

nag_ip_mps_read (h02buc)

1. Purpose

nag_ip_mps_read (h02buc) reads data for a linear, quadratic, or integer programming problem from a file which is in standard or compatible MPSX input format.

2. Specification

```
#include <nag.h>
#include <nagh02.h>

void nag_ip_mps_read(char *mps_file, Boolean minimize, Integer *n,
                    Integer *m, double **a, double **bl, double **bu,
                    Boolean **intvar, double **cvec, double **x,
                    Nag_H02_Opt *h02_options, NagError *fail)
```

3. Description

nag_ip_mps_read reads linear programming (LP), linear terms of quadratic programming (QP), or integer programming (IP) problem data from a file which is prepared in standard or compatible MPSX (unattributed (1971)) input format and then initializes n (the number of variables), m (the number of general linear constraints), the m by n matrix A , and the vectors l , u and c for use with functions which are designed to solve problems of the form

$$\underset{x \in R^n}{\text{minimize}} \quad f(x) \quad \text{subject to} \quad l \leq \begin{Bmatrix} x \\ Ax \end{Bmatrix} \leq u, \quad (1)$$

where $f(x)$ is a linear function (of the form $c^T x$) or a quadratic function (of the form $c^T x + \frac{1}{2} x^T H x$). Note that for quadratic problems, **nag_ip_mps_read** reads only the linear part of the objective; the quadratic part must be supplied separately to the solver. See the documentation for the appropriate solver for further details. (**nag_ip_mps_read** is primarily designed for use with **nag_ip_bb** (h02bbc), but may also be used in conjunction with **nag_opt_lp** (e04mfc), **nag_opt_lin_lsq** (e04ncc) and **nag_opt_qp** (e04nfc)).

Since, in general, the exact size of the problem defined by an MPSX file may not be known in advance, the arrays returned by **nag_ip_mps_read** are all allocated internally.

MPSX Input Format

The MPSX data file may only contain two types of line:

- (1) Indicator lines (specifying the type of data which is to follow).
- (2) Data lines (specifying the actual data).

The input file must not contain any blank lines. Any characters beyond column 80 are ignored. Indicator lines must not contain leading blank characters (in other words they must begin in column 1). The following displays the order in which the indicator lines must appear in the file:

```
NAME          user-supplied name
ROWS
  data line(s)
COLUMNS
  data line(s)
RHS
  data line(s)
RANGES      (optional)
  data line(s)
BOUNDS      (optional)
  data line(s)
ENDATA
```

The ‘user-supplied name’ specifies a name for the problem and must occupy columns 15-22. The name can either be blank or up to a maximum of 8 characters.

A data line follows the same fixed format made up of fields defined below. The contents of the fields may have different significance depending upon the section of data in which they appear.

	Field 1	Field 2	Field 3	Field 4	Field 5	Field 6
Columns	2-3	5-12	15-22	25-36	40-47	50-61
Contents	Code	Name	Name	Value	Name	Value

The names and codes consist of ‘alphanumeric’ characters (i.e., a–z, A–Z, 0–9, +, –, asterisk (*), blank (), colon (:), dollar sign (\$) or full stop (.) only) and the names must not contain leading blank characters. Values may be entered in several equivalent forms. For example, 1.2345678, 1.2345678e+0, 123.45678e–2 and 12345678E–07 all represent the same number. It is safest to include an explicit decimal point. Note that the lower case ‘e’ exponential notation is not standard MPSX, and if compatibility with other MPSX readers is required then the upper case notation should be used. The lower case notation is supported by nag_ip_mps_read since this is the natural notation in a C programming language environment.

It is recommended that numeric values be *right-justified* in the 12-character field, with no trailing blanks. This is to ensure compatibility with other MPSX readers, some of which may, in certain situations, interpret trailing blanks as zeros. This can dramatically affect the interpretation of the value and is relevant if the value contains an exponent, or if it contains neither an exponent nor an explicit decimal point.

Comment lines are allowed in the data file. These must have an asterisk (*) in column 1 and any characters in columns 2–80. In any data line, a dollar sign (\$) as the first character in field 3 or 5 indicates that the information from that point through column 80 consists of comments.

Columns outside the six fields must be blank, except for columns 72–80, whose contents are ignored by the routine. These columns may be used to enter a sequence number. A non-blank character outside the predefined six fields and columns 72–80 is considered to be a major error unless it is part of a comment.

ROWS Data Lines

These lines specify row (constraint) names and their inequality types (i.e., =, ≥ or ≤).

Field 1: defines the constraint type as follows (may be in column 2 or column 3):

N	free row, i.e., no constraint. It may be used to define the objective row.
G	greater than or equal to (i.e., ≥).
L	less than or equal to (i.e., ≤).
E	exactly equal to (i.e., =).

Field 2: defines the row name.

Row type N stands for ‘Not binding’, also known as ‘Free’. It can be used to define the objective row. The objective row is a free row that specifies the vector c in the linear objective term $c^T x$. It is taken to be the first free row, unless some other free row name is specified by the optional parameter **obj_name** (see Section 7.2). Note that c is assumed to be zero if (for example) the line

```
%N%DUMMYROW
```

(where % denotes a blank) appears in the ROWS section of the MPSX data file, and the row name DUMMYROW is omitted from the COLUMNS section.

COLUMNS Data Lines

These lines specify the names to be assigned to the variables (columns) in the general linear constraint matrix A , and define, in terms of column vectors, the actual values of the corresponding matrix elements.

Field 1: blank (ignored).

Field 2: gives the name of the column associated with the elements specified in the following fields.

Field 3: contains the name of a row.

Field 4: used in conjunction with field 3; contains the value of the matrix element.

Field 5: is optional (may be used like field 3).

Field 6: is optional (may be used like field 4).

Note that only the non-zero elements of A and c need to be specified in the COLUMNS section, as any unspecified elements of A and c are assumed to be zero. In addition, any non-zero elements in the j th column of A must be grouped together before those in the $(j + 1)$ -th column, for $j = 1, 2, \dots, n-1$. Non-zero elements within a column may however appear in any order.

RHS Data Lines

This section specifies the right-hand side values of the general linear constraint matrix A (if any). The lines specify the name to be given to the right-hand side (RHS) vector along with the numerical values of the elements of the vector, which may appear in any order. The data lines have exactly the same format as the COLUMNS data lines, except that the column name is replaced by the RHS name. Only the non-zero elements need be specified. Note that this section may be empty, in which case the RHS vector is assumed to be zero.

RANGES Data Lines (optional)

Ranges are used for constraints of the form $l \leq Ax \leq u$, where both l and u are finite. The effect of specifying a range r_j for constraint j depends on the type of the constraint (i.e., **G**, **L** or **E**), the sign of r_j , and the bound associated with the constraint in the RHS section. (Recall that this bound is taken to be zero if the constraint has no entry in the RHS section.) The various possibilities may be summarised as follows.

Row Type	Sign of r_j	Bound from RHS	Resultant l_j	Resultant u_j
G	+ or -	l_j	l_j	$l_j + r_j $
L	+ or -	u_j	$u_j - r_j $	u_j
E	+	l_j	l_j	$l_j + r_j$
E	-	u_j	$u_j - r_j $	u_j

The data lines have exactly the same format as the COLUMNS data lines, except that the column name is replaced by the RANGE name.

BOUNDS Data Lines (optional)

These lines specify limits on the values of the variables (l and u in $l \leq x \leq u$). If the variable is not specified in the bound set then it is automatically assumed to lie between default lower and upper bounds (usually 0 and $+\infty$). (These default bounds may be reset to the values specified by the optional parameters **col.lo.default** and **col.up.default**; see Section 7.2.) Like an RHS column which is given a name, the set of variables in one bound set is also given a name.

Field 1: specifies the type of bound or defines the variable type as follows:

L0	lower bound.
UP	upper bound.
FX	fixed variable.
FR	free variable ($-\infty$ to $+\infty$).
MI	lower bound is $-\infty$.
PL	upper bound is $+\infty$. This is the default variable type.

Field 2: identifies a name for the bound set.

Field 3: identifies the column name of the variable belonging to this set.

Field 4: identifies the value of the bound; this has a numerical value only in association with **L0**, **UP**, **FX** in field 1, otherwise it is blank.

Field 5: is blank and ignored.

Field 6: is blank and ignored.

Note that if RANGES and BOUNDS sections are both present, the RANGES section must appear first.

Integer Programming Problems

In IP problems there are two common integer variable types: (a) 0–1 integer variables (or ‘binary’ variables) which represent ‘on’ or ‘off’ situations and (b) general integer variables which are forced to take an integer value, in a specified range, at the optimal integer solution. Integer variables can be defined in the following compatible and standard MPSX forms.

In the compatible MPSX format, the type of integer variables are defined in field 1 of the BOUNDS section, that is:

Field 1: specifies the type of the integer variable as follows:

BV	0-1 integer variable (bound value is 1.0).
UI	general integer variable (bound value is in field 4).

In the standard MPSX format, the integer variables are treated the same as ‘ordinary’ bounded variables, in the BOUNDS section. Integer markers are, however, introduced in the COLUMNS section to specify the integer variables. The indicator lines for these markers are:

	Field 1	Field 2	Field 3	Field 4	Field 5	Field 6
Columns	2-3	5-12	15-22	25-36	40-47	50-61
Contents		<i>name</i>	’MARKER’		’INTORG’	

to mark the beginning of the integer variables and

	Field 1	Field 2	Field 3	Field 4	Field 5	Field 6
Columns	2-3	5-12	15-22	25-36	40-47	50-61
Contents		<i>name</i>	’MARKER’		’INTEND’	

to mark the end. That is, any variables between these markers are treated as integer variables. The *name* in Field 2 may be any name different from the preceding and following column names, the other entries in the indicator lines must be exactly as described above (including quotation marks). Note that if the INTEND indicator line is not specified then all columns between the INTORG indicator line and the end of the COLUMNS section are assumed to be integer variables. nag_ip_mps_read accepts both standard and/or compatible MPSX format as a means of specifying integer variables.

An example of the compatible MPSX format is given in Section 6 and an example of the standard MPSX format is given in Section 12.

4. Parameters

mps_file

Input: the name of the MPSX data file. If **mps_file** is a null pointer or null string, then the data is assumed to come from **stdin**.

minimize

Input: specifies the direction of the optimization. **minimize** must be set to **TRUE** for minimization and to **FALSE** for maximization. For a maximization problem, *c*, the coefficients of the linear part of the objective function, is negated with respect to its definition in the MPSX file. For maximization problems involving a quadratic objective function, the user must also modify the sign of the quadratic term as appropriate.

n

Output: *n*, the number of variables specified by the data file.

m

Output: *m*, the number of general linear constraints specified by the data file.

a

Output: *A*, the matrix of general linear constraints.

Sufficient memory is allocated internally by nag_ip_mps_read and may be freed by the utility function nag_ip_mps_free (h02bvc).

bl
bu

Output: **bl** and **bu** hold the lower bounds and upper bounds, respectively, for all the variables and constraints, in the following order. The first **n** elements contain the bounds on the variables x and the next **m** elements contain the bounds for the general linear constraints Ax (if any). Note that an ‘infinite’ lower bound is indicated by $\mathbf{bl}[j - 1] = -10^{20}$, an ‘infinite’ upper bound by $\mathbf{bu}[j - 1] = 10^{20}$, and an equality constraint by $\mathbf{bl}[j - 1] = \mathbf{bu}[j - 1]$.

Sufficient memory is allocated internally by `nag_ip_mps_read` and may be freed by the utility function `nag_ip_mps_free` (h02bvc).

intvar

Output: indicates which are the integer variables in the problem. More precisely, $\mathbf{intvar}[j - 1] = \mathbf{TRUE}$ if x_j is an integer variable, and **FALSE** otherwise, for $j = 1, 2, \dots, n$.

Sufficient memory is allocated internally by `nag_ip_mps_read` and may be freed by the utility function `nag_ip_mps_free` (h02bvc).

cvec

Output: c , the coefficients of the linear term of the objective function. The signs of these coefficients are determined by the problem and the direction of the optimization (see **minimize** above).

Sufficient memory is allocated internally by `nag_ip_mps_read` and may be freed by the utility function `nag_ip_mps_free` (h02bvc).

x

Output: an initial estimate of the solution to the problem. More precisely, $\mathbf{x}[j] = \min(\max(0.0, \mathbf{bl}[j]), \mathbf{bu}[j])$, for $j = 0, 1, \dots, n - 1$.

Sufficient memory is allocated internally by `nag_ip_mps_read` and may be freed by the utility function `nag_ip_mps_free` (h02bvc).

options

Input/Output: a pointer to a structure of type `Nag_H02_Opt` whose members are optional parameters for `nag_ip_mps_read`. These structure members offer the means of adjusting the parameter values used when reading in the MPSX file and on output will supply further details of the results. A description of the members of **options** is given below in Section 7.2.

If any of these optional parameters are required then the structure **options** should be declared and initialised by a call to `nag_ip_init` (h02xxc) and supplied as an argument to `nag_ip_mps_read`. However, if the optional parameters are not required the NAG defined null pointer, `H02_DEFAULT`, can be used in the function call.

fail

The NAG error parameter, see the Essential Introduction to the NAG C Library.

Users are recommended to declare and initialise **fail** and set **fail.print** = **TRUE** for this function.

4.1. Description of Printed Output

Results are printed out by default. The level of printed output can be controlled by the user with the structure members **options.list** and **options.output_level** (see Section 7.2). If **list** = **TRUE** then the parameter values to `nag_ip_mps_read` are listed, whereas the printout of results is governed by the value of **output_level**. The default print level of **Nag_MPS_Summary** gives the following information if the MPSX file has been read successfully:

- (a) the number of lines read.
- (b) the number of columns specified by the data. If any of these are specified as integer variables, the number of such variables is also reported.
- (c) the number of rows specified by the data. The objective row is counted amongst these.

In addition, the names of the problem, the objective row, the RHS set, the RANGES set, and the BOUNDS set read are listed. Unless specified otherwise by the optional parameters **prob_name**, **obj_name**, **rhs_name**, **range_name** and/or **bind_name** (see Section 7), these names will correspond to the first problem, objective row, etc., encountered in the MPSX file. Where no set was encountered (RANGES and BOUNDS are optional), a 'blank' is output.

Additionally, when **output_level** = **Nag_MPS_List**, each line of the MPSX file is echoed as it is read. This may be useful as a debugging aid.

If **output_level** = **Nag_NoOutput** then printout will be suppressed; the user can print the information contained in (b) and (c) when nag_ip_mps_read returns to the calling program.

5. Comments

A list of possible error exits and warnings from nag_ip_mps_read is given in Section 8.

6. Example 1

This example reads a compatible format MPSX file (see Section 3 for a description of standard and compatible MPSX formats) which specifies an instance of the so-called diet problem, and solves it as an LP problem. Given the nutritional content of a selection of foods, the cost of each food, the amount available of each food and the consumer's minimum daily energy requirements, the problem is to find the cheapest combination. This gives rise to the following problem:

minimize

$$c^T x \text{ subject to } Ax \geq b, \quad 0 \leq x \leq u,$$

where

$$c = (3 \quad 24 \quad 13 \quad 9 \quad 20 \quad 19)^T, \quad x = (x_1, x_2, x_3, x_4, x_5, x_6)^T \text{ is real}$$

$$A = \begin{pmatrix} 110 & 205 & 160 & 160 & 420 & 260 \\ 4 & 32 & 13 & 8 & 4 & 14 \\ 2 & 12 & 54 & 285 & 22 & 80 \end{pmatrix}, \quad b = \begin{pmatrix} 2000 \\ 55 \\ 800 \end{pmatrix} \text{ and}$$

$$u = (4 \quad 3 \quad 2 \quad 8 \quad 2 \quad 2)^T.$$

The rows of A correspond to energy, protein and calcium and the columns of A correspond to oatmeal, chicken, eggs, milk, pie and bacon respectively.

The following program calls nag_ip_mps_read to read the MPSX file containing the data for this problem and solves it as an LP problem by calling nag_opt_lp (e04mfc). Finally, nag_ip_mps_free (h02bvc) is called to free the memory allocated by nag_ip_mps_read.

This example shows the simple use of nag_ip_mps_read where default values are used for all optional parameters. An example showing the use of optional parameters is given in Section 12. There is one example program file, the main program of which calls both examples. The main program and Example 1 are given below.

6.1. Program Text

```
/* nag_ip_mps_read (h02buc) Example Program.
 *
 * Copyright 1998 Numerical Algorithms Group.
 *
 * Mark 5, 1998.
 *
 * Mark 6 revised, 2000.
 */
#include <nag.h>
#include <stdio.h>
```

```

#include <nag_stdlib.h>
#include <nage04.h>
#include <nagh02.h>

static void ex1(void);
static void ex2(void);

main()
{
  Vprintf("h02buc Example Program Results.\n");
  ex1();
  ex2();
  exit(EXIT_SUCCESS);
}

static void ex1(void)
{
  Integer n, m;
  double *a, *c, *bl, *bu, *x, objf;
  Boolean *intvar;
  static NagError fail;

  Vprintf("\nExample 1: default options used.\n");

  fail.print = TRUE;

  /* Read MPSX data */
  h02buc(0, TRUE, &n, &m, &a, &bl, &bu, &intvar, &c, &x,
  H02_DEFAULT, &fail);

  /* Solve IP problem defined by data as an LP problem */
  e04mfc(n, m, a, n, bl, bu, c, x, &objf, E04_DEFAULT, NAGCOMM_NULL, &fail);

  /* Free memory allocated by h02buc */
  h02bvc(&a, &bl, &bu, &intvar, &c, &x);
}

```

6.2. Program Data

```

* h02buc Example Program Data
*
* Example 1 data.
*
* This is an example of 'compatible format' MPSX data.
*
*
NAME          DIET
ROWS
  G ENERGY
  G PROTEIN
  G CALCIUM
  N COST
COLUMNS
  OATMEAL ENERGY 110.0
  OATMEAL PROTEIN  4.0
  OATMEAL CALCIUM  2.0
  OATMEAL COST     3.0
  CHICKEN ENERGY 205.0
  CHICKEN PROTEIN  32.0
  CHICKEN CALCIUM 12.0
  CHICKEN COST     24.0
  EGGS ENERGY   160.0
  EGGS PROTEIN   13.0
  EGGS CALCIUM   54.0
  EGGS COST      13.0
  MILK ENERGY  160.0
  MILK PROTEIN   8.0
  MILK CALCIUM  285.0
  MILK COST      9.0
  PIE ENERGY   420.0
  PIE PROTEIN    4.0

```

```

    PIE      CALCIUM    22.0
    PIE      COST       20.0
    BACON    ENERGY   260.0
    BACON    PROTEIN   14.0
    BACON    CALCIUM   80.0
    BACON    COST      19.0
RHS
    DEMANDS  ENERGY   2000.0
    DEMANDS  PROTEIN   55.0
    DEMANDS  CALCIUM   800.0
BOUNDS
    UI SERVINGS OATMEAL  4.0
    UI SERVINGS CHICKEN  3.0
    UP SERVINGS EGGS     2.0
    UP SERVINGS MILK     8.0
    UP SERVINGS PIE      2.0
    UI SERVINGS BACON    2.0
ENDATA

```

6.3. Program Results

h02buc Example Program Results.

Example 1: default options used.

Parameters to h02buc

```

prob_name..... (first)
obj_name..... (first)
range_name..... (first)
col_lo_default..... 0.00e+00
ncol_approx..... 100
output_level..... Nag_MPS_Summary
outfile..... stdout
rhs_name..... (first)
bnd_name..... (first)
col_up_default..... 1.00e+20
nrow_approx..... 100

```

The MPS file has successfully been read.

```

Number of lines read: 50
Number of columns: 6 (of which 3 are integers)
Number of rows: 4 (including objective row)

```

MPS Names Selected:

```

Problem DIET
Objective COST      RHS      DEMANDS
RANGES          BOUNDS  SERVINGS

```

MPS data successfully assigned to problem data.

Parameters to e04mfc

```

Linear constraints..... 3      Number of variables..... 6
prob..... Nag_LP      start..... Nag_Cold
ftol..... 1.05e-08    reset_ftol..... 5
fcheck..... 50        crash_tol..... 1.00e-02
inf_bound..... 1.00e+20  inf_step..... 1.00e+20
max_iter..... 50       machine_precision..... 1.11e-16
optim_tol..... 1.72e-13  min_infeas..... FALSE
print_level..... Nag_Soln_Iter
outfile..... stdout

Memory allocation:
state..... Nag
ax..... Nag      lambda..... Nag

```

Results from e04mfc:

Itn	Jdel	Jadd	Step	Ninf	Sinf/Obj	Bnd	Lin	Nart	Nrz	Norm	Gz
0	0	0	0.0e+00	3	2.8550e+03	6	0	0	0	0.00e+00	
1	4 L	4 U	1.8e-02	1	7.2000e+02	6	0	0	0	0.00e+00	
2	5 L	7 L	4.1e-03	0	1.0629e+02	5	1	0	0	0.00e+00	
3	1 L	1 U	1.9e+00	0	9.7333e+01	5	1	0	0	0.00e+00	
4	4 U	5 U	2.9e+00	0	9.2500e+01	5	1	0	0	0.00e+00	

Final solution:

Varbl	State	Value	Lower Bound	Upper Bound	Lagr Mult	Residual
V 1	UL	4.00000e+00	0.0000e+00	4.0000e+00	-3.187e+00	0.000e+00
V 2	LL	0.00000e+00	0.0000e+00	3.0000e+00	1.247e+01	0.000e+00
V 3	LL	0.00000e+00	0.0000e+00	2.0000e+00	4.000e+00	0.000e+00
V 4	FR	4.50000e+00	0.0000e+00	8.0000e+00	0.000e+00	3.500e+00
V 5	UL	2.00000e+00	0.0000e+00	2.0000e+00	-3.625e+00	0.000e+00
V 6	LL	0.00000e+00	0.0000e+00	2.0000e+00	4.375e+00	0.000e+00

LCon	State	Value	Lower Bound	Upper Bound	Lagr Mult	Residual
L 1	LL	2.00000e+03	2.0000e+03	None	5.625e-02	-2.274e-13
L 2	FR	6.00000e+01	5.5000e+01	None	0.000e+00	5.000e+00
L 3	FR	1.33450e+03	8.0000e+02	None	0.000e+00	5.345e+02

Exit after 4 iterations.

Optimal LP solution found.

Final LP objective value = 9.2500000e+01

7. Optional Parameters

A number of optional input and output parameters to `nag_ip_mps_read` are available through the structure argument **options**, type `Nag_H02_Opt`. A parameter may be selected by assigning an appropriate value to the relevant structure member; those parameters not selected will be assigned default values. If no use is to be made of any of the optional parameters the user should use the NAG defined null pointer, `H02_DEFAULT`, in place of **options** when calling `nag_ip_mps_read`; the default settings will then be used for all parameters.

Before assigning values to **options** directly the structure **must** be initialised by a call to the function `nag_ip_init` (`h02xxc`). Values may then be assigned to the structure members in the normal C manner.

Option settings may also be read from a text file using the function `nag_ip_read` (`h02xyc`) in which case initialisation of the **options** structure will be performed automatically if not already done. Any subsequent direct assignment to the **options** structure **must not** be preceded by initialisation.

7.1. Optional Parameter Checklist and Default Values

For easy reference, the following list shows the members of **options** which are valid for `nag_ip_mps_read` together with their default values where relevant.

Boolean list	TRUE
<code>Nag_OutputType</code>	<code>output_level</code>
<code>char outfile[80]</code>	<code>stdout</code>
<code>char prob_name[9]</code>	<code>'\0'</code>
<code>char obj_name[9]</code>	<code>'\0'</code>
<code>char rhs_name[9]</code>	<code>'\0'</code>
<code>char range_name[9]</code>	<code>'\0'</code>
<code>char bnd_name[9]</code>	<code>'\0'</code>
<code>double col_lo_default</code>	<code>0.0</code>
<code>double col_up_default</code>	<code>10²⁰</code>
<code>Integer ncol_approx</code>	<code>100</code>

col_up_default – double Default = 10^{20}

Input: the default upper bound to be used for the variables in the problem when none is specified in the BOUNDS section of the MPSX data file.

Constraint: **options.col_up_default** \geq **options.col_lo_default**.

ncol_approx – Integer Default = 100

nrow_approx – Integer Default = 100

Input: an estimate of the number of columns and rows in the problem. `nag_ip_mps_read` is designed so that the problem size does not have to be known in advance, and allocates memory according to the data contained in the MPSX file. However, for very large problems, an advance estimate of the problem size might allow slightly more efficient memory usage to be achieved.

Constraints:

options.ncol_approx $>$ 0,

options.nrow_approx $>$ 0.

crnames – char ** Default memory **n+m** array of char *

Output: the MPSX names of all the variables and constraints in the problem in the following order. **crnames**[*j*–1] contains the name of the *j*th column, for *j* = 1, 2, ..., **n**. **crnames**[**n**+*i*–1] contains the name of the *i*th row, for *i* = 1, 2, ..., **m**. Each name is 8 characters long, and includes any trailing blank characters which appear in the appropriate name field of the MPSX file.

Sufficient memory to hold the names is allocated internally by `nag_ip_mps_read`. The memory freeing function `nag_ip_free` (`h02xzc`) should be used to free this memory. Users should **not** use the standard C function `free()` for this purpose.

If, on return from `nag_ip_mps_read`, `nag_ip_bb` (`h02bbc`) is called with **options** as an argument, and the memory pointed to by **crnames** has not been freed, `nag_ip_bb` (`h02bbc`) will use the row and column names stored in **crnames** in its solution output.

n_ivar – Integer

Output: the number of integer variables specified by the data file.

8. Error Indications and Warnings

NE_NULL_ARGUMENT

Argument **n** is a null pointer. It should contain the address of a variable of type Integer.
 Argument **m** is a null pointer. It should contain the address of a variable of type Integer.
 Argument **a** is a null pointer. It should contain the address of a variable of type double *.
 Argument **bl** is a null pointer. It should contain the address of a variable of type double *.
 Argument **bu** is a null pointer. It should contain the address of a variable of type double *.
 Argument **intvar** is a null pointer. It should contain the address of a variable of type Boolean *.
 Argument **cvec** is a null pointer. It should contain the address of a variable of type double *.
 Argument **x** is a null pointer. It should contain the address of a variable of type double *.

NE_OPT_NOT_INIT

Options structure not initialized.

NE_BAD_PARAM

On entry, parameter **options.prob_name** had an illegal value.
 On entry, parameter **options.obj_name** had an illegal value.
 On entry, parameter **options.rhs_name** had an illegal value.
 On entry, parameter **options.range_name** had an illegal value.
 On entry, parameter **options.bnd_name** had an illegal value.
 On entry, parameter **options.output_level** had an illegal value.

NE_2_REAL_EE_OPT_ARG_CONS

On entry, **options.col_lo_default** = $\langle value \rangle$ while **options.col_up_default** = $\langle value \rangle$.
 Constraint: **options.col_lo_default** \leq **options.col_up_default**.

NE_INT_OPT_ARG_LT

On entry, **options.nrow_approx** = $\langle value \rangle$. Constraint: **nrow_approx** ≥ 1

On entry, **options.ncol_approx** = $\langle value \rangle$. Constraint: **ncol_approx** ≥ 1

NE_NAMES_NOT_NAG_MEM

options.cnames is not null but does not point to memory allocated by an earlier call to this function. This function does not accept user-allocated memory assigned to **cnames**.

NE_MPS_PROB_NOT_FOUND

The specified problem has not been found in the MPSX file.

NE_MPS_ILLEGAL_DATA_LINE

An illegal data line has been read from the MPSX file. This is neither a comment nor a legal data line.

Error at MPSX line $\langle value \rangle$: $\langle string \rangle$.

NE_MPS_ILLEGAL_NAME

An illegal row or column name has been detected. Names must contain only alphanumeric characters with no leading blanks.

Error at MPSX line $\langle value \rangle$: $\langle string \rangle$.

NE_MPS_ILLEGAL_NUMBER

Number expected but value could not be read. Check numerical fields.

Error at MPSX line $\langle value \rangle$: $\langle string \rangle$.

NE_MPS_ILLEGAL_SETNAME

An illegal name has been detected in field 2 of the RHS, RANGES or BOUNDS section. Names must contain only alphanumeric characters with no leading blanks.

Error at MPSX line $\langle value \rangle$: $\langle string \rangle$.

NE_MPS_INVALID_BND_TYPE

An invalid bound type appears in the BOUNDS section. Expect: LO, UP, FX, FR, MI, PL, BV or UI.

Error at MPSX line $\langle value \rangle$: $\langle string \rangle$.

NE_MPS_INVALID_BND_VAL

Invalid numeric field in bound data. Value expected for types: LO, UP, FX, UI. Blank field expected for types: FR, MI, PL, BV.

Error at MPSX line $\langle value \rangle$: $\langle string \rangle$.

NE_MPS_INVALID_INDICATOR

Unknown, unexpected or invalid indicator line read. Expect: NAME, ROWS, COLUMNS, RHS, RANGES, BOUNDS or ENDATA, starting in column 1 of file, and in that order. RANGES and/or BOUNDS may be omitted.

Error at MPSX line $\langle value \rangle$: $\langle string \rangle$.

NE_MPS_INVALID_INTORG_INTEND

An INTORG or INTEND marker is not correctly specified or is unexpected (e.g., INTEND has no matching INTORG).

Error at MPSX line $\langle value \rangle$: $\langle string \rangle$.

NE_MPS_INVALID_ROW_TYPE

An invalid row type appears in the ROWS section. Expect: N, G, L or E.

Error at MPSX line $\langle value \rangle$: $\langle string \rangle$.

NE_MPS_NO_COLS

There were no columns specified in the COLUMNS section.

Last MPSX line read ($\langle value \rangle$): $\langle string \rangle$.

NE_MPS_NO_NEWLINE

New line expected but not found.

Last MPSX line read ($\langle value \rangle$): $\langle string \rangle$.

NE_MPS_NO_OBJ

The objective row was not found. There must be at least one row of type N in the ROWS section and, if an objective name was specified, there must be a type N row with this name.

Last MPSX line read ($\langle value \rangle$): $\langle string \rangle$.

NE_MPS_NO_ROWS

There were no rows specified in the ROWS section.
Last MPSX line read (*value*): *string*.

NE_MPS_REPEAT_ROW

A row has been specified more than once.
Error at MPSX line *value*: *string*.

NE_MPS_RHS_RANGE_BND_NOT_FOUND

The name of the RHS, RANGES or BOUNDS set to be used was not found in the file.

NE_MPS_ENDATA_NOT_FOUND

The file does not contain an ENDATA indicator.

NE_MPS_SPLIT_COL

Column data is not contiguous. All entries for a given column must appear together in the COLUMNS section.
Error at MPSX line *value*: *string*.

NE_MPS_UNKNOWN_COLNAME

An unknown column name appears in the BOUNDS section. All the column names must be specified in the COLUMNS section.
Error at MPSX line *value*: *string*.

NE_MPS_UNKNOWN_ROWNAME

An unknown row name appears in the *string* section. All the row names must be specified in the ROWS section.
Error at MPSX line *value*: *string*.

NE_ALLOC_FAIL

Memory allocation failed.

NE_NOT_APPEND_FILE

Cannot open file *string* for appending.

NE_WRITE_ERROR

Error occurred when writing to file *string*.

NE_NOT_CLOSE_FILE

Cannot close file *string*.

NE_NOT_READ_FILE

Cannot open file *string* for reading.

NE_INTERNAL_ERROR

An internal error has occurred in this function. Check the function call and any array sizes. If the call is correct then please consult NAG for assistance.

9. Further Comments

Although `nag_ip_mps_read` is designed primarily for use with `nag_ip_bb` (h02bbc), it can also be used in conjunction with `nag_opt_lp` (e04mfc) (as illustrated by example 1), `nag_opt_lin_lsq` (e04ncc) and `nag_opt_qp` (e04nfc). However, these last three functions do not provide a direct means of using the row and column names which can be read by `nag_ip_mps_read` and stored in the optional parameter **crnames**. By making use of the user-defined printing facilities of the functions, the user can customize the solution printing to print the row and column names (see Section 8.3.1 of the documentation for the appropriate function). Alternatively, the user may call `nag_ip_bb` (h02bbc) to solve the LP or QP problem by specifying all variables to be non-integer via the **intvar** parameter (see Section 4 of the documentation for `nag_ip_bb` (h02bbc)).

10. References

(1971) Program Number 5734 XM4 *MPSX - Mathematical programming system* IBM Trade Corporation, New York.

11. See Also

nag_opt_lp (e04mfc)
 nag_opt_lin_lsqr (e04nce)
 nag_opt_qp (e04nfc)
 nag_ip_mps_free (h02bvc)
 nag_ip_bb (h02bbc)
 nag_ip_init (h02xxc)
 nag_ip_read (h02xyc)
 nag_ip_free (h02xzc)

12. Example 2

This example reads in an MPSX file specifying the same IP problem as in example 1. Here the problem is specified in the standard MPSX format (in example 1 it was presented in the compatible MPSX format; see Section 3 for details of the standard and compatible MPSX formats). It is then solved as an IP problem.

The following program calls nag_ip_read (h02xyc), which initialises the **options** structure and reads optional parameter settings from the data file. The parameter settings suppress all output from nag_ip_mps_read. The program then calls nag_ip_mps_read to read the MPSX data and sets the optional parameter **list** to **TRUE** before calling nag_ip_bb (h02bbc) to solve the IP problem. As the **options** structure is passed as an argument, the row and column names read from the file are stored in **options.cnames** and used in the solution output (see Section 12.3).

Finally, nag_ip_mps_free (h02bvc) is called to free the problem arrays, and nag_ip_free (h02xzc) is called to free the memory in **options**.

12.1. Program Text

```
static void ex2(void)
{
  Integer n, m;
  double *a, *c, *bl, *bu, *x, objf;
  Boolean *intvar;
  static NagError fail;
  Nag_H02_Opt options;

  Vprintf("\nExample 2: use of options structure.\n");

  /* Initialise options structure and read option settings. */
  h02xxc(&options);
  h02xyc("h02buc", "stdin", &options, (Boolean)TRUE, "stdout", NAGERR_DEFAULT);

  /* Read MPSX data */
  fail.print = TRUE;
  h02buc(0, TRUE, &n, &m, &a, &bl, &bu, &intvar, &c, &x,
  &options, &fail);

  /* Solve IP problem defined by data */
  options.list = TRUE;
  h02bbc(n, m, a, n, bl, bu, intvar, c, (double*)0, 0, NULLFN,
  x, &objf, &options, NAGCOMM_NULL, &fail);

  /* Free memory allocated by h02buc */
  h02bvc(&a, &bl, &bu, &intvar, &c, &x);

  /* Free options memory */
  h02xzc(&options, "all", NAGERR_DEFAULT);
}
```

12.2. Program Data

```
* Example 2 data.
*
* This defines the same problem as Example 1 but
* is in the 'standard' MPSX format.
*
* Options setting first:
```

```
BEGIN nag_ip_mps_read

  list = FALSE
  output_level = Nag_NoOutput
  prob_name = DIET2

END

NAME          DIET2
ROWS
G  ENERGY
G  PROTEIN
G  CALCIUM
N  COST
COLUMNS
  OATMEAL  ENERGY  110.0
  OATMEAL  PROTEIN   4.0
  OATMEAL  CALCIUM   2.0
  OATMEAL  COST      3.0
  CHICKEN  ENERGY  205.0
  CHICKEN  PROTEIN   32.0
  CHICKEN  CALCIUM   12.0
  CHICKEN  COST      24.0
  INTEGER  'MARKER'          'INTORG'
  EGGS     ENERGY  160.0
  EGGS     PROTEIN   13.0
  EGGS     CALCIUM   54.0
  EGGS     COST      13.0
  MILK     ENERGY  160.0
  MILK     PROTEIN   8.0
  MILK     CALCIUM  285.0
  MILK     COST      9.0
  PIE      ENERGY  420.0
  PIE      PROTEIN   4.0
  PIE      CALCIUM   22.0
  PIE      COST      20.0
  INTEGER  'MARKER'          'INTEND'
  BACON    ENERGY  260.0
  BACON    PROTEIN   14.0
  BACON    CALCIUM   80.0
  BACON    COST      19.0
RHS
  DEMANDS  ENERGY  2000.0
  DEMANDS  PROTEIN   55.0
  DEMANDS  CALCIUM   800.0
BOUNDS
  UI SERVINGS  OATMEAL  4.0
  UI SERVINGS  CHICKEN  3.0
  UP SERVINGS  EGGS     2.0
  UP SERVINGS  MILK     8.0
  UP SERVINGS  PIE      2.0
  UI SERVINGS  BACON    2.0
```

12.3. Program Results

Example 2: use of options structure.

Optional parameter setting for h02buc.

Option file: stdin

list set to FALSE
output_level set to Nag_NoOutput
prob_name set to DIET2

Parameters to h02bbc

Linear constraints..... 3 Number of variables..... 6
Number of integer variables... 6

```

prob..... Nag_MILP
feas_tol..... 1.05e-08      machine precision..... 1.11e-16
inf_bound..... 1.00e+20      max_iter..... 50
first_soln..... FALSE      max_depth..... 10
max_nodes..... ALL_NODES      int_tol..... 1.00e-05
int_obj_bound..... 1.00e+20      soln_tol..... 1.05e-08
nodsel..... Nag_MinObj_Search      varsel..... Nag_First_Int
branch_dir..... Nag_Branch_Left      crnames..... supplied
print_level..... Nag_Soln_Iter
outfile..... stdout
    
```

Memory allocation:
lower..... Nag
upper..... Nag
state..... Nag
lambda..... Nag

Node No	Parent Node	Obj Value	Varbl Chosen	Value Before	Lower Bound	Upper Bound	Value After	Depth
1		9.250e+01						
2	1	9.385e+01	4	4.50e+00	0.00e+00	4.00e+00	4.00e+00	1
3	1	9.319e+01	4	4.50e+00	5.00e+00	8.00e+00	5.00e+00	1
4	3	9.612e+01	5	1.81e+00	0.00e+00	1.00e+00	1.00e+00	2
5	3	9.482e+01	5	1.81e+00	2.00e+00	2.00e+00	2.00e+00	2
6	2	9.450e+01	6	3.08e-01	0.00e+00	0.00e+00	0.00e+00	2
7	2	9.688e+01	6	3.08e-01	1.00e+00	2.00e+00	1.00e+00	2
8	6	9.737e+01	3	5.00e-01	0.00e+00	0.00e+00	0.00e+00	3
9	6	9.650e+01	3	5.00e-01	1.00e+00	2.00e+00	1.00e+00	3
10	5	9.569e+01	1	3.27e+00	0.00e+00	3.00e+00	3.00e+00	3
11	5	9.700e+01	1	3.27e+00	4.00e+00	4.00e+00	4.00e+00	3

*** Integer Solution ***

12	10	9.619e+01	4	5.19e+00	5.00e+00	5.00e+00	5.00e+00	4
13	10	9.945e+01 CO	4	5.19e+00	6.00e+00	8.00e+00	6.00e+00	4
14	4	9.646e+01	4	7.12e+00	5.00e+00	7.00e+00	7.00e+00	3
15	4	9.733e+01 CO	4	7.12e+00	8.00e+00	8.00e+00	8.00e+00	3
16	12	9.644e+01	6	1.15e-01	0.00e+00	0.00e+00	0.00e+00	5
17	12	1.067e+02 CO	6	1.15e-01	1.00e+00	2.00e+00	1.00e+00	5
18	16	9.751e+01 CO	3	1.88e-01	0.00e+00	0.00e+00	0.00e+00	6
19	16	1.035e+02 CO	3	1.88e-01	1.00e+00	2.00e+00	1.00e+00	6
20	14	9.662e+01	6	7.69e-02	0.00e+00	0.00e+00	0.00e+00	4
21	14	1.005e+02 CO	6	7.69e-02	1.00e+00	2.00e+00	1.00e+00	4
22	9	9.850e+01 CO	4	3.50e+00	0.00e+00	3.00e+00	3.00e+00	4
23	9	9.719e+01 CO	4	3.50e+00	4.00e+00	4.00e+00	4.00e+00	4
24	20	9.734e+01 CO	3	1.25e-01	0.00e+00	0.00e+00	0.00e+00	5

Node No	Parent Node	Obj Value	Varbl Chosen	Value Before	Lower Bound	Upper Bound	Value After	Depth
25	20	1.001e+02 CO	3	1.25e-01	1.00e+00	2.00e+00	1.00e+00	5
26	7	1.052e+02 CO	4	2.88e+00	0.00e+00	2.00e+00	2.00e+00	3
27	7	9.705e+01 CO	4	2.88e+00	3.00e+00	4.00e+00	3.00e+00	3

Final solution:

Varbl	State	Value	Lower Bound	Upper Bound	Lagr Mult	Residual
OATMEAL	EQ	4.00000e+00	4.0000e+00	4.0000e+00	3.000e+00	0.000e+00
CHICKEN	LL	0.00000e+00	0.0000e+00	3.0000e+00	2.400e+01	0.000e+00
EGGS	LL	0.00000e+00	0.0000e+00	2.0000e+00	1.300e+01	0.000e+00
MILK	LL	5.00000e+00	5.0000e+00	8.0000e+00	9.000e+00	0.000e+00
PIE	EQ	2.00000e+00	2.0000e+00	2.0000e+00	2.000e+01	0.000e+00
BACON	LL	0.00000e+00	0.0000e+00	2.0000e+00	1.900e+01	0.000e+00
Constr	State	Value	Lower Bound	Upper Bound	Lagr Mult	Residual
ENERGY	FR	2.08000e+03	2.0000e+03	None	0.000e+00	8.000e+01
PROTEIN	FR	6.40000e+01	5.5000e+01	None	0.000e+00	9.000e+00
CALCIUM	FR	1.47700e+03	8.0000e+02	None	0.000e+00	6.770e+02

Exit from branch and bound tree search after 27 nodes.

Optimal IP solution found.

Final IP objective value = 9.7000000e+01
