSKmpsformate

MPS file format type

MSK_MPS_FORMAT_STRICT (0)

It is assumed that the input file satisfies the MPS format strictly.

MSK_MPS_FORMAT_RELAXED (1)

It is assumed that the input file satisfies a slightly relaxed version of the MPS format.

MSK_MPS_FORMAT_FREE (2)

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.

MSK_MPS_FORMAT_CPLEX (3)

The CPLEX compatible version of the MPS format is employed.

MSKobjsensee

Objective sense types

MSK_OBJECTIVE_SENSE_MINIMIZE (0)

The problem should be minimized.

MSK_OBJECTIVE_SENSE_MAXIMIZE (1)

The problem should be maximized.

MSKonoffkeye

On/off

MSK_ON (1)

Switch the option on.

MSK_OFF (0)

Switch the option off.

MSKoptimizertypee

Optimizer types

MSK_OPTIMIZER_CONIC (0)

The optimizer for problems having conic constraints.

MSK_OPTIMIZER_DUAL_SIMPLEX (1)

The dual simplex optimizer is used.

MSK_OPTIMIZER_FREE (2)

The optimizer is chosen automatically.

MSK_OPTIMIZER_FREE_SIMPLEX (3)

One of the simplex optimizers is used.

MSK_OPTIMIZER_INTPNT (4)
The interior-point optimizer is used.

MSK_OPTIMIZER_MIXED_INT (5)
The mixed-integer optimizer.

MSK_OPTIMIZER_PRIMAL_SIMPLEX (6)
The primal simplex optimizer is used.

MSKorderingtypee
Ordering strategies

MSK_ORDER_METHOD_FREE (0)
The ordering method is chosen automatically.

MSK_ORDER_METHOD_APPMINLOC (1)
Approximate minimum local fill-in ordering is employed.

MSK_ORDER_METHOD_EXPERIMENTAL (2)
This option should not be used.

MSK_ORDER_METHOD_TRY_GRAPHPAR (3)
Always try the graph partitioning based ordering.

MSK_ORDER_METHOD_FORCE_GRAPHPAR (4)
Always use the graph partitioning based ordering even if it is worse than the approximate minimum local fill ordering.

MSK_ORDER_METHOD_NONE (5)
No ordering is used.

MSKpresolvemodee
Presolve method.

MSK_PRESOLVE_MODE_OFF (0)
The problem is not presolved before it is optimized.

MSK_PRESOLVE_MODE_ON (1)
The problem is presolved before it is optimized.

MSK_PRESOLVE_MODE_FREE (2)
It is decided automatically whether to presolve before the problem is optimized.

MSKparametertypee
Parameter type

MSK_PAR_INVALID_TYPE (0)
Not a valid parameter.

MSK_PAR_DOUB_TYPE (1)
Is a double parameter.

MSK_PAR_INT_TYPE (2)
Is an integer parameter.

MSK_PAR_STR_TYPE (3)
Is a string parameter.

MSKproblemiteme
Problem data items

MSK_PI_VAR (0)
Item is a variable.

MSK_PI_CON (1)
Item is a constraint.

MSK_PI_CONE (2)
Item is a cone.

MSKproblemtypede
Problem types

MSK_PROBTYPE_LO (0)
The problem is a linear optimization problem.

MSK_PROBTYPE_QQ (1)
The problem is a quadratic optimization problem.

MSK_PROBTYPE_QCQQ (2)
The problem is a quadratically constrained optimization problem.

MSK_PROBTYPE_CONIC (3)
A conic optimization.

MSK_PROBTYPE_MIXED (4)
General nonlinear constraints and conic constraints. This combination can not be solved by **MOSEK**.

MSKprostaes
Problem status keys

MSK_PRO_STA_UNKNOWN (0)
Unknown problem status.

MSK_PRO_STA_PRIM_AND_DUAL_FEAS (1)
The problem is primal and dual feasible.

MSK_PRO_STA_PRIM_FEAS (2)
The problem is primal feasible.

MSK_PRO_STA_DUAL_FEAS (3)
The problem is dual feasible.

MSK_PRO_STA_PRIM_INFEAS (4)
The problem is primal infeasible.

MSK_PRO_STA_DUAL_INFEAS (5)
The problem is dual infeasible.

MSK_PRO_STA_PRIM_AND_DUAL_INFEAS (6)
The problem is primal and dual infeasible.

MSK_PRO_STA_ILL_POSED (7)
The problem is ill-posed. For example, it may be primal and dual feasible but have a positive duality gap.

MSK_PRO_STA_PRIM_INFEAS_OR_UNBOUNDED (8)
The problem is either primal infeasible or unbounded. This may occur for mixed-integer problems.

MSKxmlwriteroutputtypee
XML writer output mode

MSK_WRITE_XML_MODE_ROW (0)
Write in row order.

MSK_WRITE_XML_MODE_COL (1)
Write in column order.

MSKrescodetypee
Response code type

MSK_RESPONSE_OK (0)
The response code is OK.

MSK_RESPONSE_WRN (1)
The response code is a warning.

MSK_RESPONSE_TRM (2)
The response code is an optimizer termination status.

MSK_RESPONSE_ERR (3)
The response code is an error.

MSK_RESPONSE_UNK (4)
The response code does not belong to any class.

MSKscalingtypee
Scaling type

MSK_SCALING_FREE (0)
The optimizer chooses the scaling heuristic.

MSK_SCALING_NONE (1)
No scaling is performed.

MSK_SCALING_MODERATE (2)
A conservative scaling is performed.

MSK_SCALING_AGGRESSIVE (3)
A very aggressive scaling is performed.

MSKscalingmethode
Scaling method

MSK_SCALING_METHOD_POW2 (0)
Scales only with power of 2 leaving the mantissa untouched.

MSK_SCALING_METHOD_FREE (1)
The optimizer chooses the scaling heuristic.

MSKsensitivitytypee
Sensitivity types

MSK_SENSITIVITY_TYPE_BASIS (0)
Basis sensitivity analysis is performed.

MSKsimseltypee
Simplex selection strategy

MSK_SIM_SELECTION_FREE (0)
The optimizer chooses the pricing strategy.

MSK_SIM_SELECTION_FULL (1)
The optimizer uses full pricing.

MSK_SIM_SELECTION_ASE (2)
The optimizer uses approximate steepest-edge pricing.

MSK_SIM_SELECTION_DEVEX (3)
The optimizer uses devex steepest-edge pricing (or if it is not available an approximate steep-edge selection).

MSK_SIM_SELECTION_SE (4)
The optimizer uses steepest-edge selection (or if it is not available an approximate steep-edge selection).

MSK_SIM_SELECTION_PARTIAL (5)
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.

MSKsoliteme
Solution items

MSK_SOL_ITEM_XC (0)
Solution for the constraints.

MSK_SOL_ITEM_XX (1)
Variable solution.

MSK_SOL_ITEM_Y (2)
Lagrange multipliers for equations.

MSK_SOL_ITEM_SLC (3)
Lagrange multipliers for lower bounds on the constraints.

MSK_SOL_ITEM_SUC (4)
Lagrange multipliers for upper bounds on the constraints.

MSK_SOL_ITEM_SLX (5)
Lagrange multipliers for lower bounds on the variables.

MSK_SOL_ITEM_SUX (6)
Lagrange multipliers for upper bounds on the variables.

MSK_SOL_ITEM_SNX (7)
 Lagrange multipliers corresponding to the conic constraints on the variables.

MSKsolstae
 Solution status keys

MSK_SOL_STA_UNKNOWN (0)
 Status of the solution is unknown.

MSK_SOL_STA_OPTIMAL (1)
 The solution is optimal.

MSK_SOL_STA_PRIM_FEAS (2)
 The solution is primal feasible.

MSK_SOL_STA_DUAL_FEAS (3)
 The solution is dual feasible.

MSK_SOL_STA_PRIM_AND_DUAL_FEAS (4)
 The solution is both primal and dual feasible.

MSK_SOL_STA_PRIM_INFEAS_CER (5)
 The solution is a certificate of primal infeasibility.

MSK_SOL_STA_DUAL_INFEAS_CER (6)
 The solution is a certificate of dual infeasibility.

MSK_SOL_STA_PRIM_ILLPOSED_CER (7)
 The solution is a certificate that the primal problem is illposed.

MSK_SOL_STA_DUAL_ILLPOSED_CER (8)
 The solution is a certificate that the dual problem is illposed.

MSK_SOL_STA_INTEGER_OPTIMAL (9)
 The primal solution is integer optimal.

MSKsoltypee
 Solution types

MSK_SOL_BAS (1)
 The basic solution.

MSK_SOL_ITR (0)
 The interior solution.

MSK_SOL_ITG (2)
 The integer solution.

MSKsolveforme
 Solve primal or dual form

MSK_SOLVE_FREE (0)
 The optimizer is free to solve either the primal or the dual problem.

MSK_SOLVE_PRIMAL (1)
 The optimizer should solve the primal problem.

MSK_SOLVE_DUAL (2)
 The optimizer should solve the dual problem.

MSKstakeye
 Status keys

MSK_SK_UNK (0)
 The status for the constraint or variable is unknown.

MSK_SK_BAS (1)
 The constraint or variable is in the basis.

MSK_SK_SUPBAS (2)
 The constraint or variable is super basic.

MSK_SK_LOW (3)
 The constraint or variable is at its lower bound.

MSK_SK_UPR (4)
 The constraint or variable is at its upper bound.

MSK_SK_FIX (5)
The constraint or variable is fixed.

MSK_SK_INF (6)
The constraint or variable is infeasible in the bounds.

MSKstartpointtypee
Starting point types

MSK_STARTING_POINT_FREE (0)
The starting point is chosen automatically.

MSK_STARTING_POINT_GUESS (1)
The optimizer guesses a starting point.

MSK_STARTING_POINT_CONSTANT (2)
The optimizer constructs a starting point by assigning a constant value to all primal and dual variables. This starting point is normally robust.

MSK_STARTING_POINT_SATISFY_BOUNDS (3)
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.

MSKstreamtypee
Stream types

MSK_STREAM_LOG (0)
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.

MSK_STREAM_MSG (1)
Message stream. Log information relating to performance and progress of the optimization is written to this stream.

MSK_STREAM_ERR (2)
Error stream. Error messages are written to this stream.

MSK_STREAM_WRN (3)
Warning stream. Warning messages are written to this stream.

MSKvaluee
Integer values

MSK_MAX_STR_LEN (1024)
Maximum string length allowed in **MOSEK**.

MSK_LICENSE_BUFFER_LENGTH (21)
The length of a license key buffer.

MSKvariabletypee
Variable types

MSK_VAR_TYPE_CONT (0)
Is a continuous variable.

MSK_VAR_TYPE_INT (1)
Is an integer variable.

15.8 Data Types

MSKenv_t
The **MOSEK** Environment type.

MSKtask_t
The **MOSEK** Task type.

MSKuserhandle_t
A pointer to a user-defined structure.

MSKboolean_t
A signed integer interpreted as a boolean value.

MSKint32t
Signed 32bit integer.

MSKint64t

Signed 64bit integer.

MSKwchar_t

Wide char type. The actual type may differ depending on the platform; it is either a 16 or 32 bits signed or unsigned integer.

MSKrealt

The floating point type used by **MOSEK**.

MSKstring_t

The string type used by **MOSEK**. This is an UTF-8 encoded zero-terminated char string.

15.9 Function Types

MSKcallbackfunc

```
MSKint32t (MSKAPI * MSKcallbackfunc) (  
    MSKtask_t task,  
    MSKuserhandle_t usrptr,  
    MSKcallbackcodee caller,  
    const MSKrealt * douinf,  
    const MSKint32t * intinf,  
    const MSKint64t * lintinf)
```

The progress callback function 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 *MSK_IPAR_LOG_SIM_FREQ* controls how frequently the callback is called. The callback provides an code denoting the point in the solver from which the call happened, and a set of arrays containing information items related to the current state of the solver. Typically the user-defined callback function displays information about the solution process. The callback function can also be used to terminate the optimization process by returning a non-zero value.

The user *must not* call any **MOSEK** function directly or indirectly from the callback function. The only exception is the possibility to retrieve a current best integer solution from the mixed-integer optimizer, see Section *Progress and data callback*.

Parameters

- **task** (*MSKtask_t*) – An optimization task. (input)
- **usrptr** (*MSKuserhandle_t*) – A pointer to a user-defined structure. (input/output)
- **caller** (*MSKcallbackcodee*) – The caller key indicating the current progress of the solver. (input)
- **douinf** (*MSKrealt**) – An array of double information items. The elements correspond to the definitions in *MSKdinfiteme*. (input)
- **intinf** (*MSKint32t**) – An array of integer information items. The elements correspond to the definitions in *MSKiinfiteme*. (input)
- **lintinf** (*MSKint64t**) – An array of long information items. The elements correspond to the definitions in *MSKliinfiteme*. (input)

Return (*MSKint32t*) – If the return value is non-zero, **MOSEK** terminates whatever it is doing and returns control to the calling application.

MSKcallocfunc

```
void * (MSKAPI * MSKcallocfunc) (  
    MSKuserhandle_t usrptr,  
    const size_t num,  
    const size_t size)
```

A user-defined memory allocation function. The function must be compatible with the C `calloc` function.

Parameters

- `usrptr` (*MSKuserhandle_t*) – A pointer to a user-defined structure. (input)
- `num` (`size_t`) – The number of elements. (input)
- `size` (`size_t`) – The size of an element. (input)

Return (`void*`) – A pointer to the allocated memory.

MSKexitfunc

```
void (MSKAPI * MSKexitfunc) (  
    MSKuserhandle_t usrptr,  
    const char * file,  
    MSKint32t line,  
    const char * msg)
```

A user-defined exit function which is called in case of fatal errors to handle an error message and terminate the program. The function should never return.

Parameters

- `usrptr` (*MSKuserhandle_t*) – A pointer to a user-defined structure. (input/output)
- `file` (*MSKstring_t*) – The name of the file where the fatal error occurred. (input)
- `line` (*MSKint32t*) – The line number in the file where the fatal error occurred. (input)
- `msg` (*MSKstring_t*) – A message about the error. (input)

Return (`void`)

MSKfreefunc

```
void (MSKAPI * MSKfreefunc) (  
    MSKuserhandle_t usrptr,  
    void * buffer)
```

A user-defined memory freeing function.

Parameters

- `usrptr` (*MSKuserhandle_t*) – A pointer to a user-defined structure. (input)
- `buffer` (`void*`) – A pointer to the buffer which should be freed. (input/output)

Return (`void`)

MSKhreadfunc

```
size_t (MSKAPI * MSKhreadfunc) (  
    MSKuserhandle_t handle,  
    void * dest,  
    const size_t count)
```

Behaves similarly to system read function. Returns the number of bytes read.

Parameters

- `handle` (*MSKuserhandle_t*) – A pointer to a user-defined data structure (or a null pointer). (input/output)
- `dest` (`void*`) – Read into this destination (output)
- `count` (`size_t`) – Number of bytes to read. (input)

Return (`size_t`) – The function response code.

MSKwritefunc

```
size_t (MSKAPI * MSKwritefunc) (  
    MSKuserhandle_t handle,  
    void * src,  
    const size_t count)
```

Behaves similarly to system read function. Returns the number of bytes written.

Parameters

- `handle` (*MSKuserhandle_t*) – A pointer to a user-defined data structure (or a null pointer). (input/output)
- `src` (`void*`) – Write from this location (output)
- `count` (`size_t`) – Number of bytes to write. (input)

Return (`size_t`) – The function response code.

MSKmallocfunc

```
void * (MSKAPI * MSKmallocfunc) (  
    MSKuserhandle_t usrptr,  
    const size_t size)
```

A user-defined memory allocation function. The function must be compatible with the C `malloc` function.

Parameters

- `usrptr` (*MSKuserhandle_t*) – A pointer to a user-defined structure. (input)
- `size` (`size_t`) – The number of characters to allocate. (input)

Return (`void*`) – A pointer to the allocated memory.

MSKreallocfunc

```
void * (MSKAPI * MSKreallocfunc) (  
    MSKuserhandle_t usrptr,  
    void * ptr,  
    const size_t size)
```

A user-defined memory reallocation function. The function must be compatible with the C `realloc` function.

Parameters

- `usrptr` (*MSKuserhandle_t*) – A pointer to a user-defined structure. (input)
- `ptr` (`void*`) – The pointer to the allocated memory. (input/output)
- `size` (`size_t`) – Size of the new block. (input)

Return (`void*`) – A pointer to the allocated memory.

MSKresponsefunc

```
MSKrescodee (MSKAPI * MSKresponsefunc) (  
    MSKuserhandle_t handle,  
    MSKrescodee r,  
    const char * msg)
```

Whenever **MOSEK** generates a warning or an error this function is called. The argument `r` contains the code of the error/warning and the argument `msg` contains the corresponding error/warning message. This function should always return *MSK_RES_OK*.

Parameters

- `handle` (*MSKuserhandle_t*) – A pointer to a user-defined data structure (or a null pointer). (input/output)
- `r` (*MSKrescodee*) – The response code corresponding to the exception. (input)
- `msg` (*MSKstring_t*) – A string containing the exception message. (input)

Return (*MSKrescodee*) – The function response code.

MSKstreamfunc

```
void (MSKAPI * MSKstreamfunc) (  
    MSKuserhandle_t handle,  
    const char * str)
```

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

- `handle` (*MSKuserhandle_t*) – A pointer to a user-defined data structure (or a null pointer). (input/output)
- `str` (*MSKstring_t*) – A string containing a message to a stream. (input)

Return (void)

15.10 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 $t \geq fx \log x$ 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 (EXPopt), 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

$$\begin{array}{ll} \text{minimize/maximize} & c^T x + \frac{1}{2} q^o(x) \\ \text{subject to} & l^c \leq Ax + \frac{1}{2} q(x) \leq u^c, \\ & l^x \leq x \leq u^x, \\ & x_{\mathcal{J}} \text{ integer,} \end{array}$$

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.

$$c^T x + \frac{1}{2} x^T Q^o x.$$

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 `<=`) may be any of `<`, `<=`, `=`, `>`, `>=` (`<` and `<=` 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.

$$-5 \leq x_1 + x_2 \leq 5 \tag{16.1}$$

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 `mil01.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 `_XXXXXXXX`, 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 `MSK_IPAR_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 `MSK_IPAR_READ_LP_QUOTED_NAMES` and `MSK_IPAR_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 `MSK_IPAR_WRITE_LP_STRICT_FORMAT` to `MSK_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 `MSK_IPAR_WRITE_GENERIC_NAMES` to `MSK_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

- `MSK_IPAR_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.
- `MSK_IPAR_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

$$\begin{aligned} & \text{maximize/minimize} && c^T x + q_0(x) \\ & l^c \leq && Ax + q(x) \leq u^c, \\ & l^x \leq && x \leq u^x, \\ & && x \in \mathcal{K}, \\ & && x_{\mathcal{J}} \text{ integer}, \end{aligned} \tag{16.2}$$

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
    [vname1]  [vname2]  [value1]          [vname3]  [value2]
QMATRIX
    [vname1]  [vname2]  [value1]
```

(continues on next page)

```

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

```
[+|-]XXXXXXX.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

```

(continues on next page)

(continued from previous page)

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

$$\begin{aligned}
 &\text{minimize} && -x_2 + \frac{1}{2}(2x_1^2 - 2x_1x_3 + 0.2x_2^2 + 2x_3^2) \\
 &\text{subject to} && x_1 + x_2 + x_3 \geq 1, \\
 &&& x \geq 0
 \end{aligned}$$

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

(continues on next page)

(continued from previous page)

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

$$\begin{aligned}
 &\text{minimize} && -x_2 + \frac{1}{2}(2x_1^2 - 2x_1x_3 + 0.2x_2^2 + 2x_3^2) \\
 &\text{subject to} && x_1 + x_2 + x_3 \geq 1, \\
 &&& x \geq 0
 \end{aligned}$$

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

(continues on next page)

(continued from previous page)

	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	
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.

QCMATRIX (optional)

A QCMATRIX section allows to specify the quadratic part of a given constraint. Within the QCMATRIX 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
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
*23456789012345678901234567890123456789012345678901234567890						
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 `MSK_IPAR_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 `MSK_IPAR_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>MSK_IPAR_OPF_WRITE_SOL_BAS</i>	Include basic solution, if defined.
<i>MSK_IPAR_OPF_WRITE_SOL_ITG</i>	Include integer solution, if defined.
<i>MSK_IPAR_OPF_WRITE_SOL_ITR</i>	Include interior solution, if defined.
<i>MSK_IPAR_OPF_WRITE_SOLUTIONS</i>	Include solutions if they are defined. If this is off, no solutions are included.
<i>MSK_IPAR_OPF_WRITE_HEADER</i>	Include a small header with comments.
<i>MSK_IPAR_OPF_WRITE_PROBLEM</i>	Include the problem itself — objective, constraints and bounds.
<i>MSK_IPAR_OPF_WRITE_PARAMETERS</i>	Include all parameter settings.
<i>MSK_IPAR_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:

$$\begin{aligned}
 &\text{maximize} && 3x_0 &+& 1x_1 &+& 5x_2 &+& 1x_3 \\
 &\text{subject to} && 3x_0 &+& 1x_1 &+& 2x_2 && = 30, \\
 &&& 2x_0 &+& 1x_1 &+& 3x_2 &+& 1x_3 \geq 15, \\
 &&& && 2x_1 && &+& 3x_3 \leq 25,
 \end{aligned}$$

having the bounds

$$\begin{aligned}
 0 &\leq x_0 \leq \infty, \\
 0 &\leq x_1 \leq 10, \\
 0 &\leq x_2 \leq \infty, \\
 0 &\leq x_3 \leq \infty.
 \end{aligned}$$

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]

[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]

```

(continues on next page)

```
[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]

[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:  $x_4 \geq \sqrt{x_1^2 + x_2^2}$ 
  [cone quad 'k1'] x4, x1, x2 [/cone]

  # Define rotated quadratic cone:  $2 x_5 x_6 \geq x_3^2$ 
  [cone rquad 'k2'] x5, x6, x3 [/cone]
[/bounds]
```

Mixed Integer Example milo1.opf

Consider the mixed integer problem:

$$\begin{aligned} & \text{maximize} && x_0 + 0.64x_1 \\ & \text{subject to} && 50x_0 + 31x_1 \leq 250, \\ & && 3x_0 - 2x_1 \geq -4, \\ & && x_0, x_1 \geq 0 \quad \text{and integer} \end{aligned}$$

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]
```

(continues on next page)

```

[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:

$$\begin{aligned}
 & \min / \max && g^{obj} \\
 \text{s.t.} &&& g_i \in \mathcal{K}_i, \quad i \in \mathcal{I}, \\
 &&& G_i \in \mathcal{K}_i, \quad i \in \mathcal{I}^{PSD}, \\
 &&& x_j \in \mathcal{K}_j, \quad j \in \mathcal{J}, \\
 &&& \overline{X}_j \in \mathcal{K}_j, \quad j \in \mathcal{J}^{PSD}.
 \end{aligned} \tag{16.10}$$

- **Variables** are either scalar variables, x_j for $j \in \mathcal{J}$, or variables, \overline{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.

OBJCOORD

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.

ACCOORD

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

(continues on next page)

(continued from previous page)

```

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

```

(continues on next page)

```

# Three scalar variables in this one conic domain:
#   | Three are free.
VAR
3 1
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  $F^{\text{obj}}_j$  coefficients:
#   |  $F^{\text{obj}}[0][0,0] = 2.0$ 
#   |  $F^{\text{obj}}[0][1,0] = 1.0$ 
#   | 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  $a^{\text{obj}}_j$  coefficients:
#   |  $a^{\text{obj}}[1] = 1.0$ 
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

```

(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.

$$\begin{aligned} & \text{minimize} && \left\langle \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}, X_1 \right\rangle + x_1 + x_2 + 1 \\ & \text{subject to} && \left\langle \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}, X_1 \right\rangle - x_1 - x_2 \geq 0.0, \\ & && x_1 \begin{bmatrix} 0 & 1 \\ 1 & 3 \end{bmatrix} + x_2 \begin{bmatrix} 3 & 1 \\ 1 & 0 \end{bmatrix} - \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \succeq \mathbf{0}, \\ & && X_1 \succeq \mathbf{0}. \end{aligned} \tag{16.13}$$

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
L+ 1

# Two coordinates in F^{obj}_j coefficients:
```

(continues on next page)

```

#      | 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
DCCOORD
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
OBJACCOORD
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...
ACCOORD
4
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
BCCOORD
2
0 -250.0
1 4.0
```

(continues on next page)

```
# 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
OBJCOORD
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
OBJCOORD
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
    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**.

You can read and write *jtask* files using *MSK_readdata* and *MSK_writedata* specifying the extension *.jtask*.

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 *MSK_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": [
```

(continues on next page)

```

        "x1",
        "x2",
        "x3",
        "x4"
    ],
    "bk": [
        "lo",
        "ra",
        "lo",
        "lo"
    ],
    "b1": [
        0.0,
        0.0,
        0.0,
        0.0
    ],
    "bu": [
        1e+30,
        1e+1,
        1e+30,
        1e+30
    ],
    "type": [
        "cont",
        "cont",
        "cont",
        "cont"
    ]
    ],
    "con": {
        "name": [
            "c1",
            "c2",
            "c3"
        ],
        "bk": [
            "fx",
            "lo",
            "up"
        ],
        "b1": [
            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,
        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_REDUCATIONS":-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>						
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	DUAL UPPER	
?	<name>	?? <a value>	<a value>	<a value>	<a value>	<a value>	
VARIABLES							
INDEX	NAME	AT ACTIVITY	LOWER LIMIT	UPPER LIMIT	DUAL LOWER	DUAL UPPER	CONIC
↔DUAL							
?	<name>	?? <a value>	<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: 1o1.sol

In Listing 16.7 we show the solution file for the 1o1.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
			-2.49999999741653e+00		-0.
1	c2	SB	5.33333333049187e+01	1.50000000e+01	NONE
			-0.00000000000000e+00		2.
2	c3	UL	2.49999999842049e+01	NONE	2.50000000e+01
			-3.33333332895108e-01		-0.
VARIABLES					
INDEX	NAME	AT	ACTIVITY	LOWER LIMIT	UPPER LIMIT
			DUAL LOWER		DUAL UPPER
0	x1	LL	1.67020427038537e-09	0.00000000e+00	NONE
			-0.00000000000000e+00		-4.

(continues on next page)

(continued from previous page)

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 C:

Table 17.1: List of distributed examples

File	Description
<code>blas_lapack.c</code>	Demonstrates the MOSEK interface to BLAS/LAPACK linear algebra routines
<code>callback.c</code>	An example of data/progress callback
<code>ceo1.c</code>	A simple conic exponential problem
<code>concurrent.cc</code>	Implementation of a concurrent optimizer for linear and mixed-integer problems
<code>cqo1.c</code>	A simple conic quadratic problem
<code>errorreporting.c</code>	Demonstrates how error reporting can be customized
<code>feasrepair1.c</code>	A simple example of how to repair an infeasible problem
<code>gp1.c</code>	A simple geometric program (GP) in conic form
<code>lo1.c</code>	A simple linear problem
<code>lo2.c</code>	A simple linear problem
<code>logistic.c</code>	Implements logistic regression and simple log-sum-exp (CEO)
<code>mico1.c</code>	A simple mixed-integer conic problem
<code>milol.c</code>	A simple mixed-integer linear problem
<code>miointsol.c</code>	A simple mixed-integer linear problem with an initial guess
<code>modelLib.c</code>	Library of implementations of basic functions
<code>opt_server_async.c</code>	Uses MOSEK OptServer to solve an optimization problem asynchronously
<code>opt_server_sync.c</code>	Uses MOSEK OptServer to solve an optimization problem synchronously
<code>parallel.cc</code>	Demonstrates parallel optimization
<code>parameters.c</code>	Shows how to set optimizer parameters and read information items
<code>portfolio_1_basic.c</code>	Portfolio optimization - basic Markowitz model
<code>portfolio_2_frontier.c</code>	Portfolio optimization - efficient frontier
<code>portfolio_3_impact.c</code>	Portfolio optimization - market impact costs
<code>portfolio_4_transaction.c</code>	Portfolio optimization - transaction costs
<code>portfolio_5_cardinality.c</code>	Portfolio optimization - cardinality constraints
<code>pow1.c</code>	A simple power cone problem
<code>qcqo1.c</code>	A simple quadratically constrained quadratic problem
<code>qo1.c</code>	A simple quadratic problem

Continued on next page

Table 17.1 – continued from previous page

File	Description
<code>reoptimization.c</code>	Demonstrate how to modify and re-optimize a linear problem
<code>response.c</code>	Demonstrates proper response handling
<code>sdo1.c</code>	A simple semidefinite optimization problem
<code>sensitivity.c</code>	Sensitivity analysis performed on a small linear problem
<code>simple.c</code>	A simple I/O example: read problem from a file, solve and write solutions
<code>solutionquality.c</code>	Demonstrates how to examine the quality of a solution
<code>solvebasis.c</code>	Demonstrates solving a linear system with the basis matrix
<code>solvelinear.c</code>	Demonstrates solving a general linear system
<code>sparsecholesky.c</code>	Shows how to find a Cholesky factorization of a sparse matrix
<code>unicode.c</code>	Demonstrates string conversion to Unicode

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 C in version 9.0. 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 `MSKaccmodee` were removed. Use corresponding separate functions for manipulating variables and constraints. For example, instead of

```
MSK_putbound(task, MSK_ACC_VAR, ...);  
MSK_putbound(task, MSK_ACC_CON, ...);
```

use

```
MSK_putvarbound(task, ...);  
MSK_putconbound(task, ...);
```

and so on.

- Removed all Near problem and solution statuses i.e. `MSK_SOL_STA_NEAR_OPTIMAL`, `MSK_SOL_STA_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. 15.10](#).

18.2 Functions

Added

- *MSK_setupthreads*
- *MSK_appendsparsesymmatlist*
- *MSK_generateconenames*
- *MSK_generateconnames*
- *MSK_generatevarnames*
- *MSK_getacolslice*
- *MSK_getacolslicenummz*

- *MSK_getarowslice*
- *MSK_getarowslicenumz*
- *MSK_getatruncatetol*
- *MSK_getbarsslice*
- *MSK_getbarxslice*
- *MSK_getclist*
- *MSK_getskn*
- *MSK_putatruncatetol*
- *MSK_putbaraijlist*
- *MSK_putbararowlist*
- *MSK_putconboundlistconst*
- *MSK_putconboundsliceconst*
- *MSK_putconsolutioni*
- *MSK_putvarboundlistconst*
- *MSK_putvarboundsliceconst*
- *MSK_putvarsolutionj*
- *MSK_readjsonstring*
- *MSK_readlpstring*
- *MSK_readopfstring*
- *MSK_readptfstring*

Removed

- *MSK_checkconvexity*
- *MSK_chgbound*
- *MSK_getaslice*
- *MSK_getaslicenumz*
- *MSK_getbound*
- *MSK_getboundslice*
- *MSK_getnlfunc*
- *MSK_getsolutioni*
- *MSK_printdata*
- *MSK_putbound*
- *MSK_putboundlist*
- *MSK_putboundslice*
- *MSK_putnlfunc*
- *MSK_putsolutioni*

18.3 Parameters

Added

- *MSK_IPAR_INTPNT_ORDER_GP_NUM_SEEDS*
- *MSK_IPAR_INTPNT_PURIFY*
- *MSK_IPAR_LOG_INCLUDE_SUMMARY*
- *MSK_IPAR_LOG_LOCAL_INFO*
- *MSK_IPAR_MIO_CONIC_OUTER_APPROXIMATION*
- *MSK_IPAR_MIO_FEASPUMP_LEVEL*
- *MSK_IPAR_MIO_MAX_NUM_ROOT_CUT_ROUNDS*
- *MSK_IPAR_MIO_PROPAGATE_OBJECTIVE_CONSTRAINT*
- *MSK_IPAR_MIO_SEED*
- *MSK_IPAR_OPF_WRITE_LINE_LENGTH*
- *MSK_IPAR_PREOLVE_MAX_NUM_PASS*
- *MSK_IPAR_PTF_WRITE_TRANSFORM*
- *MSK_IPAR_SIM_SEED*
- *MSK_IPAR_WRITE_COMPRESSION*

Removed

- *MSK_DPAR_DATA_TOL_AIJ*
- *MSK_DPAR_INTPNT_NL_MERIT_BAL*
- *MSK_DPAR_INTPNT_NL_TOL_DFEAS*
- *MSK_DPAR_INTPNT_NL_TOL_MU_RED*
- *MSK_DPAR_INTPNT_NL_TOL_NEAR_REL*
- *MSK_DPAR_INTPNT_NL_TOL_PFEAS*
- *MSK_DPAR_INTPNT_NL_TOL_REL_GAP*
- *MSK_DPAR_INTPNT_NL_TOL_REL_STEP*
- *MSK_DPAR_MIO_DISABLE_TERM_TIME*
- *MSK_DPAR_MIO_NEAR_TOL_ABS_GAP*
- *MSK_DPAR_MIO_NEAR_TOL_REL_GAP*
- *MSK_IPAR_MIO_CONSTRUCT_SOL*
- *MSK_IPAR_MIO_MT_USER_CB*
- *MSK_IPAR_OPF_MAX_TERMS_PER_LINE*
- *MSK_IPAR_READ_DATA_COMPRESSED*
- *MSK_IPAR_READ_DATA_FORMAT*
- *MSK_IPAR_WRITE_DATA_COMPRESSED*
- *MSK_IPAR_WRITE_DATA_FORMAT*

18.4 Constants

Added

- *MSK_COMPRESS_ZSTD*
- *MSK_CT_DEXP*
- *MSK_CT_DPOW*
- *MSK_CT_PEXP*
- *MSK_CT_PPOW*
- *MSK_CT_ZERO*
- *MSK_DATA_FORMAT_PTF*
- *MSK_IINF_MIO_NUMBIN*
- *MSK_IINF_MIO_NUMBINCONEVAR*
- *MSK_IINF_MIO_NUMCONE*
- *MSK_IINF_MIO_NUMCONEVAR*
- *MSK_IINF_MIO_NUMCONT*
- *MSK_IINF_MIO_NUMCONTCONEVAR*
- *MSK_IINF_MIO_NUMDEXPCONES*
- *MSK_IINF_MIO_NUMDPOWCONES*
- *MSK_IINF_MIO_NUMINTCONEVAR*
- *MSK_IINF_MIO_NUMPEXPONES*
- *MSK_IINF_MIO_NUMPPOWCONES*
- *MSK_IINF_MIO_NUMQCONES*
- *MSK_IINF_MIO_NUMRQCONES*
- *MSK_IINF_MIO_PRE SOLVED_NUMBINCONEVAR*
- *MSK_IINF_MIO_PRE SOLVED_NUMCONE*
- *MSK_IINF_MIO_PRE SOLVED_NUMCONEVAR*
- *MSK_IINF_MIO_PRE SOLVED_NUMCONTCONEVAR*
- *MSK_IINF_MIO_PRE SOLVED_NUMDEXPCONES*
- *MSK_IINF_MIO_PRE SOLVED_NUMDPOWCONES*
- *MSK_IINF_MIO_PRE SOLVED_NUMINTCONEVAR*
- *MSK_IINF_MIO_PRE SOLVED_NUMPEXPONES*
- *MSK_IINF_MIO_PRE SOLVED_NUMPPOWCONES*
- *MSK_IINF_MIO_PRE SOLVED_NUMQCONES*
- *MSK_IINF_MIO_PRE SOLVED_NUMRQCONES*
- *MSK_IINF_PURIFY_DUAL_SUCCESS*
- *MSK_IINF_PURIFY_PRIMAL_SUCCESS*
- *MSK_LIINF_MIO_ANZ*

Removed

- MSK_DATAFORMAT_XML
- MSK_DINFITEM_MIO_HEURISTIC_TIME
- MSK_DINFITEM_MIO_OPTIMIZER_TIME
- MSK_IINFITEM_MIO_CONSTRUCT_NUM_ROUNDINGS
- MSK_IINFITEM_MIO_INITIAL_SOLUTION
- MSK_IINFITEM_MIO_NEAR_ABSGAP_SATISFIED
- MSK_IINFITEM_MIO_NEAR_RELGAP_SATISFIED
- MSK_LIINFITEM_MIO_SIM_MAXITER_SETBACKS
- MSK_MIONODESELTYPE_HYBRID
- MSK_MIONODESELTYPE_WORST
- MSK_PROBLEMTYPE_GECO
- MSK_PROSTA_NEAR_DUAL_FEAS
- MSK_PROSTA_NEAR_PRIM_AND_DUAL_FEAS
- MSK_PROSTA_NEAR_PRIM_FEAS
- MSK_SENSITIVITYTYPE_OPTIMAL_PARTITION
- MSK_SOLSTA_NEAR_DUAL_FEAS
- MSK_SOLSTA_NEAR_DUAL_INFEAS_CER
- MSK_SOLSTA_NEAR_INTEGER_OPTIMAL
- MSK_SOLSTA_NEAR_OPTIMAL
- MSK_SOLSTA_NEAR_PRIM_AND_DUAL_FEAS
- MSK_SOLSTA_NEAR_PRIM_FEAS
- MSK_SOLSTA_NEAR_PRIM_INFEAS_CER

18.5 Response Codes

Added

- *MSK_RES_ERR_APPENDING_TOO_BIG_CONE*
- *MSK_RES_ERR_CBF_DUPLICATE_POW_CONES*
- *MSK_RES_ERR_CBF_DUPLICATE_POW_STAR_CONES*
- *MSK_RES_ERR_CBF_INVALID_DIMENSION_OF_CONES*
- *MSK_RES_ERR_CBF_INVALID_EXP_DIMENSION*
- *MSK_RES_ERR_CBF_INVALID_NUMBER_OF_CONES*
- *MSK_RES_ERR_CBF_INVALID_POWER*
- *MSK_RES_ERR_CBF_INVALID_POWER_CONE_INDEX*

- *MSK_RES_ERR_CBF_INVALID_POWER_STAR_CONE_INDEX*
- *MSK_RES_ERR_CBF_POWER_CONE_IS_TOO_LONG*
- *MSK_RES_ERR_CBF_POWER_CONE_MISMATCH*
- *MSK_RES_ERR_CBF_POWER_STAR_CONE_MISMATCH*
- *MSK_RES_ERR_CBF_UNHANDLED_POWER_CONE_TYPE*
- *MSK_RES_ERR_CBF_UNHANDLED_POWER_STAR_CONE_TYPE*
- *MSK_RES_ERR_CONE_PARAMETER*
- *MSK_RES_ERR_FORMAT_STRING*
- *MSK_RES_ERR_INVALID_FILE_FORMAT_FOR_CFIX*
- *MSK_RES_ERR_INVALID_FILE_FORMAT_FOR_FREE_CONSTRAINTS*
- *MSK_RES_ERR_INVALID_FILE_FORMAT_FOR_NONLINEAR*
- *MSK_RES_ERR_INVALID_FILE_FORMAT_FOR_RANGED_CONSTRAINTS*
- *MSK_RES_ERR_NUM_ARGUMENTS*
- *MSK_RES_ERR_PTF_FORMAT*
- *MSK_RES_ERR_SHAPE_IS_TOO_LARGE*
- *MSK_RES_ERR_SLICE_SIZE*
- *MSK_RES_ERR_TOO_SMALL_A_TRUNCATION_VALUE*
- *MSK_RES_WRN_EXP_CONES_WITH_VARIABLES_FIXED_AT_ZERO*
- *MSK_RES_WRN_POW_CONES_WITH_ROOT_FIXED_AT_ZERO*

Removed

- *MSK_RES_ERR_CANNOT_CLONE_NL*
- *MSK_RES_ERR_CANNOT_HANDLE_NL*
- *MSK_RES_ERR_INVALID_ACCMODE*
- *MSK_RES_ERR_INVALID_FILE_FORMAT_FOR_GENERAL_NL*
- *MSK_RES_ERR_NONLINEAR_FUNCTIONS_NOT_ALLOWED*
- *MSK_RES_ERR_NR_ARGUMENTS*
- *MSK_RES_ERR_OPEN_DL*
- *MSK_RES_ERR_USER_FUNC_RET*
- *MSK_RES_ERR_USER_FUNC_RET_DATA*
- *MSK_RES_ERR_USER_NLO_EVAL*
- *MSK_RES_ERR_USER_NLO_EVAL_HESSUBI*
- *MSK_RES_ERR_USER_NLO_EVAL_HESSUBJ*
- *MSK_RES_ERR_USER_NLO_FUNC*
- *MSK_RES_TRM_MIO_NEAR_ABS_GAP*
- *MSK_RES_TRM_MIO_NEAR_REL_GAP*
- *MSK_RES_WRN_CONSTRUCT_INVALID_SOL_ITG*
- *MSK_RES_WRN_CONSTRUCT_NO_SOL_ITG*
- *MSK_RES_WRN_CONSTRUCT_SOLUTION_INFEAS*
- *MSK_RES_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.
- [AGMX96] 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.
- [Chv83] 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

Enumerations

MSKbasindtypee, 427
MSK_BI_RESERVED, 427
MSK_BI_NO_ERROR, 427
MSK_BI_NEVER, 427
MSK_BI_IF_FEASIBLE, 427
MSK_BI_ALWAYS, 427
MSKboundkeye, 427
MSK_BK_UP, 428
MSK_BK_RA, 428
MSK_BK_LO, 427
MSK_BK_FX, 428
MSK_BK_FR, 428
MSKbranchdire, 445
MSK_BRANCH_DIR_UP, 445
MSK_BRANCH_DIR_ROOT_LP, 445
MSK_BRANCH_DIR_PSEUDOCOST, 445
MSK_BRANCH_DIR_NEAR, 445
MSK_BRANCH_DIR_GUIDED, 445
MSK_BRANCH_DIR_FREE, 445
MSK_BRANCH_DIR_FAR, 445
MSK_BRANCH_DIR_DOWN, 445
MSKcallbackcodee, 429
MSK_CALLBACK_WRITE_OPF, 433
MSK_CALLBACK_UPDATE_PRIMAL_SIMPLEX_BI, 433
MSK_CALLBACK_UPDATE_PRIMAL_SIMPLEX, 433
MSK_CALLBACK_UPDATE_PRIMAL_BI, 433
MSK_CALLBACK_UPDATE_PRESOLVE, 433
MSK_CALLBACK_UPDATE_DUAL_SIMPLEX_BI, 433
MSK_CALLBACK_UPDATE_DUAL_SIMPLEX, 433
MSK_CALLBACK_UPDATE_DUAL_BI, 433
MSK_CALLBACK_SOLVING_REMOTE, 433
MSK_CALLBACK_READ_OPF_SECTION, 433
MSK_CALLBACK_READ_OPF, 433
MSK_CALLBACK_PRIMAL_SIMPLEX, 433
MSK_CALLBACK_NEW_INT_MIO, 433
MSK_CALLBACK_INTPNT, 433
MSK_CALLBACK_IM_SIMPLEX_BI, 433
MSK_CALLBACK_IM_SIMPLEX, 433
MSK_CALLBACK_IM_ROOT_CUTGEN, 433
MSK_CALLBACK_IM_READ, 433
MSK_CALLBACK_IM_QO_REFORMULATE, 433
MSK_CALLBACK_IM_PRIMAL_SIMPLEX, 433
MSK_CALLBACK_IM_PRIMAL_SENSIVITY, 432
MSK_CALLBACK_IM_PRIMAL_BI, 432
MSK_CALLBACK_IM_PRESOLVE, 432
MSK_CALLBACK_IM_ORDER, 432
MSK_CALLBACK_IM_MIO_PRIMAL_SIMPLEX, 432
MSK_CALLBACK_IM_MIO_INTPNT, 432
MSK_CALLBACK_IM_MIO_DUAL_SIMPLEX, 432
MSK_CALLBACK_IM_MIO, 432
MSK_CALLBACK_IM_LU, 432
MSK_CALLBACK_IM_LICENSE_WAIT, 432
MSK_CALLBACK_IM_INTPNT, 432
MSK_CALLBACK_IM_FULL_CONVEXITY_CHECK, 432
MSK_CALLBACK_IM_DUAL_SIMPLEX, 432
MSK_CALLBACK_IM_DUAL_SENSIVITY, 432
MSK_CALLBACK_IM_DUAL_BI, 432
MSK_CALLBACK_IM_CONIC, 432
MSK_CALLBACK_IM_BI, 432
MSK_CALLBACK_END_WRITE, 432
MSK_CALLBACK_END_TO_CONIC, 432
MSK_CALLBACK_END_SIMPLEX_BI, 432
MSK_CALLBACK_END_SIMPLEX, 431
MSK_CALLBACK_END_ROOT_CUTGEN, 431
MSK_CALLBACK_END_READ, 431
MSK_CALLBACK_END_QCQO_REFORMULATE, 431
MSK_CALLBACK_END_PRIMAL_SIMPLEX_BI, 431
MSK_CALLBACK_END_PRIMAL_SIMPLEX, 431
MSK_CALLBACK_END_PRIMAL_SETUP_BI, 431
MSK_CALLBACK_END_PRIMAL_SENSITIVITY, 431
MSK_CALLBACK_END_PRIMAL_REPAIR, 431
MSK_CALLBACK_END_PRIMAL_BI, 431
MSK_CALLBACK_END_PRESOLVE, 431
MSK_CALLBACK_END_OPTIMIZER, 431
MSK_CALLBACK_END_MIO, 431
MSK_CALLBACK_END_LICENSE_WAIT, 431
MSK_CALLBACK_END_INTPNT, 431
MSK_CALLBACK_END_INFEAS_ANA, 431
MSK_CALLBACK_END_FULL_CONVEXITY_CHECK, 431
MSK_CALLBACK_END_DUAL_SIMPLEX_BI, 431
MSK_CALLBACK_END_DUAL_SIMPLEX, 431
MSK_CALLBACK_END_DUAL_SETUP_BI, 431
MSK_CALLBACK_END_DUAL_SENSITIVITY, 431
MSK_CALLBACK_END_DUAL_BI, 431
MSK_CALLBACK_END_CONIC, 431
MSK_CALLBACK_END_BI, 430
MSK_CALLBACK_DUAL_SIMPLEX, 430
MSK_CALLBACK_CONIC, 430
MSK_CALLBACK_BEGIN_WRITE, 430
MSK_CALLBACK_BEGIN_TO_CONIC, 430
MSK_CALLBACK_BEGIN_SIMPLEX_BI, 430
MSK_CALLBACK_BEGIN_SIMPLEX, 430
MSK_CALLBACK_BEGIN_ROOT_CUTGEN, 430
MSK_CALLBACK_BEGIN_READ, 430
MSK_CALLBACK_BEGIN_QCQO_REFORMULATE, 430
MSK_CALLBACK_BEGIN_PRIMAL_SIMPLEX_BI, 430

MSK_CALLBACK_BEGIN_PRIMAL_SIMPLEX, 430
 MSK_CALLBACK_BEGIN_PRIMAL_SETUP_BI, 430
 MSK_CALLBACK_BEGIN_PRIMAL_SENSITIVITY, 430
 MSK_CALLBACK_BEGIN_PRIMAL_REPAIR, 430
 MSK_CALLBACK_BEGIN_PRIMAL_BI, 430
 MSK_CALLBACK_BEGIN_PRESOLVE, 430
 MSK_CALLBACK_BEGIN_OPTIMIZER, 430
 MSK_CALLBACK_BEGIN_MIO, 430
 MSK_CALLBACK_BEGIN_LICENSE_WAIT, 430
 MSK_CALLBACK_BEGIN_INTPNT, 430
 MSK_CALLBACK_BEGIN_INFEAS_ANA, 430
 MSK_CALLBACK_BEGIN_FULL_CONVEXITY_CHECK, 430
 MSK_CALLBACK_BEGIN_DUAL_SIMPLEX_BI, 429
 MSK_CALLBACK_BEGIN_DUAL_SIMPLEX, 429
 MSK_CALLBACK_BEGIN_DUAL_SETUP_BI, 429
 MSK_CALLBACK_BEGIN_DUAL_SENSITIVITY, 429
 MSK_CALLBACK_BEGIN_DUAL_BI, 429
 MSK_CALLBACK_BEGIN_CONIC, 429
 MSK_CALLBACK_BEGIN_BI, 429
 MSKcheckconvexitytypee, 433
 MSK_CHECK_CONVEXITY_SIMPLE, 434
 MSK_CHECK_CONVEXITY_NONE, 434
 MSK_CHECK_CONVEXITY_FULL, 434
 MSKcompressypee, 434
 MSK_COMPRESS_ZSTD, 434
 MSK_COMPRESS_NONE, 434
 MSK_COMPRESS_GZIP, 434
 MSK_COMPRESS_FREE, 434
 MSKconetypee, 434
 MSK_CT_ZERO, 434
 MSK_CT_RQUAD, 434
 MSK_CT_QUAD, 434
 MSK_CT_PPOW, 434
 MSK_CT_PEXP, 434
 MSK_CT_DPOW, 434
 MSK_CT_DEXP, 434
 MSKdataformat, 435
 MSK_DATA_FORMAT_TASK, 435
 MSK_DATA_FORMAT_PTF, 435
 MSK_DATA_FORMAT_OP, 435
 MSK_DATA_FORMAT_MPS, 435
 MSK_DATA_FORMAT_LP, 435
 MSK_DATA_FORMAT_JSON_TASK, 435
 MSK_DATA_FORMAT_FREE_MPS, 435
 MSK_DATA_FORMAT_EXTENSION, 435
 MSK_DATA_FORMAT_CB, 435
 MSKdinfiteme, 435
 MSK_DINF_TO_CONIC_TIME, 439
 MSK_DINF_SOL_ITR_PVIOLVAR, 439
 MSK_DINF_SOL_ITR_PVIOLCONES, 439
 MSK_DINF_SOL_ITR_PVIOLCON, 439
 MSK_DINF_SOL_ITR_PVIOLBARVAR, 439
 MSK_DINF_SOL_ITR_PRIMAL_OBJ, 439
 MSK_DINF_SOL_ITR_NRM_Y, 439
 MSK_DINF_SOL_ITR_NRM_XX, 439
 MSK_DINF_SOL_ITR_NRM_XC, 439
 MSK_DINF_SOL_ITR_NRM_SUX, 439
 MSK_DINF_SOL_ITR_NRM_SUC, 439
 MSK_DINF_SOL_ITR_NRM_SNX, 439
 MSK_DINF_SOL_ITR_NRM_SLX, 439
 MSK_DINF_SOL_ITR_NRM_SLC, 439
 MSK_DINF_SOL_ITR_NRM_BARX, 439
 MSK_DINF_SOL_ITR_NRM_BARS, 439
 MSK_DINF_SOL_ITR_DVIOLVAR, 439
 MSK_DINF_SOL_ITR_DVIOLCONES, 439
 MSK_DINF_SOL_ITR_DVIOLCON, 439
 MSK_DINF_SOL_ITR_DVIOLBARVAR, 439
 MSK_DINF_SOL_ITR_DUAL_OBJ, 438
 MSK_DINF_SOL_ITG_PVIOLVAR, 438
 MSK_DINF_SOL_ITG_PVIOLITG, 438
 MSK_DINF_SOL_ITG_PVIOLCONES, 438
 MSK_DINF_SOL_ITG_PVIOLCON, 438
 MSK_DINF_SOL_ITG_PVIOLBARVAR, 438
 MSK_DINF_SOL_ITG_PRIMAL_OBJ, 438
 MSK_DINF_SOL_ITG_NRM_XX, 438
 MSK_DINF_SOL_ITG_NRM_XC, 438
 MSK_DINF_SOL_ITG_NRM_BARX, 438
 MSK_DINF_SOL_BAS_PVIOLVAR, 438
 MSK_DINF_SOL_BAS_PVIOLCON, 438
 MSK_DINF_SOL_BAS_PRIMAL_OBJ, 438
 MSK_DINF_SOL_BAS_NRM_Y, 438
 MSK_DINF_SOL_BAS_NRM_XX, 438
 MSK_DINF_SOL_BAS_NRM_XC, 438
 MSK_DINF_SOL_BAS_NRM_SUX, 438
 MSK_DINF_SOL_BAS_NRM_SUC, 438
 MSK_DINF_SOL_BAS_NRM_SLX, 438
 MSK_DINF_SOL_BAS_NRM_SLC, 438
 MSK_DINF_SOL_BAS_NRM_BARX, 437
 MSK_DINF_SOL_BAS_DVIOLVAR, 437
 MSK_DINF_SOL_BAS_DVIOLCON, 437
 MSK_DINF_SOL_BAS_DUAL_OBJ, 437
 MSK_DINF_SIM_TIME, 437
 MSK_DINF_SIM_PRIMAL_TIME, 437
 MSK_DINF_SIM_OBJ, 437
 MSK_DINF_SIM_FEAS, 437
 MSK_DINF_SIM_DUAL_TIME, 437
 MSK_DINF_RD_TIME, 437
 MSK_DINF_QCQO_REFORMULATE_WORST_CHOLESKY_DIAG_SCALING, 437
 MSK_DINF_QCQO_REFORMULATE_WORST_CHOLESKY_COLUMN_SCALING, 437
 MSK_DINF_QCQO_REFORMULATE_TIME, 437
 MSK_DINF_QCQO_REFORMULATE_MAX_PERTURBATION, 437
 MSK_DINF_PRIMAL_REPAIR_PENALTY_OBJ, 437
 MSK_DINF_PRESOLVE_TIME, 437
 MSK_DINF_PRESOLVE_LINDEP_TIME, 437
 MSK_DINF_PRESOLVE_ELI_TIME, 437
 MSK_DINF_OPTIMIZER_TIME, 437
 MSK_DINF_MIO_USER_OBJ_CUT, 437
 MSK_DINF_MIO_TIME, 437
 MSK_DINF_MIO_ROOT_PRESOLVE_TIME, 437
 MSK_DINF_MIO_ROOT_OPTIMIZER_TIME, 437
 MSK_DINF_MIO_ROOT_CUTGEN_TIME, 437
 MSK_DINF_MIO_PROBING_TIME, 436

MSK_DINF_MIO_OBJ_REL_GAP, 436
 MSK_DINF_MIO_OBJ_INT, 436
 MSK_DINF_MIO_OBJ_BOUND, 436
 MSK_DINF_MIO_OBJ_ABS_GAP, 436
 MSK_DINF_MIO_KNAPSACK_COVER_SEPARATION_TIME, 436
 MSK_DINF_MIO_IMPLIED_BOUND_TIME, 436
 MSK_DINF_MIO_GMI_SEPARATION_TIME, 436
 MSK_DINF_MIO_DUAL_BOUND_AFTER_PREOLVE, 436
 MSK_DINF_MIO_CONSTRUCT_SOLUTION_OBJ, 436
 MSK_DINF_MIO_CMIR_SEPARATION_TIME, 436
 MSK_DINF_MIO_CLIQUE_SEPARATION_TIME, 436
 MSK_DINF_INTPNT_TIME, 436
 MSK_DINF_INTPNT_PRIMAL_OBJ, 436
 MSK_DINF_INTPNT_PRIMAL_FEAS, 436
 MSK_DINF_INTPNT_ORDER_TIME, 436
 MSK_DINF_INTPNT_OPT_STATUS, 435
 MSK_DINF_INTPNT_FACTOR_NUM_FLOPS, 435
 MSK_DINF_INTPNT_DUAL_OBJ, 435
 MSK_DINF_INTPNT_DUAL_FEAS, 435
 MSK_DINF_BI_TIME, 435
 MSK_DINF_BI_PRIMAL_TIME, 435
 MSK_DINF_BI_DUAL_TIME, 435
 MSK_DINF_BI_CLEAN_TIME, 435
 MSK_DINF_BI_CLEAN_PRIMAL_TIME, 435
 MSK_DINF_BI_CLEAN_DUAL_TIME, 435
 MSKdparame, 369
 MSKfeaturee, 439
 MSK_FEATURE_PTS, 439
 MSK_FEATURE_PTON, 440
 MSKiinfiteme, 440
 MSK_IINF_STO_NUM_A_REALLOC, 444
 MSK_IINF_SOL_ITR_SOLSTA, 444
 MSK_IINF_SOL_ITR_PROSTA, 444
 MSK_IINF_SOL_ITG_SOLSTA, 444
 MSK_IINF_SOL_ITG_PROSTA, 444
 MSK_IINF_SOL_BAS_SOLSTA, 444
 MSK_IINF_SOL_BAS_PROSTA, 444
 MSK_IINF_SIM_SOLVE_DUAL, 444
 MSK_IINF_SIM_PRIMAL_ITER, 444
 MSK_IINF_SIM_PRIMAL_INF_ITER, 444
 MSK_IINF_SIM_PRIMAL_HOTSTART_LU, 444
 MSK_IINF_SIM_PRIMAL_HOTSTART, 444
 MSK_IINF_SIM_PRIMAL_DEG_ITER, 444
 MSK_IINF_SIM_NUMVAR, 444
 MSK_IINF_SIM_NUMCON, 444
 MSK_IINF_SIM_DUAL_ITER, 444
 MSK_IINF_SIM_DUAL_INF_ITER, 444
 MSK_IINF_SIM_DUAL_HOTSTART_LU, 444
 MSK_IINF_SIM_DUAL_HOTSTART, 444
 MSK_IINF_SIM_DUAL_DEG_ITER, 444
 MSK_IINF_RD_PROTYPE, 444
 MSK_IINF_RD_NUMVAR, 444
 MSK_IINF_RD_NUMQ, 444
 MSK_IINF_RD_NUMINTVAR, 444
 MSK_IINF_RD_NUMCONE, 443
 MSK_IINF_RD_NUMCON, 443
 MSK_IINF_RD_NUMBARVAR, 443
 MSK_IINF_PURIFY_PRIMAL_SUCCESS, 443
 MSK_IINF_PURIFY_DUAL_SUCCESS, 443
 MSK_IINF_OPTIMIZE_RESPONSE, 443
 MSK_IINF_OPT_NUMVAR, 443
 MSK_IINF_OPT_NUMCON, 443
 MSK_IINF_MIO_USER_OBJ_CUT, 443
 MSK_IINF_MIO_TOTAL_NUM_CUTS, 443
 MSK_IINF_MIO_RELGAP_SATISFIED, 443
 MSK_IINF_MIO_PREOLVED_NUMVAR, 443
 MSK_IINF_MIO_PREOLVED_NUMRQCONES, 443
 MSK_IINF_MIO_PREOLVED_NUMQCONES, 443
 MSK_IINF_MIO_PREOLVED_NUMPPOWCONES, 443
 MSK_IINF_MIO_PREOLVED_NUMPEXPCONES, 443
 MSK_IINF_MIO_PREOLVED_NUMINTCONEVAR, 443
 MSK_IINF_MIO_PREOLVED_NUMINT, 443
 MSK_IINF_MIO_PREOLVED_NUMDPOWCONES, 443
 MSK_IINF_MIO_PREOLVED_NUMDEXPCONES, 443
 MSK_IINF_MIO_PREOLVED_NUMCONTCONEVAR, 443
 MSK_IINF_MIO_PREOLVED_NUMCONT, 443
 MSK_IINF_MIO_PREOLVED_NUMCONEVAR, 443
 MSK_IINF_MIO_PREOLVED_NUMCONE, 443
 MSK_IINF_MIO_PREOLVED_NUMCON, 442
 MSK_IINF_MIO_PREOLVED_NUMBINCONEVAR, 442
 MSK_IINF_MIO_PREOLVED_NUMBIN, 442
 MSK_IINF_MIO_OBJ_BOUND_DEFINED, 442
 MSK_IINF_MIO_NUMVAR, 442
 MSK_IINF_MIO_NUMRQCONES, 442
 MSK_IINF_MIO_NUMQCONES, 442
 MSK_IINF_MIO_NUMPPOWCONES, 442
 MSK_IINF_MIO_NUMPEXPCONES, 442
 MSK_IINF_MIO_NUMINTCONEVAR, 442
 MSK_IINF_MIO_NUMINT, 442
 MSK_IINF_MIO_NUMDPOWCONES, 442
 MSK_IINF_MIO_NUMDEXPCONES, 442
 MSK_IINF_MIO_NUMCONTCONEVAR, 442
 MSK_IINF_MIO_NUMCONT, 442
 MSK_IINF_MIO_NUMCONEVAR, 442
 MSK_IINF_MIO_NUMCONE, 442
 MSK_IINF_MIO_NUMCON, 442
 MSK_IINF_MIO_NUMBINCONEVAR, 442
 MSK_IINF_MIO_NUMBIN, 442
 MSK_IINF_MIO_NUM_REPEATED_PREOLVE, 442
 MSK_IINF_MIO_NUM_RELAX, 442
 MSK_IINF_MIO_NUM_KNAPSACK_COVER_CUTS, 442
 MSK_IINF_MIO_NUM_INT_SOLUTIONS, 442
 MSK_IINF_MIO_NUM_IMPLIED_BOUND_CUTS, 441
 MSK_IINF_MIO_NUM_GOMORY_CUTS, 441
 MSK_IINF_MIO_NUM_CMIR_CUTS, 441
 MSK_IINF_MIO_NUM_CLIQUE_CUTS, 441
 MSK_IINF_MIO_NUM_BRANCH, 441
 MSK_IINF_MIO_NUM_ACTIVE_NODES, 441
 MSK_IINF_MIO_NODE_DEPTH, 441
 MSK_IINF_MIO_CONSTRUCT_SOLUTION, 441
 MSK_IINF_MIO_CLIQUE_TABLE_SIZE, 441
 MSK_IINF_MIO_ABSGAP_SATISFIED, 441
 MSK_IINF_INTPNT_SOLVE_DUAL, 441
 MSK_IINF_INTPNT_NUM_THREADS, 441
 MSK_IINF_INTPNT_ITER, 441

MSK_IINF_INTPNT_FACTOR_DIM_DENSE, 441
 MSK_IINF_ANA_PRO_NUM_VAR_UP, 441
 MSK_IINF_ANA_PRO_NUM_VAR_RA, 441
 MSK_IINF_ANA_PRO_NUM_VAR_LO, 441
 MSK_IINF_ANA_PRO_NUM_VAR_INT, 441
 MSK_IINF_ANA_PRO_NUM_VAR_FR, 441
 MSK_IINF_ANA_PRO_NUM_VAR_EQ, 441
 MSK_IINF_ANA_PRO_NUM_VAR_CONT, 441
 MSK_IINF_ANA_PRO_NUM_VAR_BIN, 441
 MSK_IINF_ANA_PRO_NUM_VAR, 440
 MSK_IINF_ANA_PRO_NUM_CON_UP, 440
 MSK_IINF_ANA_PRO_NUM_CON_RA, 440
 MSK_IINF_ANA_PRO_NUM_CON_LO, 440
 MSK_IINF_ANA_PRO_NUM_CON_FR, 440
 MSK_IINF_ANA_PRO_NUM_CON_EQ, 440
 MSK_IINF_ANA_PRO_NUM_CON, 440
 MSKinfypee, 445
 MSK_INF_LINT_TYPE, 445
 MSK_INF_INT_TYPE, 445
 MSK_INF_DOU_TYPE, 445
 MSKintpnthotstarte, 429
 MSK_INTPNT_HOTSTART_PRIMAL_DUAL, 429
 MSK_INTPNT_HOTSTART_PRIMAL, 429
 MSK_INTPNT_HOTSTART_NONE, 429
 MSK_INTPNT_HOTSTART_DUAL, 429
 MSKiomodee, 445
 MSK_IOMODE_WRITE, 445
 MSK_IOMODE_READWRITE, 445
 MSK_IOMODE_READ, 445
 MSKiparame, 378
 MSKliinfiteme, 440
 MSK_LIINF_RD_NUMQNZ, 440
 MSK_LIINF_RD_NUMANZ, 440
 MSK_LIINF_MIO_SIMPLEX_ITER, 440
 MSK_LIINF_MIO_PRE SOLVED_ANZ, 440
 MSK_LIINF_MIO_INTPNT_ITER, 440
 MSK_LIINF_MIO_ANZ, 440
 MSK_LIINF_INTPNT_FACTOR_NUM_NZ, 440
 MSK_LIINF_BI_PRIMAL_ITER, 440
 MSK_LIINF_BI_DUAL_ITER, 440
 MSK_LIINF_BI_CLEAN_PRIMAL_ITER, 440
 MSK_LIINF_BI_CLEAN_PRIMAL_DEG_ITER, 440
 MSK_LIINF_BI_CLEAN_DUAL_ITER, 440
 MSK_LIINF_BI_CLEAN_DUAL_DEG_ITER, 440
 MSKmarke, 428
 MSK_MARK_UP, 428
 MSK_MARK_LO, 428
 MSKmiocontsoltypee, 445
 MSK_MIO_CONT_SOL_ROOT, 445
 MSK_MIO_CONT_SOL_NONE, 445
 MSK_MIO_CONT_SOL_ITG_REL, 445
 MSK_MIO_CONT_SOL_ITG, 445
 MSKmiomodee, 446
 MSK_MIO_MODE_SATISFIED, 446
 MSK_MIO_MODE_IGNORED, 446
 MSKmionodeseltypee, 446
 MSK_MIO_NODE_SELECTION_PSEUDO, 446
 MSK_MIO_NODE_SELECTION_FREE, 446
 MSK_MIO_NODE_SELECTION_FIRST, 446
 MSK_MIO_NODE_SELECTION_BEST, 446
 MSKmpsformate, 446
 MSK_MPS_FORMAT_STRICT, 446
 MSK_MPS_FORMAT_RELAXED, 446
 MSK_MPS_FORMAT_FREE, 446
 MSK_MPS_FORMAT_CPLEX, 446
 MSKnametypee, 434
 MSK_NAME_TYPE_MPS, 434
 MSK_NAME_TYPE_LP, 434
 MSK_NAME_TYPE_GEN, 434
 MSKobjsensee, 446
 MSK_OBJECTIVE_SENSE_MINIMIZE, 446
 MSK_OBJECTIVE_SENSE_MAXIMIZE, 446
 MSKonoffkeye, 446
 MSK_ON, 446
 MSK_OFF, 446
 MSKoptimizertypee, 446
 MSK_OPTIMIZER_PRIMAL_SIMPLEX, 447
 MSK_OPTIMIZER_MIXED_INT, 447
 MSK_OPTIMIZER_INTPNT, 446
 MSK_OPTIMIZER_FREE_SIMPLEX, 446
 MSK_OPTIMIZER_FREE, 446
 MSK_OPTIMIZER_DUAL_SIMPLEX, 446
 MSK_OPTIMIZER_CONIC, 446
 MSKorderingtypee, 447
 MSK_ORDER_METHOD_TRY_GRAPHPAR, 447
 MSK_ORDER_METHOD_NONE, 447
 MSK_ORDER_METHOD_FREE, 447
 MSK_ORDER_METHOD_FORCE_GRAPHPAR, 447
 MSK_ORDER_METHOD_EXPERIMENTAL, 447
 MSK_ORDER_METHOD_APPMINLOC, 447
 MSKparametertypee, 447
 MSK_PAR_STR_TYPE, 447
 MSK_PAR_INVALID_TYPE, 447
 MSK_PAR_INT_TYPE, 447
 MSK_PAR_DOU_TYPE, 447
 MSKpresolvemodee, 447
 MSK_PRE SOLVE_MODE_ON, 447
 MSK_PRE SOLVE_MODE_OFF, 447
 MSK_PRE SOLVE_MODE_FREE, 447
 MSKproblemiteme, 447
 MSK_PI_VAR, 447
 MSK_PI_CONE, 447
 MSK_PI_CON, 447
 MSKproblemtypee, 447
 MSK_PROBTYPE_QO, 447
 MSK_PROBTYPE_QCQO, 448
 MSK_PROBTYPE_MIXED, 448
 MSK_PROBTYPE_LO, 447
 MSK_PROBTYPE_CONIC, 448
 MSKprostae, 448
 MSK_PRO_STA_UNKNOWN, 448
 MSK_PRO_STA_PRIM_INFEAS_OR_UNBOUNDED, 448
 MSK_PRO_STA_PRIM_INFEAS, 448
 MSK_PRO_STA_PRIM_FEAS, 448
 MSK_PRO_STA_PRIM_AND_DUAL_INFEAS, 448
 MSK_PRO_STA_PRIM_AND_DUAL_FEAS, 448

MSK_PRO_STA_ILL_POSED, 448
 MSK_PRO_STA_DUAL_INFEAS, 448
 MSK_PRO_STA_DUAL_FEAS, 448
 MSKpurifye, 429
 MSK_PURIFY_PRIMAL_DUAL, 429
 MSK_PURIFY_PRIMAL, 429
 MSK_PURIFY_NONE, 429
 MSK_PURIFY_DUAL, 429
 MSK_PURIFY_AUTO, 429
 MSKrescodee, 409
 MSKrescodetypee, 448
 MSK_RESPONSE_WRN, 448
 MSK_RESPONSE_UNK, 448
 MSK_RESPONSE_TRM, 448
 MSK_RESPONSE_OK, 448
 MSK_RESPONSE_ERR, 448
 MSKscalingmethode, 449
 MSK_SCALING_METHOD_POW2, 449
 MSK_SCALING_METHOD_FREE, 449
 MSKscalingtypee, 448
 MSK_SCALING_NONE, 449
 MSK_SCALING_MODERATE, 449
 MSK_SCALING_FREE, 448
 MSK_SCALING_AGGRESSIVE, 449
 MSKscopre, 434
 MSK_OPR_SQRT, 434
 MSK_OPR_POW, 434
 MSK_OPR_LOG, 434
 MSK_OPR_EXP, 434
 MSK_OPR_ENT, 434
 MSKsensitivitytypee, 449
 MSK_SENSITIVITY_TYPE_BASIS, 449
 MSKsimdegene, 428
 MSK_SIM_DEGEN_NONE, 428
 MSK_SIM_DEGEN_MODERATE, 428
 MSK_SIM_DEGEN_MINIMUM, 428
 MSK_SIM_DEGEN_FREE, 428
 MSK_SIM_DEGEN_AGGRESSIVE, 428
 MSKsimdupvece, 428
 MSK_SIM_EXPLOIT_DUPVEC_ON, 428
 MSK_SIM_EXPLOIT_DUPVEC_OFF, 429
 MSK_SIM_EXPLOIT_DUPVEC_FREE, 429
 MSKsimhotstarte, 429
 MSK_SIM_HOTSTART_STATUS_KEYS, 429
 MSK_SIM_HOTSTART_NONE, 429
 MSK_SIM_HOTSTART_FREE, 429
 MSKsimreforme, 428
 MSK_SIM_REFORMULATION_ON, 428
 MSK_SIM_REFORMULATION_OFF, 428
 MSK_SIM_REFORMULATION_FREE, 428
 MSK_SIM_REFORMULATION_AGGRESSIVE, 428
 MSKsimselftypee, 449
 MSK_SIM_SELECTION_SE, 449
 MSK_SIM_SELECTION_PARTIAL, 449
 MSK_SIM_SELECTION_FULL, 449
 MSK_SIM_SELECTION_FREE, 449
 MSK_SIM_SELECTION_DEVEX, 449
 MSK_SIM_SELECTION_ASE, 449
 MSKsoliteme, 449
 MSK_SOL_ITEM_Y, 449
 MSK_SOL_ITEM_XX, 449
 MSK_SOL_ITEM_XC, 449
 MSK_SOL_ITEM_SUX, 449
 MSK_SOL_ITEM_SUC, 449
 MSK_SOL_ITEM_SNX, 449
 MSK_SOL_ITEM_SLX, 449
 MSK_SOL_ITEM_SLC, 449
 MSKsolstae, 450
 MSK_SOL_STA_UNKNOWN, 450
 MSK_SOL_STA_PRIM_INFEAS_CER, 450
 MSK_SOL_STA_PRIM_ILLPOSED_CER, 450
 MSK_SOL_STA_PRIM_FEAS, 450
 MSK_SOL_STA_PRIM_AND_DUAL_FEAS, 450
 MSK_SOL_STA_OPTIMAL, 450
 MSK_SOL_STA_INTEGER_OPTIMAL, 450
 MSK_SOL_STA_DUAL_INFEAS_CER, 450
 MSK_SOL_STA_DUAL_ILLPOSED_CER, 450
 MSK_SOL_STA_DUAL_FEAS, 450
 MSKsoltypee, 450
 MSK_SOL_ITR, 450
 MSK_SOL_ITG, 450
 MSK_SOL_BAS, 450
 MSKsolveforme, 450
 MSK_SOLVE_PRIMAL, 450
 MSK_SOLVE_FREE, 450
 MSK_SOLVE_DUAL, 450
 MSKsparame, 406
 MSKstakeye, 450
 MSK_SK_UPR, 450
 MSK_SK_UNK, 450
 MSK_SK_SUPBAS, 450
 MSK_SK_LOW, 450
 MSK_SK_INF, 451
 MSK_SK_FIX, 451
 MSK_SK_BAS, 450
 MSKstartpointtypee, 451
 MSK_STARTING_POINT_SATISFY_BOUNDS, 451
 MSK_STARTING_POINT_GUESS, 451
 MSK_STARTING_POINT_FREE, 451
 MSK_STARTING_POINT_CONSTANT, 451
 MSKstreamtypee, 451
 MSK_STREAM_WRN, 451
 MSK_STREAM_MSG, 451
 MSK_STREAM_LOG, 451
 MSK_STREAM_ERR, 451
 MSKsymmattypee, 435
 MSK_SYMMAT_TYPE_SPARSE, 435
 MSKtransposee, 428
 MSK_TRANSPOSE_YES, 428
 MSK_TRANSPOSE_NO, 428
 MSKuploe, 428
 MSK_UPLO_UP, 428
 MSK_UPLO_LO, 428
 MSKvaluee, 451
 MSK_MAX_STR_LEN, 451
 MSK_LICENSE_BUFFER_LENGTH, 451

MSKvariabletypee, 451
 MSK_VAR_TYPE_INT, 451
 MSK_VAR_TYPE_CONT, 451
 MSKxmlwriteroutputtypee, 448
 MSK_WRITE_XML_MODE_ROW, 448
 MSK_WRITE_XML_MODE_COL, 448

Functions

MSK_analyzenames, 213
 MSK_analyzeproblem, 213
 MSK_analyzesolution, 213
 MSK_appendbarvars, 214
 MSK_appendcone, 214
 MSK_appendconeseq, 215
 MSK_appendconesseq, 216
 MSK_appendcons, 216
 MSK_appendsparsesymmat, 216
 MSK_appendsparsesymmatlist, 217
 MSK_appendvars, 218
 MSK_asyncgetresult, 218
 MSK_asyncoptimize, 219
 MSK_asyncpoll, 219
 MSK_asyncstop, 219
 MSK_axpy, 220
 MSK_basiscond, 220
 MSK_bktostr, 221
 MSK_callbackcodetostr, 221
 MSK_calloclbgen, 221
 MSK_calloclbgtask, 221
 MSK_calloclenv, 222
 MSK_calloctask, 222
 MSK_checkinall, 222
 MSK_checkinlicense, 223
 MSK_checkmemenv, 223
 MSK_checkmentask, 223
 MSK_checkoutlicense, 224
 MSK_checkversion, 224
 MSK_chgconbound, 224
 MSK_chgvarbound, 225
 MSK_clonetask, 225
 MSK_commitchanges, 226
 MSK_computesparsescholesky, 226
 MSK_conetypetostr, 227
 MSK_deleteenv, 227
 MSK_deletesolution, 228
 MSK_deletetask, 228
 MSK_dot, 228
 MSK_dualsensitivity, 229
 MSK_echoenv, 229
 MSK_echointro, 229
 MSK_echotask, 230
 MSK_freedbgen, 230
 MSK_freedbgtask, 230
 MSK_freeenv, 231
 MSK_freetask, 231
 MSK_gemm, 231
 MSK_genv, 232
 MSK_generateconenames, 233

MSK_generateconenames, 233
 MSK_generatevarnames, 234
 MSK_getacol, 234
 MSK_getacolnumnz, 234
 MSK_getacolslice, 235
 MSK_getacolslice64, 235
 MSK_getacolslicenumnz, 236
 MSK_getacolslicenumnz64, 236
 MSK_getacolslicetrip, 237
 MSK_getaij, 237
 MSK_getapiecennumnz, 237
 MSK_getarow, 238
 MSK_getarownumnz, 238
 MSK_getarowslice, 239
 MSK_getarowslice64, 239
 MSK_getarowslicenumnz, 240
 MSK_getarowslicenumnz64, 240
 MSK_getarowslicetrip, 240
 MSK_getatruncatetol, 241
 MSK_getbarablocktriplet, 241
 MSK_getbaraidx, 242
 MSK_getbaraidxij, 242
 MSK_getbaraidxinfo, 243
 MSK_getbarasparsity, 243
 MSK_getbarcbblocktriplet, 244
 MSK_getbarcidx, 244
 MSK_getbarcidxinfo, 244
 MSK_getbarcidxj, 245
 MSK_getbarcsparsity, 245
 MSK_getbarsj, 245
 MSK_getbarsslice, 246
 MSK_getbarvarname, 246
 MSK_getbarvarnameindex, 247
 MSK_getbarvarnamelen, 247
 MSK_getbarxj, 247
 MSK_getbarxslice, 248
 MSK_getbuildinfo, 248
 MSK_getc, 248
 MSK_getcallbackfunc, 249
 MSK_getcfix, 249
 MSK_getcj, 249
 MSK_getclist, 249
 MSK_getcodedesc, 250
 MSK_getconbound, 250
 MSK_getconboundslice, 250
 MSK_getcone, 251
 MSK_getconeinfo, 251
 MSK_getconename, 252
 MSK_getconenameindex, 252
 MSK_getconenamelen, 252
 MSK_getconname, 253
 MSK_getconnameindex, 253
 MSK_getconnamelen, 253
 MSK_getcslice, 254
 MSK_getdimbarvarj, 254
 MSK_getdouinf, 254
 MSK_getdoupam, 255
 MSK_getdualobj, 255

MSK_getdualsolutionnorms, 255
 MSK_getdviolbarvar, 256
 MSK_getdviolcon, 256
 MSK_getdviolcones, 257
 MSK_getdviolvar, 257
 MSK_getenv, 258
 MSK_getinfeasiblesubproblem, 258
 MSK_getinfindex, 258
 MSK_getinfmax, 259
 MSK_getinfname, 259
 MSK_getintinf, 259
 MSK_getintparam, 260
 MSK_getlasterror, 260
 MSK_getlasterror64, 260
 MSK_getlenbarvarj, 261
 MSK_getlintinf, 261
 MSK_getmaxnamelen, 261
 MSK_getmaxnumanz, 262
 MSK_getmaxnumanz64, 262
 MSK_getmaxnumbarvar, 262
 MSK_getmaxnumcon, 263
 MSK_getmaxnumcone, 263
 MSK_getmaxnumqnz, 263
 MSK_getmaxnumqnz64, 263
 MSK_getmaxnumvar, 264
 MSK_getmemusagetask, 264
 MSK_getnadouinf, 264
 MSK_getnadouparam, 265
 MSK_getnaintinf, 265
 MSK_getnaintparam, 265
 MSK_getnastrparam, 265
 MSK_getnastrparamal, 266
 MSK_getnumanz, 266
 MSK_getnumanz64, 266
 MSK_getnumbarablocktriplets, 267
 MSK_getnumbaranz, 267
 MSK_getnumbarcblocktriplets, 267
 MSK_getnumbarcnz, 268
 MSK_getnumbarvar, 268
 MSK_getnumcon, 268
 MSK_getnumcone, 268
 MSK_getnumconemem, 269
 MSK_getnumintvar, 269
 MSK_getnumparam, 269
 MSK_getnumqconknz, 269
 MSK_getnumqconknz64, 270
 MSK_getnumqobjnz, 270
 MSK_getnumqobjnz64, 270
 MSK_getnumsymmat, 270
 MSK_getnumvar, 271
 MSK_getobjname, 271
 MSK_getobjnamelen, 271
 MSK_getobjsense, 272
 MSK_getparammax, 272
 MSK_getparamname, 272
 MSK_getprimalobj, 272
 MSK_getprimalsolutionnorms, 273
 MSK_getprobtype, 273
 MSK_getprosta, 273
 MSK_getpviolbarvar, 274
 MSK_getpviolcon, 274
 MSK_getpviolcones, 275
 MSK_getpviolvar, 275
 MSK_getqconk, 276
 MSK_getqconk64, 276
 MSK_getqobj, 277
 MSK_getqobj64, 277
 MSK_getqobjij, 278
 MSK_getreducedcosts, 278
 MSK_getresponseclass, 279
 MSK_getskc, 279
 MSK_getskcslice, 279
 MSK_getskn, 279
 MSK_getskx, 280
 MSK_getskxslice, 280
 MSK_getslc, 280
 MSK_getslcslice, 281
 MSK_getslx, 281
 MSK_getslxslice, 281
 MSK_getsnx, 282
 MSK_getsnxslice, 282
 MSK_getsolsta, 282
 MSK_getsolution, 283
 MSK_getsolutioninfo, 284
 MSK_getsolutionslice, 285
 MSK_getsparsesymmat, 286
 MSK_getstrparam, 286
 MSK_getstrparamal, 286
 MSK_getstrparamlen, 287
 MSK_getsuc, 287
 MSK_getsucslice, 287
 MSK_getsux, 288
 MSK_getsuxslice, 288
 MSK_getsymbcon, 288
 MSK_getsymbcondim, 289
 MSK_getsymmatinfo, 289
 MSK_gettaskname, 290
 MSK_gettasknamelen, 290
 MSK_getvarbound, 290
 MSK_getvarboundslice, 291
 MSK_getvarname, 291
 MSK_getvarnameindex, 291
 MSK_getvarnamelen, 292
 MSK_getvartype, 292
 MSK_getvartypelist, 292
 MSK_getversion, 293
 MSK_getxc, 293
 MSK_getxcslice, 293
 MSK_getxx, 294
 MSK_getxxslice, 294
 MSK_gety, 294
 MSK_getyslice, 294
 MSK_initbasissolve, 295
 MSK_inputdata, 295
 MSK_inputdata64, 296
 MSK_iparvaltosymnam, 297

MSK_isdoupname, 297
 MSK_isinfinity, 298
 MSK_isintparname, 298
 MSK_isstrparname, 298
 MSK_licensecleanup, 298
 MSK_linkfiletoenvstream, 299
 MSK_linkfiletotaskstream, 299
 MSK_linkfunctoenvstream, 299
 MSK_linkfunctotaskstream, 300
 MSK_makeemptytask, 300
 MSK_makeenv, 300
 MSK_makeenvalloc, 301
 MSK_maketask, 301
 MSK_onesolutionsummary, 301
 MSK_optimize, 302
 MSK_optimizermt, 302
 MSK_optimizersummary, 303
 MSK_optimizetrm, 303
 MSK_potrf, 303
 MSK_primalrepair, 303
 MSK_primalsensitivity, 304
 MSK_printparam, 305
 MSK_probttypetostr, 306
 MSK_prostattostr, 306
 MSK_putacol, 306
 MSK_putacollist, 307
 MSK_putacollist64, 307
 MSK_putacolslice, 308
 MSK_putacolslice64, 308
 MSK_putaij, 309
 MSK_putaijlist, 309
 MSK_putaijlist64, 310
 MSK_putarow, 310
 MSK_putarowlist, 310
 MSK_putarowlist64, 311
 MSK_putarowslice, 311
 MSK_putarowslice64, 312
 MSK_putatruncatetol, 312
 MSK_putbarablocktriplet, 313
 MSK_putbaraij, 313
 MSK_putbaraijlist, 314
 MSK_putbararowlist, 314
 MSK_putbarcblocktriplet, 315
 MSK_putbarcj, 315
 MSK_putbarsj, 316
 MSK_putbarvarname, 316
 MSK_putbarxj, 316
 MSK_putcallbackfunc, 317
 MSK_putcfix, 317
 MSK_putcj, 317
 MSK_putclist, 318
 MSK_putconbound, 318
 MSK_putconboundlist, 318
 MSK_putconboundlistconst, 319
 MSK_putconboundslice, 319
 MSK_putconboundsliceconst, 320
 MSK_putcone, 320
 MSK_putconename, 320
 MSK_putconname, 321
 MSK_putconsolutioni, 321
 MSK_putcslice, 321
 MSK_putdoupparam, 322
 MSK_putexitfunc, 322
 MSK_putintparam, 322
 MSK_putlicensecode, 323
 MSK_putlicensedebug, 323
 MSK_putlicensepath, 323
 MSK_putlicensewait, 324
 MSK_putmaxnumanz, 324
 MSK_putmaxnumbarvar, 324
 MSK_putmaxnumcon, 325
 MSK_putmaxnumcone, 325
 MSK_putmaxnumqnz, 325
 MSK_putmaxnumvar, 326
 MSK_putnadoupparam, 326
 MSK_putnaintparam, 326
 MSK_putnastrparam, 327
 MSK_putobjname, 327
 MSK_putobjsense, 327
 MSK_putparam, 327
 MSK_putqcon, 328
 MSK_putqconk, 329
 MSK_putqobj, 329
 MSK_putqobjij, 330
 MSK_putresponsefunc, 330
 MSK_putskc, 330
 MSK_putskcslice, 331
 MSK_putskx, 331
 MSK_putskxslice, 331
 MSK_putslc, 332
 MSK_putslcslice, 332
 MSK_putslx, 332
 MSK_putslxslice, 333
 MSK_putsnx, 333
 MSK_putsnxslice, 333
 MSK_putsolution, 334
 MSK_putsolutionyi, 334
 MSK_putstrparam, 335
 MSK_putsuc, 335
 MSK_putsucslice, 335
 MSK_putsux, 336
 MSK_putsuxslice, 336
 MSK_puttaskname, 336
 MSK_putvarbound, 336
 MSK_putvarboundlist, 337
 MSK_putvarboundlistconst, 337
 MSK_putvarboundslice, 338
 MSK_putvarboundsliceconst, 338
 MSK_putvarname, 338
 MSK_putvarsolutionj, 339
 MSK_putvartype, 339
 MSK_putvartypelist, 340
 MSK_putxc, 340
 MSK_putxcslice, 340
 MSK_putxx, 341
 MSK_putxxslice, 341

- MSK_puty, 341
- MSK_putyslice, 342
- MSK_readdata, 342
- MSK_readdataautoformat, 342
- MSK_readdataformat, 342
- MSK_readjsonstring, 343
- MSK_readlpstring, 343
- MSK_readopfstring, 343
- MSK_readparamfile, 344
- MSK_readptfstring, 344
- MSK_readsolution, 344
- MSK_readsummary, 344
- MSK_readtask, 345
- MSK_removebarvars, 345
- MSK_removecones, 345
- MSK_removecons, 346
- MSK_removevars, 346
- MSK_resize task, 346
- MSK_sensitivityreport, 347
- MSK_setdefaults, 347
- MSK_setupthreads, 347
- MSK_sktostr, 348
- MSK_solstatostr, 348
- MSK_solutiondef, 348
- MSK_solutionsummary, 348
- MSK_solvewithbasis, 349
- MSK_sparsetriangularsolvedense, 350
- MSK_strdupbgtask, 350
- MSK_strduptask, 351
- MSK_strtoconetype, 351
- MSK_strtosk, 351
- MSK_syeig, 351
- MSK_syevd, 352
- MSK_symnamtovalue, 352
- MSK_syrk, 353
- MSK_toconic, 353
- MSK_unlinkfuncfromenvstream, 354
- MSK_unlinkfuncfromtaskstream, 354
- MSK_updatesolutioninfo, 354
- MSK_utf8towchar, 355
- MSK_wchartoutf8, 355
- MSK_whichparam, 355
- MSK_writedata, 356
- MSK_writejsonsol, 356
- MSK_writeparamfile, 356
- MSK_writesolution, 357
- MSK_writetask, 357
- MSKcallbackfunc, 452
- MSKcallocfunc, 452
- MSKexitfunc, 453
- MSKfreefunc, 453
- MSKhreadfunc, 453
- MSKhwritefunc, 454
- MSKmallocfunc, 454
- MSKreallocfunc, 454
- MSKresponsefunc, 454
- MSKstreamfunc, 455

Parameters

- Double parameters, 369
- MSK_DPAR_ANA_SOL_INFEAS_TOL, 369
- MSK_DPAR_BASIS_REL_TOL_S, 369
- MSK_DPAR_BASIS_TOL_S, 369
- MSK_DPAR_BASIS_TOL_X, 369
- MSK_DPAR_CHECK_CONVEXITY_REL_TOL, 370
- MSK_DPAR_DATA_SYM_MAT_TOL, 370
- MSK_DPAR_DATA_SYM_MAT_TOL_HUGE, 370
- MSK_DPAR_DATA_SYM_MAT_TOL_LARGE, 370
- MSK_DPAR_DATA_TOL_AIJ_HUGE, 370
- MSK_DPAR_DATA_TOL_AIJ_LARGE, 370
- MSK_DPAR_DATA_TOL_BOUND_INF, 371
- MSK_DPAR_DATA_TOL_BOUND_WRN, 371
- MSK_DPAR_DATA_TOL_C_HUGE, 371
- MSK_DPAR_DATA_TOL_CJ_LARGE, 371
- MSK_DPAR_DATA_TOL_QIJ, 371
- MSK_DPAR_DATA_TOL_X, 371
- MSK_DPAR_INTPNT_CO_TOL_DFEAS, 371
- MSK_DPAR_INTPNT_CO_TOL_INFEAS, 372
- MSK_DPAR_INTPNT_CO_TOL_MU_RED, 372
- MSK_DPAR_INTPNT_CO_TOL_NEAR_REL, 372
- MSK_DPAR_INTPNT_CO_TOL_PFEAS, 372
- MSK_DPAR_INTPNT_CO_TOL_REL_GAP, 372
- MSK_DPAR_INTPNT_QO_TOL_DFEAS, 372
- MSK_DPAR_INTPNT_QO_TOL_INFEAS, 372
- MSK_DPAR_INTPNT_QO_TOL_MU_RED, 373
- MSK_DPAR_INTPNT_QO_TOL_NEAR_REL, 373
- MSK_DPAR_INTPNT_QO_TOL_PFEAS, 373
- MSK_DPAR_INTPNT_QO_TOL_REL_GAP, 373
- MSK_DPAR_INTPNT_TOL_DFEAS, 373
- MSK_DPAR_INTPNT_TOL_DSAFE, 373
- MSK_DPAR_INTPNT_TOL_INFEAS, 374
- MSK_DPAR_INTPNT_TOL_MU_RED, 374
- MSK_DPAR_INTPNT_TOL_PATH, 374
- MSK_DPAR_INTPNT_TOL_PFEAS, 374
- MSK_DPAR_INTPNT_TOL_PSAFE, 374
- MSK_DPAR_INTPNT_TOL_REL_GAP, 374
- MSK_DPAR_INTPNT_TOL_REL_STEP, 374
- MSK_DPAR_INTPNT_TOL_STEP_SIZE, 375
- MSK_DPAR_LOWER_OBJ_CUT, 375
- MSK_DPAR_LOWER_OBJ_CUT_FINITE_TRH, 375
- MSK_DPAR_MIO_MAX_TIME, 375
- MSK_DPAR_MIO_REL_GAP_CONST, 375
- MSK_DPAR_MIO_TOL_ABS_GAP, 375
- MSK_DPAR_MIO_TOL_ABS_RELAX_INT, 376
- MSK_DPAR_MIO_TOL_FEAS, 376
- MSK_DPAR_MIO_TOL_REL_DUAL_BOUND_IMPROVEMENT, 376
- MSK_DPAR_MIO_TOL_REL_GAP, 376
- MSK_DPAR_OPTIMIZER_MAX_TIME, 376
- MSK_DPAR_PRESOLVE_TOL_ABS_LINDEP, 376
- MSK_DPAR_PRESOLVE_TOL_AIJ, 376
- MSK_DPAR_PRESOLVE_TOL_REL_LINDEP, 377
- MSK_DPAR_PRESOLVE_TOL_S, 377
- MSK_DPAR_PRESOLVE_TOL_X, 377
- MSK_DPAR_QCQO_REFORMULATE_REL_DROP_TOL, 377
- MSK_DPAR_SEMIDEFINITE_TOL_APPROX, 377

MSK_DPAR_SIM_LU_TOL_REL_PIV, 377
 MSK_DPAR_SIMPLEX_ABS_TOL_PIV, 377
 MSK_DPAR_UPPER_OBJ_CUT, 378
 MSK_DPAR_UPPER_OBJ_CUT_FINITE_TRH, 378
 Integer parameters, 378
 MSK_IPAR_ANA_SOL_BASIS, 378
 MSK_IPAR_ANA_SOL_PRINT_VIOLATED, 378
 MSK_IPAR_AUTO_SORT_A_BEFORE_OPT, 378
 MSK_IPAR_AUTO_UPDATE_SOL_INFO, 378
 MSK_IPAR_BASIS_SOLVE_USE_PLUS_ONE, 379
 MSK_IPAR_BI_CLEAN_OPTIMIZER, 379
 MSK_IPAR_BI_IGNORE_MAX_ITER, 379
 MSK_IPAR_BI_IGNORE_NUM_ERROR, 379
 MSK_IPAR_BI_MAX_ITERATIONS, 379
 MSK_IPAR_CACHE_LICENSE, 380
 MSK_IPAR_CHECK_CONVEXITY, 380
 MSK_IPAR_COMPRESS_STATFILE, 380
 MSK_IPAR_INFEAS_GENERIC_NAMES, 380
 MSK_IPAR_INFEAS_PREFER_PRIMAL, 380
 MSK_IPAR_INFEAS_REPORT_AUTO, 380
 MSK_IPAR_INFEAS_REPORT_LEVEL, 380
 MSK_IPAR_INTPNT_BASIS, 381
 MSK_IPAR_INTPNT_DIFF_STEP, 381
 MSK_IPAR_INTPNT_HOTSTART, 381
 MSK_IPAR_INTPNT_MAX_ITERATIONS, 381
 MSK_IPAR_INTPNT_MAX_NUM_COR, 381
 MSK_IPAR_INTPNT_MAX_NUM_REFINEMENT_STEPS, 381
 MSK_IPAR_INTPNT_MULTI_THREAD, 382
 MSK_IPAR_INTPNT_OFF_COL_TRH, 382
 MSK_IPAR_INTPNT_ORDER_GP_NUM_SEEDS, 382
 MSK_IPAR_INTPNT_ORDER_METHOD, 382
 MSK_IPAR_INTPNT_PURIFY, 382
 MSK_IPAR_INTPNT_REGULARIZATION_USE, 382
 MSK_IPAR_INTPNT_SCALING, 383
 MSK_IPAR_INTPNT_SOLVE_FORM, 383
 MSK_IPAR_INTPNT_STARTING_POINT, 383
 MSK_IPAR_LICENSE_DEBUG, 383
 MSK_IPAR_LICENSE_PAUSE_TIME, 383
 MSK_IPAR_LICENSE_SUPPRESS_EXPIRE_WRNS, 383
 MSK_IPAR_LICENSE_TRH_EXPIRY_WRN, 383
 MSK_IPAR_LICENSE_WAIT, 384
 MSK_IPAR_LOG, 384
 MSK_IPAR_LOG_ANA_PRO, 384
 MSK_IPAR_LOG_BI, 384
 MSK_IPAR_LOG_BI_FREQ, 384
 MSK_IPAR_LOG_CHECK_CONVEXITY, 384
 MSK_IPAR_LOG_CUT_SECOND_OPT, 385
 MSK_IPAR_LOG_EXPAND, 385
 MSK_IPAR_LOG_FEAS_REPAIR, 385
 MSK_IPAR_LOG_FILE, 385
 MSK_IPAR_LOG_INCLUDE_SUMMARY, 385
 MSK_IPAR_LOG_INFEAS_ANA, 385
 MSK_IPAR_LOG_INTPNT, 386
 MSK_IPAR_LOG_LOCAL_INFO, 386
 MSK_IPAR_LOG_MIO, 386
 MSK_IPAR_LOG_MIO_FREQ, 386
 MSK_IPAR_LOG_ORDER, 386
 MSK_IPAR_LOG_PREOLVE, 386
 MSK_IPAR_LOG_RESPONSE, 387
 MSK_IPAR_LOG_SENSITIVITY, 387
 MSK_IPAR_LOG_SENSITIVITY_OPT, 387
 MSK_IPAR_LOG_SIM, 387
 MSK_IPAR_LOG_SIM_FREQ, 387
 MSK_IPAR_LOG_SIM_MINOR, 387
 MSK_IPAR_LOG_STORAGE, 388
 MSK_IPAR_MAX_NUM_WARNINGS, 388
 MSK_IPAR_MIO_BRANCH_DIR, 388
 MSK_IPAR_MIO_CONIC_OUTER_APPROXIMATION, 388
 MSK_IPAR_MIO_CUT_CLIQUE, 388
 MSK_IPAR_MIO_CUT_CMIR, 388
 MSK_IPAR_MIO_CUT_GMI, 388
 MSK_IPAR_MIO_CUT_IMPLIED_BOUND, 389
 MSK_IPAR_MIO_CUT_KNAPSACK_COVER, 389
 MSK_IPAR_MIO_CUT_SELECTION_LEVEL, 389
 MSK_IPAR_MIO_FEASPUMP_LEVEL, 389
 MSK_IPAR_MIO_HEURISTIC_LEVEL, 389
 MSK_IPAR_MIO_MAX_NUM_BRANCHES, 389
 MSK_IPAR_MIO_MAX_NUM_RELAXS, 390
 MSK_IPAR_MIO_MAX_NUM_ROOT_CUT_ROUNDS, 390
 MSK_IPAR_MIO_MAX_NUM_SOLUTIONS, 390
 MSK_IPAR_MIO_MODE, 390
 MSK_IPAR_MIO_NODE_OPTIMIZER, 390
 MSK_IPAR_MIO_NODE_SELECTION, 390
 MSK_IPAR_MIO_PERSPECTIVE_REFORMULATE, 390
 MSK_IPAR_MIO_PROBING_LEVEL, 391
 MSK_IPAR_MIO_PROPAGATE_OBJECTIVE_CONSTRAINT, 391
 MSK_IPAR_MIO_RINS_MAX_NODES, 391
 MSK_IPAR_MIO_ROOT_OPTIMIZER, 391
 MSK_IPAR_MIO_ROOT_REPEAT_PREOLVE_LEVEL, 391
 MSK_IPAR_MIO_SEED, 392
 MSK_IPAR_MIO_VB_DETECTION_LEVEL, 392
 MSK_IPAR_MT_SPINCOUNT, 392
 MSK_IPAR_NUM_THREADS, 392
 MSK_IPAR_OPF_WRITE_HEADER, 392
 MSK_IPAR_OPF_WRITE_HINTS, 392
 MSK_IPAR_OPF_WRITE_LINE_LENGTH, 393
 MSK_IPAR_OPF_WRITE_PARAMETERS, 393
 MSK_IPAR_OPF_WRITE_PROBLEM, 393
 MSK_IPAR_OPF_WRITE_SOL_BAS, 393
 MSK_IPAR_OPF_WRITE_SOL_ITG, 393
 MSK_IPAR_OPF_WRITE_SOL_ITR, 393
 MSK_IPAR_OPF_WRITE_SOLUTIONS, 393
 MSK_IPAR_OPTIMIZER, 394
 MSK_IPAR_PARAM_READ_CASE_NAME, 394
 MSK_IPAR_PARAM_READ_IGN_ERROR, 394
 MSK_IPAR_PREOLVE_ELIMINATOR_MAX_FILL, 394
 MSK_IPAR_PREOLVE_ELIMINATOR_MAX_NUM_TRIES, 394
 MSK_IPAR_PREOLVE_LEVEL, 394
 MSK_IPAR_PREOLVE_LINDEP_ABS_WORK_TRH, 395
 MSK_IPAR_PREOLVE_LINDEP_REL_WORK_TRH, 395
 MSK_IPAR_PREOLVE_LINDEP_USE, 395
 MSK_IPAR_PREOLVE_MAX_NUM_PASS, 395

MSK_IPAR_PREOLVE_MAX_NUM_REDUCTIONS, 395
 MSK_IPAR_PREOLVE_USE, 395
 MSK_IPAR_PRIMAL_REPAIR_OPTIMIZER, 395
 MSK_IPAR_PTF_WRITE_TRANSFORM, 396
 MSK_IPAR_READ_DEBUG, 396
 MSK_IPAR_READ_KEEP_FREE_CON, 396
 MSK_IPAR_READ_LP_DROP_NEW_VARS_IN_BOU, 396
 MSK_IPAR_READ_LP_QUOTED_NAMES, 396
 MSK_IPAR_READ_MPS_FORMAT, 396
 MSK_IPAR_READ_MPS_WIDTH, 397
 MSK_IPAR_READ_TASK_IGNORE_PARAM, 397
 MSK_IPAR_REMOVE_UNUSED_SOLUTIONS, 397
 MSK_IPAR_SENSITIVITY_ALL, 397
 MSK_IPAR_SENSITIVITY_OPTIMIZER, 397
 MSK_IPAR_SENSITIVITY_TYPE, 397
 MSK_IPAR_SIM_BASIS_FACTOR_USE, 397
 MSK_IPAR_SIM_DEGEN, 398
 MSK_IPAR_SIM_DUAL_CRASH, 398
 MSK_IPAR_SIM_DUAL_PHASEONE_METHOD, 398
 MSK_IPAR_SIM_DUAL_RESTRICT_SELECTION, 398
 MSK_IPAR_SIM_DUAL_SELECTION, 398
 MSK_IPAR_SIM_EXPLOIT_DUPVEC, 398
 MSK_IPAR_SIM_HOTSTART, 399
 MSK_IPAR_SIM_HOTSTART_LU, 399
 MSK_IPAR_SIM_MAX_ITERATIONS, 399
 MSK_IPAR_SIM_MAX_NUM_SETBACKS, 399
 MSK_IPAR_SIM_NON_SINGULAR, 399
 MSK_IPAR_SIM_PRIMAL_CRASH, 399
 MSK_IPAR_SIM_PRIMAL_PHASEONE_METHOD, 399
 MSK_IPAR_SIM_PRIMAL_RESTRICT_SELECTION, 400
 MSK_IPAR_SIM_PRIMAL_SELECTION, 400
 MSK_IPAR_SIM_REFACTOR_FREQ, 400
 MSK_IPAR_SIM_REFORMULATION, 400
 MSK_IPAR_SIM_SAVE_LU, 400
 MSK_IPAR_SIM_SCALING, 400
 MSK_IPAR_SIM_SCALING_METHOD, 401
 MSK_IPAR_SIM_SEED, 401
 MSK_IPAR_SIM_SOLVE_FORM, 401
 MSK_IPAR_SIM_STABILITY_PRIORITY, 401
 MSK_IPAR_SIM_SWITCH_OPTIMIZER, 401
 MSK_IPAR_SOL_FILTER_KEEP_BASIC, 401
 MSK_IPAR_SOL_FILTER_KEEP_RANGED, 401
 MSK_IPAR_SOL_READ_NAME_WIDTH, 402
 MSK_IPAR_SOL_READ_WIDTH, 402
 MSK_IPAR_SOLUTION_CALLBACK, 402
 MSK_IPAR_TIMING_LEVEL, 402
 MSK_IPAR_WRITE_BAS_CONSTRAINTS, 402
 MSK_IPAR_WRITE_BAS_HEAD, 402
 MSK_IPAR_WRITE_BAS_VARIABLES, 402
 MSK_IPAR_WRITE_COMPRESSION, 403
 MSK_IPAR_WRITE_DATA_PARAM, 403
 MSK_IPAR_WRITE_FREE_CON, 403
 MSK_IPAR_WRITE_GENERIC_NAMES, 403
 MSK_IPAR_WRITE_GENERIC_NAMES_IO, 403
 MSK_IPAR_WRITE_IGNORE_INCOMPATIBLE_ITEMS, 403
 MSK_IPAR_WRITE_INT_CONSTRAINTS, 403
 MSK_IPAR_WRITE_INT_HEAD, 404
 MSK_IPAR_WRITE_INT_VARIABLES, 404
 MSK_IPAR_WRITE_LP_FULL_OBJ, 404
 MSK_IPAR_WRITE_LP_LINE_WIDTH, 404
 MSK_IPAR_WRITE_LP_QUOTED_NAMES, 404
 MSK_IPAR_WRITE_LP_STRICT_FORMAT, 404
 MSK_IPAR_WRITE_LP_TERMS_PER_LINE, 404
 MSK_IPAR_WRITE_MPS_FORMAT, 405
 MSK_IPAR_WRITE_MPS_INT, 405
 MSK_IPAR_WRITE_PRECISION, 405
 MSK_IPAR_WRITE_SOL_BARVARIABLES, 405
 MSK_IPAR_WRITE_SOL_CONSTRAINTS, 405
 MSK_IPAR_WRITE_SOL_HEAD, 405
 MSK_IPAR_WRITE_SOL_IGNORE_INVALID_NAMES, 405
 MSK_IPAR_WRITE_SOL_VARIABLES, 406
 MSK_IPAR_WRITE_TASK_INC_SOL, 406
 MSK_IPAR_WRITE_XML_MODE, 406
 String parameters, 406
 MSK_SPAR_BAS_SOL_FILE_NAME, 406
 MSK_SPAR_DATA_FILE_NAME, 406
 MSK_SPAR_DEBUG_FILE_NAME, 406
 MSK_SPAR_INT_SOL_FILE_NAME, 406
 MSK_SPAR_ITR_SOL_FILE_NAME, 406
 MSK_SPAR_MIO_DEBUG_STRING, 407
 MSK_SPAR_PARAM_COMMENT_SIGN, 407
 MSK_SPAR_PARAM_READ_FILE_NAME, 407
 MSK_SPAR_PARAM_WRITE_FILE_NAME, 407
 MSK_SPAR_READ_MPS_BOU_NAME, 407
 MSK_SPAR_READ_MPS_OBJ_NAME, 407
 MSK_SPAR_READ_MPS_RAN_NAME, 407
 MSK_SPAR_READ_MPS_RHS_NAME, 408
 MSK_SPAR_REMOTE_ACCESS_TOKEN, 408
 MSK_SPAR_SENSITIVITY_FILE_NAME, 408
 MSK_SPAR_SENSITIVITY_RES_FILE_NAME, 408
 MSK_SPAR_SOL_FILTER_XC_LOW, 408
 MSK_SPAR_SOL_FILTER_XC_UPR, 408
 MSK_SPAR_SOL_FILTER_XX_LOW, 408
 MSK_SPAR_SOL_FILTER_XX_UPR, 408
 MSK_SPAR_STAT_FILE_NAME, 409
 MSK_SPAR_STAT_KEY, 409
 MSK_SPAR_STAT_NAME, 409
 MSK_SPAR_WRITE_LP_GEN_VAR_NAME, 409

Response codes

Termination, 410
 MSK_RES_OK, 410
 MSK_RES_TRM_INTERNAL, 410
 MSK_RES_TRM_INTERNAL_STOP, 410
 MSK_RES_TRM_MAX_ITERATIONS, 410
 MSK_RES_TRM_MAX_NUM_SETBACKS, 410
 MSK_RES_TRM_MAX_TIME, 410
 MSK_RES_TRM_MIO_NUM_BRANCHES, 410
 MSK_RES_TRM_MIO_NUM_RELAXS, 410
 MSK_RES_TRM_NUM_MAX_NUM_INT_SOLUTIONS, 410
 MSK_RES_TRM_NUMERICAL_PROBLEM, 410
 MSK_RES_TRM_OBJECTIVE_RANGE, 410
 MSK_RES_TRM_STALL, 410
 MSK_RES_TRM_USER_CALLBACK, 410

Warnings, 410

MSK_RES_WRN_ANA_ALMOST_INT_BOUNDS, 412

MSK_RES_WRN_ANA_C_ZERO, 412

MSK_RES_WRN_ANA_CLOSE_BOUNDS, 412

MSK_RES_WRN_ANA_EMPTY_COLS, 412

MSK_RES_WRN_ANA_LARGE_BOUNDS, 412

MSK_RES_WRN_DROPPED_NZ_QOBJ, 411

MSK_RES_WRN_DUPLICATE_BARVARIABLE_NAMES, 412

MSK_RES_WRN_DUPLICATE_CONE_NAMES, 412

MSK_RES_WRN_DUPLICATE_CONSTRAINT_NAMES, 412

MSK_RES_WRN_DUPLICATE_VARIABLE_NAMES, 412

MSK_RES_WRN_ELIMINATOR_SPACE, 412

MSK_RES_WRN_EMPTY_NAME, 411

MSK_RES_WRN_EXP_CONES_WITH_VARIABLES_FIXED_AT_ZERO, 412

MSK_RES_WRN_IGNORE_INTEGER, 411

MSK_RES_WRN_INCOMPLETE_LINEAR_DEPENDENCY_CHECK, 412

MSK_RES_WRN_LARGE_AIJ, 410

MSK_RES_WRN_LARGE_BOUND, 410

MSK_RES_WRN_LARGE_CJ, 410

MSK_RES_WRN_LARGE_CON_FX, 410

MSK_RES_WRN_LARGE_LO_BOUND, 410

MSK_RES_WRN_LARGE_UP_BOUND, 410

MSK_RES_WRN_LICENSE_EXPIRE, 411

MSK_RES_WRN_LICENSE_FEATURE_EXPIRE, 411

MSK_RES_WRN_LICENSE_SERVER, 411

MSK_RES_WRN_LP_DROP_VARIABLE, 411

MSK_RES_WRN_LP_OLD_QUAD_FORMAT, 411

MSK_RES_WRN_MIO_INFEASIBLE_FINAL, 411

MSK_RES_WRN_MPS_SPLIT_BOU_VECTOR, 411

MSK_RES_WRN_MPS_SPLIT_RAN_VECTOR, 411

MSK_RES_WRN_MPS_SPLIT_RHS_VECTOR, 411

MSK_RES_WRN_NAME_MAX_LEN, 411

MSK_RES_WRN_NO_DUALIZER, 413

MSK_RES_WRN_NO_GLOBAL_OPTIMIZER, 411

MSK_RES_WRN_NZ_IN_UPR_TRI, 411

MSK_RES_WRN_OPEN_PARAM_FILE, 410

MSK_RES_WRN_PARAM_IGNORED_CMIO, 412

MSK_RES_WRN_PARAM_NAME_DOU, 411

MSK_RES_WRN_PARAM_NAME_INT, 411

MSK_RES_WRN_PARAM_NAME_STR, 411

MSK_RES_WRN_PARAM_STR_VALUE, 411

MSK_RES_WRN_POW_CONES_WITH_ROOT_FIXED_AT_ZERO, 412

MSK_RES_WRN_PRESOLVE_OUTOFSPACE, 412

MSK_RES_WRN_QUAD_CONES_WITH_ROOT_FIXED_AT_ZERO, 412

MSK_RES_WRN_RQUAD_CONES_WITH_ROOT_FIXED_AT_ZERO, 412

MSK_RES_WRN_SOL_FILE_IGNORED_CON, 411

MSK_RES_WRN_SOL_FILE_IGNORED_VAR, 411

MSK_RES_WRN_SOL_FILTER, 411

MSK_RES_WRN_SPAR_MAX_LEN, 411

MSK_RES_WRN_SYM_MAT_LARGE, 413

MSK_RES_WRN_TOO_FEW_BASIS_VARS, 411

MSK_RES_WRN_TOO_MANY_BASIS_VARS, 411

MSK_RES_WRN_UNDEF_SOL_FILE_NAME, 411

MSK_RES_WRN_USING_GENERIC_NAMES, 411

MSK_RES_WRN_WRITE_CHANGED_NAMES, 412

MSK_RES_WRN_WRITE_DISCARDED_CFIX, 412

MSK_RES_WRN_ZERO_AIJ, 411

MSK_RES_WRN_ZEROS_IN_SPARSE_COL, 412

MSK_RES_WRN_ZEROS_IN_SPARSE_ROW, 412

Errors, 413

MSK_RES_ERR_AD_INVALID_CODELIST, 424

MSK_RES_ERR_API_ARRAY_TOO_SMALL, 423

MSK_RES_ERR_API_CB_CONNECT, 423

MSK_RES_ERR_API_FATAL_ERROR, 423

MSK_RES_ERR_API_INTERNAL, 423

MSK_RES_ERR_APPENDING_TOO_BIG_CONE, 420

MSK_RES_ERR_ARG_IS_TOO_LARGE, 418

MSK_RES_ERR_ARG_IS_TOO_SMALL, 418

MSK_RES_ERR_ARGUMENT_DIMENSION, 417

MSK_RES_ERR_ARGUMENT_IS_TOO_LARGE, 425

MSK_RES_ERR_ARGUMENT_LENNEQ, 417

MSK_RES_ERR_ARGUMENT_PERM_ARRAY, 420

MSK_RES_ERR_ARGUMENT_TYPE, 417

MSK_RES_ERR_BAR_VAR_DIM, 424

MSK_RES_ERR_BASIS, 419

MSK_RES_ERR_BASIS_FACTOR, 422

MSK_RES_ERR_BASIS_SINGULAR, 422

MSK_RES_ERR_BLANK_NAME, 415

MSK_RES_ERR_CBF_DUPLICATE_ACOORD, 426

MSK_RES_ERR_CBF_DUPLICATE_BCOORD, 426

MSK_RES_ERR_CBF_DUPLICATE_CON, 426

MSK_RES_ERR_CBF_DUPLICATE_INT, 426

MSK_RES_ERR_CBF_DUPLICATE_OBJ, 426

MSK_RES_ERR_CBF_DUPLICATE_OBJACCOORD, 426

MSK_RES_ERR_CBF_DUPLICATE_POW_CONES, 426

MSK_RES_ERR_CBF_DUPLICATE_POW_STAR_CONES, 426

MSK_RES_ERR_CBF_DUPLICATE_PSDVAR, 426

MSK_RES_ERR_CBF_DUPLICATE_VAR, 426

MSK_RES_ERR_CBF_INVALID_CON_TYPE, 426

MSK_RES_ERR_CBF_INVALID_DIMENSION_OF_CONES, 427

MSK_RES_ERR_CBF_INVALID_DOMAIN_DIMENSION, 426

MSK_RES_ERR_CBF_INVALID_EXP_DIMENSION, 426

MSK_RES_ERR_CBF_INVALID_INT_INDEX, 426

MSK_RES_ERR_CBF_INVALID_NUMBER_OF_CONES, 427

MSK_RES_ERR_CBF_INVALID_POWER, 426

MSK_RES_ERR_CBF_INVALID_POWER_CONE_INDEX, 426

MSK_RES_ERR_CBF_INVALID_POWER_STAR_CONE_INDEX, 426

MSK_RES_ERR_CBF_INVALID_PSDVAR_DIMENSION, 426

MSK_RES_ERR_CBF_INVALID_VAR_TYPE, 426

MSK_RES_ERR_CBF_NO_VARIABLES, 425

MSK_RES_ERR_CBF_NO_VERSION_SPECIFIED, 426

MSK_RES_ERR_CBF_OBJ_SENSE, 425

MSK_RES_ERR_CBF_PARSE, 425

MSK_RES_ERR_CBF_POWER_CONE_IS_TOO_LONG, 426
 MSK_RES_ERR_CBF_POWER_CONE_MISMATCH, 427
 MSK_RES_ERR_CBF_POWER_STAR_CONE_MISMATCH, 427
 MSK_RES_ERR_CBF_SYNTAX, 426
 MSK_RES_ERR_CBF_TOO_FEW_CONSTRAINTS, 426
 MSK_RES_ERR_CBF_TOO_FEW_INTS, 426
 MSK_RES_ERR_CBF_TOO_FEW_PSDVAR, 426
 MSK_RES_ERR_CBF_TOO_FEW_VARIABLES, 426
 MSK_RES_ERR_CBF_TOO_MANY_CONSTRAINTS, 425
 MSK_RES_ERR_CBF_TOO_MANY_INTS, 426
 MSK_RES_ERR_CBF_TOO_MANY_VARIABLES, 426
 MSK_RES_ERR_CBF_UNHANDLED_POWER_CONE_TYPE, 427
 MSK_RES_ERR_CBF_UNHANDLED_POWER_STAR_CONE_TYPE, 427
 MSK_RES_ERR_CBF_UNSUPPORTED, 426
 MSK_RES_ERR_CON_Q_NOT_NSD, 420
 MSK_RES_ERR_CON_Q_NOT_PSD, 420
 MSK_RES_ERR_CONE_INDEX, 420
 MSK_RES_ERR_CONE_OVERLAP, 420
 MSK_RES_ERR_CONE_OVERLAP_APPEND, 420
 MSK_RES_ERR_CONE_PARAMETER, 420
 MSK_RES_ERR_CONE_REP_VAR, 420
 MSK_RES_ERR_CONE_SIZE, 420
 MSK_RES_ERR_CONE_TYPE, 420
 MSK_RES_ERR_CONE_TYPE_STR, 420
 MSK_RES_ERR_DATA_FILE_EXT, 414
 MSK_RES_ERR_DUP_NAME, 415
 MSK_RES_ERR_DUPLICATE_AIJ, 421
 MSK_RES_ERR_DUPLICATE_BARVARIABLE_NAMES, 425
 MSK_RES_ERR_DUPLICATE_CONE_NAMES, 425
 MSK_RES_ERR_DUPLICATE_CONSTRAINT_NAMES, 425
 MSK_RES_ERR_DUPLICATE_VARIABLE_NAMES, 425
 MSK_RES_ERR_END_OF_FILE, 414
 MSK_RES_ERR_FACTOR, 422
 MSK_RES_ERR_FEASREPAIR_CANNOT_RELAX, 422
 MSK_RES_ERR_FEASREPAIR_INCONSISTENT_BOUND, 422
 MSK_RES_ERR_FEASREPAIR_SOLVING_RELAXED, 422
 MSK_RES_ERR_FILE_LICENSE, 413
 MSK_RES_ERR_FILE_OPEN, 414
 MSK_RES_ERR_FILE_READ, 414
 MSK_RES_ERR_FILE_WRITE, 414
 MSK_RES_ERR_FINAL_SOLUTION, 422
 MSK_RES_ERR_FIRST, 422
 MSK_RES_ERR_FIRSTI, 419
 MSK_RES_ERR_FIRSTJ, 419
 MSK_RES_ERR_FIXED_BOUND_VALUES, 421
 MSK_RES_ERR_FLEXLM, 413
 MSK_RES_ERR_FORMAT_STRING, 415
 MSK_RES_ERR_GLOBAL_INV_CONIC_PROBLEM, 422
 MSK_RES_ERR_HUGE_AIJ, 421
 MSK_RES_ERR_HUGE_C, 420
 MSK_RES_ERR_IDENTICAL_TASKS, 424
 MSK_RES_ERR_IN_ARGUMENT, 417
 MSK_RES_ERR_INDEX, 418
 MSK_RES_ERR_INDEX_ARR_IS_TOO_LARGE, 418
 MSK_RES_ERR_INDEX_ARR_IS_TOO_SMALL, 418
 MSK_RES_ERR_INDEX_IS_TOO_LARGE, 417
 MSK_RES_ERR_INDEX_IS_TOO_SMALL, 417
 MSK_RES_ERR_INF_DOU_INDEX, 417
 MSK_RES_ERR_INF_DOU_NAME, 418
 MSK_RES_ERR_INF_INT_INDEX, 418
 MSK_RES_ERR_INF_INT_NAME, 418
 MSK_RES_ERR_INF_LINT_INDEX, 418
 MSK_RES_ERR_INF_LINT_NAME, 418
 MSK_RES_ERR_INF_TYPE, 418
 MSK_RES_ERR_INFEAS_UNDEFINED, 424
 MSK_RES_ERR_INFINITE_BOUND, 421
 MSK_RES_ERR_INT64_TO_INT32_CAST, 424
 MSK_RES_ERR_INTERNAL, 423
 MSK_RES_ERR_INTERNAL_TEST_FAILED, 424
 MSK_RES_ERR_INV_APTRE, 418
 MSK_RES_ERR_INV_BK, 419
 MSK_RES_ERR_INV_BKC, 419
 MSK_RES_ERR_INV_BKX, 419
 MSK_RES_ERR_INV_CONE_TYPE, 419
 MSK_RES_ERR_INV_CONE_TYPE_STR, 419
 MSK_RES_ERR_INV_MARKI, 423
 MSK_RES_ERR_INV_MARKJ, 423
 MSK_RES_ERR_INV_NAME_ITEM, 419
 MSK_RES_ERR_INV_NUMI, 423
 MSK_RES_ERR_INV_NUMJ, 423
 MSK_RES_ERR_INV_OPTIMIZER, 422
 MSK_RES_ERR_INV_PROBLEM, 422
 MSK_RES_ERR_INV_QCON_SUBI, 421
 MSK_RES_ERR_INV_QCON_SUBJ, 421
 MSK_RES_ERR_INV_QCON_SUBK, 421
 MSK_RES_ERR_INV_QCON_VAL, 421
 MSK_RES_ERR_INV_QOBJ_SUBI, 421
 MSK_RES_ERR_INV_QOBJ_SUBJ, 421
 MSK_RES_ERR_INV_QOBJ_VAL, 421
 MSK_RES_ERR_INV_SK, 419
 MSK_RES_ERR_INV_SK_STR, 419
 MSK_RES_ERR_INV_SKC, 419
 MSK_RES_ERR_INV_SKN, 419
 MSK_RES_ERR_INV_SKX, 419
 MSK_RES_ERR_INV_VAR_TYPE, 419
 MSK_RES_ERR_INVALID_AIJ, 421
 MSK_RES_ERR_INVALID_AMPL_STUB, 424
 MSK_RES_ERR_INVALID_BARVAR_NAME, 415
 MSK_RES_ERR_INVALID_COMPRESSION, 423
 MSK_RES_ERR_INVALID_CON_NAME, 415
 MSK_RES_ERR_INVALID_CONE_NAME, 415
 MSK_RES_ERR_INVALID_FILE_FORMAT_FOR_CFIX, 424
 MSK_RES_ERR_INVALID_FILE_FORMAT_FOR_CONES, 425
 MSK_RES_ERR_INVALID_FILE_FORMAT_FOR_FREE_CONSTRAINTS, 424
 MSK_RES_ERR_INVALID_FILE_FORMAT_FOR_NONLINEAR, 425
 MSK_RES_ERR_INVALID_FILE_FORMAT_FOR_RANGED_CONSTRAINTS, 424

MSK_RES_ERR_INVALID_FILE_FORMAT_FOR_SYM_MAT, 424
 MSK_RES_ERR_INVALID_FILE_NAME, 414
 MSK_RES_ERR_INVALID_FORMAT_TYPE, 419
 MSK_RES_ERR_INVALID_IDX, 418
 MSK_RES_ERR_INVALID_IOMODE, 423
 MSK_RES_ERR_INVALID_MAX_NUM, 418
 MSK_RES_ERR_INVALID_NAME_IN_SOL_FILE, 417
 MSK_RES_ERR_INVALID_OBJ_NAME, 415
 MSK_RES_ERR_INVALID_OBJECTIVE_SENSE, 421
 MSK_RES_ERR_INVALID_PROBLEM_TYPE, 425
 MSK_RES_ERR_INVALID_SOL_FILE_NAME, 414
 MSK_RES_ERR_INVALID_STREAM, 414
 MSK_RES_ERR_INVALID_SURPLUS, 419
 MSK_RES_ERR_INVALID_SYM_MAT_DIM, 424
 MSK_RES_ERR_INVALID_TASK, 414
 MSK_RES_ERR_INVALID_UTF8, 423
 MSK_RES_ERR_INVALID_VAR_NAME, 415
 MSK_RES_ERR_INVALID_WCHAR, 423
 MSK_RES_ERR_INVALID_WHICH_SOL, 418
 MSK_RES_ERR_JSON_DATA, 417
 MSK_RES_ERR_JSON_FORMAT, 417
 MSK_RES_ERR_JSON_MISSING_DATA, 417
 MSK_RES_ERR_JSON_NUMBER_OVERFLOW, 417
 MSK_RES_ERR_JSON_STRING, 417
 MSK_RES_ERR_JSON_SYNTAX, 417
 MSK_RES_ERR_LAST, 422
 MSK_RES_ERR_LASTI, 419
 MSK_RES_ERR_LASTJ, 419
 MSK_RES_ERR_LAU_ARG_K, 425
 MSK_RES_ERR_LAU_ARG_M, 425
 MSK_RES_ERR_LAU_ARG_N, 425
 MSK_RES_ERR_LAU_ARG_TRANS, 425
 MSK_RES_ERR_LAU_ARG_TRANSA, 425
 MSK_RES_ERR_LAU_ARG_TRANSB, 425
 MSK_RES_ERR_LAU_ARG_UPLO, 425
 MSK_RES_ERR_LAU_INVALID_LOWER_TRIANGULAR_MATRIX, 425
 MSK_RES_ERR_LAU_INVALID_SPARSE_SYMMETRIC_MATRIX, 425
 MSK_RES_ERR_LAU_NOT_POSITIVE_DEFINITE, 425
 MSK_RES_ERR_LAU_SINGULAR_MATRIX, 425
 MSK_RES_ERR_LAU_UNKNOWN, 425
 MSK_RES_ERR_LICENSE, 413
 MSK_RES_ERR_LICENSE_CANNOT_ALLOCATE, 413
 MSK_RES_ERR_LICENSE_CANNOT_CONNECT, 413
 MSK_RES_ERR_LICENSE_EXPIRED, 413
 MSK_RES_ERR_LICENSE_FEATURE, 413
 MSK_RES_ERR_LICENSE_INVALID_HOSTID, 413
 MSK_RES_ERR_LICENSE_MAX, 413
 MSK_RES_ERR_LICENSE_MOSEKLM_DAEMON, 413
 MSK_RES_ERR_LICENSE_NO_SERVER_LINE, 414
 MSK_RES_ERR_LICENSE_NO_SERVER_SUPPORT, 413
 MSK_RES_ERR_LICENSE_SERVER, 413
 MSK_RES_ERR_LICENSE_SERVER_VERSION, 413
 MSK_RES_ERR_LICENSE_VERSION, 413
 MSK_RES_ERR_LINK_FILE_DLL, 414
 MSK_RES_ERR_LIVING_TASKS, 414
 MSK_RES_ERR_LOWER_BOUND_IS_A_NAN, 421
 MSK_RES_ERR_LP_DUP_SLACK_NAME, 416
 MSK_RES_ERR_LP_EMPTY, 416
 MSK_RES_ERR_LP_FILE_FORMAT, 416
 MSK_RES_ERR_LP_FORMAT, 416
 MSK_RES_ERR_LP_FREE_CONSTRAINT, 416
 MSK_RES_ERR_LP_INCOMPATIBLE, 416
 MSK_RES_ERR_LP_INVALID_CON_NAME, 417
 MSK_RES_ERR_LP_INVALID_VAR_NAME, 416
 MSK_RES_ERR_LP_WRITE_CONIC_PROBLEM, 416
 MSK_RES_ERR_LP_WRITE_GECO_PROBLEM, 416
 MSK_RES_ERR_LU_MAX_NUM_TRIES, 423
 MSK_RES_ERR_MAX_LEN_IS_TOO_SMALL, 420
 MSK_RES_ERR_MAXNUMBARVAR, 418
 MSK_RES_ERR_MAXNUMCON, 418
 MSK_RES_ERR_MAXNUMCONE, 420
 MSK_RES_ERR_MAXNUMQNZ, 418
 MSK_RES_ERR_MAXNUMVAR, 418
 MSK_RES_ERR_MIO_INTERNAL, 425
 MSK_RES_ERR_MIO_INVALID_NODE_OPTIMIZER, 427
 MSK_RES_ERR_MIO_INVALID_ROOT_OPTIMIZER, 427
 MSK_RES_ERR_MIO_NO_OPTIMIZER, 422
 MSK_RES_ERR_MISSING_LICENSE_FILE, 413
 MSK_RES_ERR_MIXED_CONIC_AND_NL, 422
 MSK_RES_ERR_MPS_CONE_OVERLAP, 416
 MSK_RES_ERR_MPS_CONE_REPEAT, 416
 MSK_RES_ERR_MPS_CONE_TYPE, 416
 MSK_RES_ERR_MPS_DUPLICATE_Q_ELEMENT, 416
 MSK_RES_ERR_MPS_FILE, 415
 MSK_RES_ERR_MPS_INV_BOUND_KEY, 415
 MSK_RES_ERR_MPS_INV_CON_KEY, 415
 MSK_RES_ERR_MPS_INV_FIELD, 415
 MSK_RES_ERR_MPS_INV_MARKER, 415
 MSK_RES_ERR_MPS_INV_SEC_NAME, 415
 MSK_RES_ERR_MPS_INV_SEC_ORDER, 415
 MSK_RES_ERR_MPS_INVALID_OBJ_NAME, 416
 MSK_RES_ERR_MPS_INVALID_OBJSENSE, 416
 MSK_RES_ERR_MPS_MUL_CON_NAME, 415
 MSK_RES_ERR_MPS_MUL_CSEC, 415
 MSK_RES_ERR_MPS_MUL_QOBJ, 415
 MSK_RES_ERR_MPS_MUL_QSEC, 415
 MSK_RES_ERR_MPS_NO_OBJECTIVE, 415
 MSK_RES_ERR_MPS_NON_SYMMETRIC_Q, 416
 MSK_RES_ERR_MPS_NULL_CON_NAME, 415
 MSK_RES_ERR_MPS_NULL_VAR_NAME, 415
 MSK_RES_ERR_MPS_SPLITTED_VAR, 415
 MSK_RES_ERR_MPS_TAB_IN_FIELD2, 416
 MSK_RES_ERR_MPS_TAB_IN_FIELD3, 416
 MSK_RES_ERR_MPS_TAB_IN_FIELD5, 416
 MSK_RES_ERR_MPS_UNDEF_CON_NAME, 415
 MSK_RES_ERR_MPS_UNDEF_VAR_NAME, 415
 MSK_RES_ERR_MUL_A_ELEMENT, 419
 MSK_RES_ERR_NAME_IS_NULL, 422
 MSK_RES_ERR_NAME_MAX_LEN, 422
 MSK_RES_ERR_NAN_IN_BLC, 421
 MSK_RES_ERR_NAN_IN_BLC, 421
 MSK_RES_ERR_NAN_IN_BUC, 421
 MSK_RES_ERR_NAN_IN_BUX, 421

MSK_RES_ERR_NAN_IN_C, 421
 MSK_RES_ERR_NAN_IN_DOUBLE_DATA, 421
 MSK_RES_ERR_NEGATIVE_APPEND, 422
 MSK_RES_ERR_NEGATIVE_SURPLUS, 422
 MSK_RES_ERR_NEWER_DLL, 414
 MSK_RES_ERR_NO_BARS_FOR_SOLUTION, 424
 MSK_RES_ERR_NO_BARX_FOR_SOLUTION, 424
 MSK_RES_ERR_NO_BASIS_SOL, 422
 MSK_RES_ERR_NO_DUAL_FOR_ITG_SOL, 423
 MSK_RES_ERR_NO_DUAL_INFEAS_CER, 423
 MSK_RES_ERR_NO_INIT_ENV, 414
 MSK_RES_ERR_NO_OPTIMIZER_VAR_TYPE, 422
 MSK_RES_ERR_NO_PRIMAL_INFEAS_CER, 423
 MSK_RES_ERR_NO_SNX_FOR_BAS_SOL, 423
 MSK_RES_ERR_NO_SOLUTION_IN_CALLBACK, 423
 MSK_RES_ERR_NON_UNIQUE_ARRAY, 425
 MSK_RES_ERR_NONCONVEX, 420
 MSK_RES_ERR_NONLINEAR_EQUALITY, 420
 MSK_RES_ERR_NONLINEAR_RANGED, 420
 MSK_RES_ERR_NULL_ENV, 414
 MSK_RES_ERR_NULL_POINTER, 414
 MSK_RES_ERR_NULL_TASK, 414
 MSK_RES_ERR_NUM_ARGUMENTS, 417
 MSK_RES_ERR_NUMCONLIM, 418
 MSK_RES_ERR_NUMVARLIM, 418
 MSK_RES_ERR_OBJ_Q_NOT_NSD, 420
 MSK_RES_ERR_OBJ_Q_NOT_PSD, 420
 MSK_RES_ERR_OBJECTIVE_RANGE, 419
 MSK_RES_ERR_OLDER_DLL, 414
 MSK_RES_ERR_OPF_FORMAT, 416
 MSK_RES_ERR_OPF_NEW_VARIABLE, 417
 MSK_RES_ERR_OPF_PREMATURE_EOF, 417
 MSK_RES_ERR_OPTIMIZER_LICENSE, 413
 MSK_RES_ERR_OVERFLOW, 422
 MSK_RES_ERR_PARAM_INDEX, 417
 MSK_RES_ERR_PARAM_IS_TOO_LARGE, 417
 MSK_RES_ERR_PARAM_IS_TOO_SMALL, 417
 MSK_RES_ERR_PARAM_NAME, 417
 MSK_RES_ERR_PARAM_NAME_DOU, 417
 MSK_RES_ERR_PARAM_NAME_INT, 417
 MSK_RES_ERR_PARAM_NAME_STR, 417
 MSK_RES_ERR_PARAM_TYPE, 417
 MSK_RES_ERR_PARAM_VALUE_STR, 417
 MSK_RES_ERR_PLATFORM_NOT_LICENSED, 413
 MSK_RES_ERR_POSTSOLVE, 422
 MSK_RES_ERR_PRO_ITEM, 419
 MSK_RES_ERR_PROB_LICENSE, 413
 MSK_RES_ERR_PTF_FORMAT, 416
 MSK_RES_ERR_QCON_SUBI_TOO_LARGE, 421
 MSK_RES_ERR_QCON_SUBI_TOO_SMALL, 421
 MSK_RES_ERR_QCON_UPPER_TRIANGLE, 421
 MSK_RES_ERR_QOBJ_UPPER_TRIANGLE, 421
 MSK_RES_ERR_READ_FORMAT, 415
 MSK_RES_ERR_READ_LP_MISSING_END_TAG, 416
 MSK_RES_ERR_READ_LP_NONEXISTING_NAME, 416
 MSK_RES_ERR_REMOVE_CONE_VARIABLE, 420
 MSK_RES_ERR_REPAIR_INVALID_PROBLEM, 422
 MSK_RES_ERR_REPAIR_OPTIMIZATION_FAILED, 422
 MSK_RES_ERR_SEN_BOUND_INVALID_LO, 423
 MSK_RES_ERR_SEN_BOUND_INVALID_UP, 423
 MSK_RES_ERR_SEN_FORMAT, 423
 MSK_RES_ERR_SEN_INDEX_INVALID, 423
 MSK_RES_ERR_SEN_INDEX_RANGE, 423
 MSK_RES_ERR_SEN_INVALID_REGEX, 424
 MSK_RES_ERR_SEN_NUMERICAL, 424
 MSK_RES_ERR_SEN_SOLUTION_STATUS, 424
 MSK_RES_ERR_SEN_UNDEF_NAME, 423
 MSK_RES_ERR_SEN_UNHANDLED_PROBLEM_TYPE, 424
 MSK_RES_ERR_SERVER_CONNECT, 427
 MSK_RES_ERR_SERVER_PROTOCOL, 427
 MSK_RES_ERR_SERVER_STATUS, 427
 MSK_RES_ERR_SERVER_TOKEN, 427
 MSK_RES_ERR_SHAPE_IS_TOO_LARGE, 417
 MSK_RES_ERR_SIZE_LICENSE, 413
 MSK_RES_ERR_SIZE_LICENSE_CON, 413
 MSK_RES_ERR_SIZE_LICENSE_INTVAR, 413
 MSK_RES_ERR_SIZE_LICENSE_NUMCORES, 424
 MSK_RES_ERR_SIZE_LICENSE_VAR, 413
 MSK_RES_ERR_SLICE_SIZE, 422
 MSK_RES_ERR_SOL_FILE_INVALID_NUMBER, 420
 MSK_RES_ERR_SOLITEM, 418
 MSK_RES_ERR_SOLVER_PROBTYPE, 419
 MSK_RES_ERR_SPACE, 414
 MSK_RES_ERR_SPACE_LEAKING, 415
 MSK_RES_ERR_SPACE_NO_INFO, 415
 MSK_RES_ERR_SYM_MAT_DUPLICATE, 424
 MSK_RES_ERR_SYM_MAT_HUGE, 422
 MSK_RES_ERR_SYM_MAT_INVALID, 422
 MSK_RES_ERR_SYM_MAT_INVALID_COL_INDEX, 424
 MSK_RES_ERR_SYM_MAT_INVALID_ROW_INDEX, 424
 MSK_RES_ERR_SYM_MAT_INVALID_VALUE, 424
 MSK_RES_ERR_SYM_MAT_NOT_LOWER_TRINGULAR, 424
 MSK_RES_ERR_TASK_INCOMPATIBLE, 423
 MSK_RES_ERR_TASK_INVALID, 423
 MSK_RES_ERR_TASK_WRITE, 423
 MSK_RES_ERR_THREAD_COND_INIT, 414
 MSK_RES_ERR_THREAD_CREATE, 414
 MSK_RES_ERR_THREAD_MUTEX_INIT, 414
 MSK_RES_ERR_THREAD_MUTEX_LOCK, 414
 MSK_RES_ERR_THREAD_MUTEX_UNLOCK, 414
 MSK_RES_ERR_TOCONIC_CONSTR_NOT_CONIC, 427
 MSK_RES_ERR_TOCONIC_CONSTR_Q_NOT_PSD, 427
 MSK_RES_ERR_TOCONIC_CONSTRAINT_FX, 427
 MSK_RES_ERR_TOCONIC_CONSTRAINT_RA, 427
 MSK_RES_ERR_TOCONIC_OBJECTIVE_NOT_PSD, 427
 MSK_RES_ERR_TOO_SMALL_A_TRUNCATION_VALUE, 421
 MSK_RES_ERR_TOO_SMALL_MAX_NUM_NZ, 418
 MSK_RES_ERR_TOO_SMALL_MAXNUMANZ, 418
 MSK_RES_ERR_UNB_STEP_SIZE, 424
 MSK_RES_ERR_UNDEF_SOLUTION, 419
 MSK_RES_ERR_UNDEFINED_OBJECTIVE_SENSE, 421
 MSK_RES_ERR_UNHANDLED_SOLUTION_STATUS, 425
 MSK_RES_ERR_UNKNOWN, 414
 MSK_RES_ERR_UPPER_BOUND_IS_A_NAN, 421

MSK_RES_ERR_UPPER_TRIANGLE, [425](#)
MSK_RES_ERR_WHICHITEM_NOT_ALLOWED, [418](#)
MSK_RES_ERR_WHICHSOL, [418](#)
MSK_RES_ERR_WRITE_LP_FORMAT, [416](#)
MSK_RES_ERR_WRITE_LP_NON_UNIQUE_NAME, [416](#)
MSK_RES_ERR_WRITE_MPS_INVALID_NAME, [416](#)
MSK_RES_ERR_WRITE_OPF_INVALID_VAR_NAME, [416](#)
MSK_RES_ERR_WRITING_FILE, [416](#)
MSK_RES_ERR_XML_INVALID_PROBLEM_TYPE, [424](#)
MSK_RES_ERR_Y_IS_UNDEFINED, [421](#)

Types

MSKboolean_t, [451](#)
MSKenv_t, [451](#)
MSKint32t, [451](#)
MSKint64t, [452](#)
MSKreal_t, [452](#)
MSKstring_t, [452](#)
MSKtask_t, [451](#)
MSKuserhandle_t, [451](#)
MSKwchart, [452](#)

Index

A

analysis
 infeasibility, 187
attaching
 streams, 15

B

basic
 solution, 73
basis identification, 103, 175
basis type
 sensitivity analysis, 194
BLAS, 110
bound
 constraint, 11, 160, 163
 linear optimization, 11
 variable, 11, 160, 163

C

callback, 83
cardinality constraints, 64, 148
CBF format, 483
ceol
 example, 38
certificate, 74
 dual, 162, 166
 primal, 161, 165
Cholesky factorization, 112, 135
column ordered
 matrix format, 203
compile
 Linux, examples, 8
complementarity, 161, 165
concurrent optimizer, 155
cone
 dual, 164
 dual exponential, 37
 exponential, 37
 power, 33
 quadratic, 28
 rotated quadratic, 28
 semidefinite, 41
conic exponential optimization, 37
conic optimization, 28, 33, 37, 163
 interior-point, 178
 termination criteria, 180
conic problem
 example, 28, 34, 38
conic quadratic optimization, 28

Conic quadratic reformulation, 115
constraint
 bound, 11, 160, 163
 linear optimization, 11
 matrix, 11, 160, 163
 quadratic, 168
correlation matrix, 124
covariance matrix, *see* correlation matrix
cqol
 example, 28
cut, 183

D

defining
 objective, 15
determinism, 119
dual
 certificate, 162, 166
 cone, 164
 feasible, 161
 infeasible, 161, 162, 166
 problem, 161, 164, 167
 solution, 75
 variable, 161, 164
duality
 conic, 164
 linear, 161
 semidefinite, 167
dualizer, 171

E

efficient frontier, 132
eliminator, 171
entropy, 62
 relative, 62
error
 optimization, 73
errors, 77
example
 ceol, 38
 conic problem, 28, 34, 38
 cqol, 28
 lo1, 15
 pow1, 34
 qol, 18
 quadratic objective, 18
examples
 compile Linux, 8
exceptions, 77

- exponential, 61
- exponential cone, 37
- F
- factor model, 135
- feasible
 - dual, 161
 - primal, 160, 173, 179
 - problem, 160
- format, 81
 - CBF, 483
 - json, 501
 - LP, 457
 - MPS, 462
 - OPF, 474
 - PTF, 497
 - sol, 509
 - task, 501
- full
 - vector format, 202
- G
- geometric mean, 61
- geometric programming, 53
- GP, 53
- H
- hot-start, 177
- I
- I/O, 81
- infeasibility, 74, 161, 165
 - analysis, 187
 - linear optimization, 161
 - repair, 187
 - semidefinite, 167
- infeasible
 - dual, 161, 162, 166
 - primal, 160, 161, 165, 173, 180
 - problem, 160, 161, 167
- information item, 83, 84
- installation, 5
 - makefile, 7
 - requirements, 5
 - troubleshooting, 5
 - Visual Studio, 7
- integer
 - optimizer, 182
 - solution, 73
 - variable, 47
- integer feasible
 - solution, 184
- integer optimization, 47, 182
 - cut, 183
 - initial solution, 51
 - objective bound, 183
 - optimality gap, 185
 - parameter, 47
 - relaxation, 183
 - termination criteria, 184
 - tolerance, 184
- integer optimizer
 - logging, 185
- interior-point
 - conic optimization, 178
 - linear optimization, 172
 - logging, 176, 182
 - optimizer, 172, 178
 - solution, 73
 - termination criteria, 174, 180
- J
- json format, 501
- L
- LAPACK, 110
- license, 121
- linear
 - objective, 15
- linear constraint matrix, 11
- linear dependency, 171
- linear optimization, 11, 160
 - bound, 11
 - constraint, 11
 - infeasibility, 161
 - interior-point, 172
 - objective, 11
 - simplex, 177
 - termination criteria, 174, 177
 - variable, 11
- linearity interval, 193
- Linux
 - examples compile, 8
- lo1
 - example, 15
- log-sum-exp, 62, 152
- logarithm, 61
- logging, 80
 - integer optimizer, 185
 - interior-point, 176, 182
 - optimizer, 176, 178, 182
 - simplex, 178
- logistic regression, 152
- LP format, 457
- M
- machine learning
 - logistic regression, 152
- market impact cost, 136
- Markowitz
 - model, 123
- Markowitz model, 124
 - portfolio optimization, 123
- matrix
 - constraint, 11, 160, 163
 - semidefinite, 41

- symmetric, 41
- matrix format
 - column ordered, 203
 - row ordered, 203
 - triplets, 203
- memory management, 119
- MIP, *see* integer optimization
- mixed-integer, *see* integer
- mixed-integer optimization, *see* integer optimization
- model
 - Markowitz, 123
 - portfolio optimization, 123
- modeling
 - design, 8
- monomial, 60
- MPS format, 462
 - free, 473

N

- near-optimal
 - solution, 184
- norm
 - 1-norm, 59
 - 2-norm, 60
 - p-norm, 61
- numerical issues
 - presolve, 171
 - scaling, 171
 - simplex, 177

O

- objective, 160, 163
 - defining, 15
 - linear, 15
 - linear optimization, 11
- objective bound, 183
- OPF format, 474
- optimal
 - solution, 74
- optimality gap, 185
- optimization
 - conic, 163
 - conic quadratic, 163
 - error, 73
 - linear, 11, 160
 - semidefinite, 167
- optimizer
 - concurrent, 155
 - determinism, 119
 - integer, 182
 - interior-point, 172, 178
 - interrupt, 83
 - logging, 176, 178, 182
 - parallel, 70
 - selection, 171, 172
 - simplex, 177

P

- parallel optimization, 70, 155
- parallelization, 119
- parameter, 81
 - integer optimization, 47
 - simplex, 177
- Pareto optimality, 124
- portfolio optimization
 - cardinality constraints, 64, 148
 - efficient frontier, 132
 - factor model, 135
 - market impact cost, 136
 - Markowitz model, 124
 - model, 123
 - Pareto optimality, 124
 - slippage cost, 136
 - transaction cost, 143
- positive semidefinite, 18
- pow1
 - example, 34
- power, 60
- power cone, 33
- power cone optimization, 33
- presolve, 170
 - eliminator, 171
 - linear dependency check, 171
 - numerical issues, 171
- primal
 - certificate, 161, 165
 - feasible, 160, 173, 179
 - infeasible, 160, 161, 165, 173, 180
 - problem, 161, 164, 167
 - solution, 75, 160
- primal-dual
 - problem, 172, 179
 - solution, 161
- problem
 - dual, 161, 164, 167
 - feasible, 160
 - infeasible, 160, 161, 167
 - load, 81
 - primal, 161, 164, 167
 - primal-dual, 172, 179
 - save, 81
 - status, 73
 - unbounded, 162, 166
- PTF format, 497

Q

- qo1
 - example, 18
- quadratic
 - constraint, 168
- quadratic cone, 28
- quadratic objective
 - example, 18
- quadratic optimization, 168
- quality

- solution, 185
- R
- regression
 - logistic, 152
- relaxation, 183
- repair
 - infeasibility, 187
- response code, 77
- rotated quadratic cone, 28
- row ordered
 - matrix format, 203
- S
- scaling, 171
- semicontinuous variable, 63
- semidefinite
 - cone, 41
 - infeasibility, 167
 - matrix, 41
 - variable, 41
- semidefinite optimization, 41, 167
- sensitivity analysis, 192
 - basis type, 194
- shadow price, 193
- simplex
 - linear optimization, 177
 - logging, 178
 - numerical issues, 177
 - optimizer, 177
 - parameter, 177
 - termination criteria, 177
- slippage cost, 136
- softplus, 62
- sol format, 509
- solution
 - basic, 73
 - dual, 75
 - file format, 509
 - integer, 73
 - integer feasible, 184
 - interior-point, 73
 - near-optimal, 184
 - optimal, 74
 - primal, 75, 160
 - primal-dual, 161
 - quality, 185
 - retrieve, 73
 - status, 14, 74
- solving linear system, 107
- sparse
 - vector format, 203
- sparse vector, 203
- status
 - problem, 73
 - solution, 14, 74
- streams
 - attaching, 15

- string
 - UTF8, 122
 - unicode, 122
- symmetric
 - matrix, 41
- T
- task format, 501
- termination, 73
- termination criteria, 83
 - conic optimization, 180
 - integer optimization, 184
 - interior-point, 174, 180
 - linear optimization, 174, 177
 - simplex, 177
 - tolerance, 175, 181, 184
- thread, 119
- time limit, 83
- tolerance
 - integer optimization, 184
 - termination criteria, 175, 181, 184
- transaction cost, 143
- triplets
 - matrix format, 203
- troubleshooting
 - installation, 5

- U
- UTF8
 - string, 122
- unbounded
 - problem, 162, 166
- unicode
 - string, 122
- unicode string, 122
- user callback, *see* callback
- UTF8, 122

- V
- variable, 160, 163
 - bound, 11, 160, 163
 - dual, 161, 164
 - integer, 47
 - linear optimization, 11
 - semicontinuous, 63
 - semidefinite, 41
- vector format
 - full, 202
 - sparse, 203
- Visual Studio
 - installation, 7