

FortMP Callable Library and DLL

Last Update
24 April 2008

OptiRisk
SYSTEMS



INVESTOR IN PEOPLE
BS EN ISO 9000 : 2000



FortMP Callable Library and DLL

Contents

1. Introduction
 - 1.1 The Problems Solved by FortMP
 - 1.2 Control of the Algorithms by SPECS commands
 - 1.3 Library Environment
2. Problem Statement
 - 2.1 LP and MIP problems
 - 2.2 QP and QMIP problems
3. Callable Library – Introduction
 - 3.1 Entries to Initiate Execution and set up Controls
 - 3.2 Entries to Read Data from file to the External Data Interface
 - 3.3 Entries to Solve a Problem and Report the Solution
 - 3.4 Call-backs and the Corresponding Set-up Entries
4. Library start-up Call Specifications
 - 4.1 Start-up Entries for the FORTRAN user
 - 4.2 Start-up Entries for the C-Language user
 - 4.3 Visual Basic Usage
5. The External Data Interface
 - 5.1 Solver Input Scalar Arguments
 - 5.2 Solver Input Array Arguments
 - 5.3 Solver Output Scalar Arguments
 - 5.4 Solver Output Array Arguments
 - 5.5 Dictionary of Row and Column Names
 - 5.6 Model Descriptors
6. Entries to Read Data to the External Data Interface
 - 6.1 Data Read Entries for the FORTRAN user
 - 6.2 Data Read Entries for the C-Language and Visual Basic user
7. Entries to Solve a Problem and Report Solution to the User
 - 7.1 Solver Entries for the FORTRAN user
 - 7.2 Solver Entries for the C-Language and Visual Basic user
8. Call-backs and the Corresponding Set-up Entries
 - 8.1 Technique and Usage of ‘Call-backs’
 - 8.2 Call-back entries for the FORTMP user (DLL)
 - 8.3 Call-back entries for the C user (DLL)
 - 8.4 Call-back entries for the Visual Basic user (DLL)
 - 8.5 Static Library Call-back Subroutines
9. SPECS commands for Control of Solver and Data Input
 - 9.1 Run-time SPECS control file. The ‘SPID’ parameter
 - 9.2 Most Important SPECS commands
 - 9.3 External Interface Save and Restart commands
10. Managing the Log File
 - 10.1 Log Levels
 - 10.2 Control of the Log in FORTRAN
 - 10.3 Control of the Log in C

FortMP Callable Library and DLL

Contents (continued)

11. Introduction

11.1 Name table Read-in

11.2 Extracting the name corresponding to an index.

11.3 Looking up the Index Corresponding to a Name – FORTRAN users

11.4 Looking up the Index Corresponding to a Name – C users

Appendix: Library headers – Compile and Link

FortMP Callable Library and DLL

1. Introduction

The FortMP Library has now been extended specifically to assist the solution of QP and QMIP problems and to be used in the Windows environment, either as a static library or as a DLL. It also has entries that make it convenient for use by C, C++ and Visual Basic systems, in addition to FORTRAN 77 and FORTRAN 90 systems. For Visual Basic the static library cannot be used.

In the FortMP manual (with extensions) there is given a full description of the original FortMP system – principally for users of the stand-alone FortMP program. This document describes the most important library entries, with particular reference to DLL use.

1.1 The Problems Solved by FortMP

FortMP itself is designed to solve problems of the following type:-

- LP: with linear objective, linear constraints and continuous variables
- MIP: with linear objective, linear constraints and mixed variable types – discrete and integer
- QP: with quadratic objective, linear constraints and continuous variables
- QMIP: with quadratic objective, linear constraints and mixed variable types – discrete and integer

The library establishes a structure of scalar variables and arrays, referred to as the '**External Data Interface**' in which the users specify a problem to be solved. The structure is passed as parameter arguments in the calls that solve a problem. Calls are available both to solve a problem stated in the external data interface form and to load a problem from an input data file to this form. Input data files may be in standard MPS (extended) layout or in a free-form equivalent.

1.2 Control of the Algorithms by SPECS Commands

A large number of parameter controls referred to as '**SPECS Commands**', are available, both to tune the solver for difficult problems, and to specify action such as '**Save**' and '**Restart**' enabling a great deal of time-saving in many cases. Initial default values for all these controls are set up when the system is initiated and in the case of library use any control may be reset by a call that passes the appropriate SPECS command prior to calling the solver. The default values should serve for many users however.

In the case of the stand-alone FortMP program SPECS commands are entered via a special control-file named '**FORTMP.SPC**'. This method of changing the controls is also available to the library user, and is more flexible as controls can be changed without re-compiling the system. All SPECS commands are described in the FortMP

FortMP Callable Library and DLL

manual and extensions (in particular file '*Commands.doc*'). In this document there is a description of those commands that are most useful for the general user.

1.3 Library Environment

The original environment for using the library is as a sub-system within a console-type program so that the system can display its progress from time to time to the user. In order to change this action for a Windows environment the user needs to write his own message output system, over-riding the mechanism provided in the library. Since it is not possible to over-ride a DLL subroutine, there is a system of 'Call-backs' to enable this to take place. The user codes certain routines in his own system to be invoked at relevant points during execution of library calls, and specifies their entry-point in the start-up calls made to the DLL prior to the main library-calls.

At the moment the library is 'Single-threaded', which means that multi-processing of separate problems cannot be attempted. A lock is provided (Windows32 version only) to safeguard the system.

2. Problem Statement

Given below are the mathematical statements of the kinds of problem that can be solved with the FortMP library.

2.1 LP and MIP problems

The LP or MIP problem may be stated as follows:

Minimise (or Maximise)

$$k + \mathbf{c}^T \cdot \mathbf{x}$$

subject to the conditions:

$$\begin{aligned} \mathbf{L} &\leq \mathbf{A} \cdot \mathbf{x} \leq \mathbf{U} \\ \mathbf{l} &\leq \mathbf{x} \leq \mathbf{u} \end{aligned}$$

Where

- k is a scalar constant offset to the objective
- \mathbf{c} is an n - vector of the cost coefficients c_1, c_2, \dots, c_n .
- \mathbf{x} is an n - vector of the structural variables x_1, x_2, \dots, x_n .
- \mathbf{L} is an m -vector of lower right hand sides L_1, L_2, \dots, L_m .
- \mathbf{A} is an $m \times n$ matrix of coefficients a_{ij} .
- \mathbf{U} is an m -vector of upper right hand sides U_1, U_2, \dots, U_m .
- \mathbf{l} is an n - vector of lower bounds l_1, l_2, \dots, l_n .
- \mathbf{u} is an n - vector of upper bounds u_1, u_2, \dots, u_n .

and in addition for MIP some or all of the variables x_j may be of one of the following types:

- **Binary**, that is to say restricted to be either zero or one.
- **Integer**, that is to say restricted to integer values
- **Semi-continuous**, that is to say either zero, or continuous in the range 1.0 to an upper bound
- **SOS type 1**, member of set introduced for discrete separable programming
- **SOS type 2**, member of set introduced for continuous separable programming

2.2 QP and QMIP Problems

The QP or QMIP problem may be stated as follows:

$$\text{Minimise} \quad k + \mathbf{c}^T \cdot \mathbf{x} + (1/2) \cdot \mathbf{x}^T \cdot \mathbf{Q} \cdot \mathbf{x}$$

subject to the conditions:

$$\begin{aligned} L &\leq \mathbf{A} \cdot \mathbf{x} \leq U \\ l &\leq \mathbf{x} \leq u \end{aligned}$$

Where

- k is a scalar constant offset to the objective
- \mathbf{c} is an n - vector of the cost coefficients c_1, c_2, \dots, c_n .
- \mathbf{x} is an n - vector of the structural variables x_1, x_2, \dots, x_n .
- \mathbf{Q} is an $n \times n$ symmetric positive semi-definite matrix.
- \mathbf{L} is an m -vector of lower right hand sides L_1, L_2, \dots, L_m .
- \mathbf{A} is an $m \times n$ matrix of coefficients a_{ij} .
- \mathbf{U} is an m -vector of upper right hand sides U_1, U_2, \dots, U_m .
- \mathbf{l} is an n - vector of lower bounds l_1, l_2, \dots, l_n .
- \mathbf{u} is an n - vector of upper bounds u_1, u_2, \dots, u_n .

and in addition for QMIP some or all of the variables x_j may be of one of the following types:

- **Binary**, that is to say restricted to be either zero or one.
- **Integer**, that is to say restricted to integer values

FortMP Callable Library and DLL

3. Callable Library - Introduction

The library possesses entries in the following categories:

- Entries to initiate and set up controls
- Entries to read data from file to the external data interface
- Entries to solve a problem and report the solution to the user
- Entries to set up call-backs into the user's system

There are also many other entries (already described in the FortMP manual) enabling user to manage the solution process in more detail, and to perform many miscellaneous tasks.

The library is designed for users coding in FORTRAN 77, FORTRAN 90, C and C++. In the DLL form the library is also callable from Visual Basic. However many entries – including all those that pass textual arguments – have different forms for the different languages, and the proper form should always be used.

Languages differ as to the treatment of the entry-point name and text-type arguments – the following table summarises these differences for Windows 32 environments:

Language	Library Entry-name treatment in WIN32 environments	Text-argument description
Fortran 77, Fortran 90	Name is capitalised and given standard decoration	Fixed length – the length to be passed in a separate argument that follows after the text-pointer
C	Standard decoration is obtained with the ' <code>__stdcall</code> ' modifier.	Variable length with NULL used as terminator.
C++	Name can be declared as a 'C' entry using 'EXTERN_C' and then given standard decoration with the ' <code>stdcall</code> ' modifier.	Variable length with NULL used as terminator.
Visual Basic (DLL library only)	Name has 'StdCall' attribute, but without decoration unless an 'Alias' is given in its declaration. Any decorated names to be used can be found in the 'EXPORT' table of the DLL.	Variable length wide-character string (type BSTR), always passed with ByVal descriptor, which causes conversion to C-type character string when Visual Basic calls a DLL.

The following table summarises differences for UNIX environments:

Language	Library Entry-name treatment in UNIX environments	Text-argument description
Fortran 77, Fortran 90	Name is translated to lower case with an underbar appended	Fixed length – the length to be passed in a separate argument placed at the end.
C	Name is unchanged	Variable length with NULL used as terminator.
C++	See the proper manual for emulating a 'C' entry.	Variable length with NULL used as terminator.

FortMP Callable Library and DLL

Reconciling the differences between WIN32 and UNIX is the chief purpose of the distributed header file 'fmphdr.h'. One of the three macros:

W32_Standard
UNIX_Standard
WAT_Standard

must be '**#define**'-d, and the other two '**#undef**'-ed. Users receiving 'fmphdr.h' must verify that the correct macro is defined for their environment.

Library entries and parameter-lists always conform to certain conventions making the system more user-friendly:

- All entries are given the modifier '**std_call**', which is defined as '**__stdcall**' when '**W32_Standard**' is defined, or as the empty string when '**UNIX_Standard**' is defined.
- Entry-names are always stated in capitals and, subject to definition of macro '**UNIX_Standard**', become translated by '**#define**' statements to the UNIX form (lower case with underbar appended)
- Entries for C-language that must pass text-type arguments do so without any extra text-length argument, and the null-terminated argument is bridged for calling the corresponding FORTRAN entry.
- Older static library entries that are not in the DLL are for FORTRAN but can be emulated in C. The length arguments of any texts are given alternative placement in the argument-list with the help of macros such as '**LCA_W32**', '**LCA_UNIX**', one of which will be nullified by the prior definition of '**W32_Standard**' or '**UNIX_Standard**'.

These conventions are supported by the distributed header files '**fmphdr.h**', '**libhd2.h**' and '**libhdr.h**'. '**libhdr.h**' is not usually needed as it gives only the older entries, not in the DLL and mostly rendered obsolete by the recent library extensions declared in '**libhd2.h**'.

In the DLL version of the library all entries for C-specific use are exported without any '**__stdcall**' decoration, and this ensures that the same entries can be used by C and by Visual Basic without the need for any alias. Fortran-specific entries are exported with the decoration, but should not be needed either in C or by Visual Basic as all exported entries have C equivalents with the letter C appended to the Fortran name.

For the C and C++ users the distributed header files **Fmphdr.h** and **Libhd2.h** should always be used¹. For Visual Basic users examples are distributed from which any necessary declarations etc may easily be re-copied. C-entries serve also for Visual Basic with one exception mentioned later in sections 3.4 and 8.4. 'Public' declarations in Visual Basic may be re-copied from distributed examples.

¹ Previous entries are covered as before with header file **Libhdr.h**. These are all Fortran-type entries (static library only).

FortMP Callable Library and DLL

It is important to note that the entry and call specifications given here must be precisely adhered to. This is guaranteed in C-language by the inclusion of files ***fmphdr.h*** and ***libhd2.h*** supplied with the distribution. However in Fortran and Visual Basic there is no equivalent and so the user is warned of unpredictable consequences in case of any mis-match. Mistakes can be avoided by adhering to the methods used in tutorial examples.

FortMP Callable Library and DLL

3.1 Entries to initiate execution and set-up controls

Entries for FORTRAN-users are:-

<i>BLDFMP</i>	This entry is used to initiate FortMP execution and set defaults for all controls. No 'Call-backs' are set up and messages are sent by default to the console .
<i>SPECMD</i>	Enters a SPECS command over-riding the defaults set by <i>BLDFMP</i> .
<i>SPECIN</i>	Reads the SPECS command-file (<i>fortmp.spc</i>) and stores it internally to save repeated re-reading when the solver has to be called a large number of times.
<i>BKDINI</i>	Resets all SPECS commands to the defaults set by <i>BLDFMP</i> .
<i>UNBFMP</i>	This entry must be used in a multi-threaded system at the completion of FortMP usage for one thread.

Entries for C-language and Visual Basic users are:-

<i>BLDFMPC</i>	C-equivalent of <i>BLDFMP</i> .
<i>SPECMD</i>	C-equivalent of <i>SPECMD</i> .
<i>SPECINC</i>	C-equivalent of <i>SPECIN</i> .
<i>BKINIC</i>	C-equivalent of <i>BKDINI</i> .
<i>UNBFMPC</i>	C-equivalent of <i>UNBFMP</i> .

FortMP Callable Library and DLL

3.2 Entries to read data from file to the external data interface

Entries for FORTRAN-users are:-

<i>MP2SIZ</i>	Performs the initial stage of input, establishing the dimensions of the input model so that user can allocate space for the external interface.
<i>LP2INP</i>	reads in an LP model from standard input data and converts it to the external interface form - together with a starting basis if specified.
<i>LPDINP</i>	Does the same as <i>LP2INP</i> and, additionally, reads in the dictionary of row and column names together with look-up tables.
<i>QP2INP</i>	reads in a QP model from standard input data and converts it to the external interface form - together with a starting basis if specified.
<i>QPDINP</i>	Does the same as <i>QP2INP</i> and, additionally, reads in the dictionary of row and column names together with look-up tables.

Note that these entries replace former library entries ***MP1SIZ*** and ***MP1INP***, which remain available for users of existing systems.

Entries for C and Visual Basic users are:-

<i>MP2SIZC</i>	Performs the initial stage of input, establishing the dimensions of the input model so that user can allocate space for the external interface.
<i>LP2INPC</i>	reads in an LP model from standard input data and converts it to the external interface form - together with a starting basis if specified.
<i>LPDINPC</i>	Does the same as <i>LP2INPC</i> and, additionally, reads in the dictionary of row and column names together with look-up tables.
<i>QP2INPC</i>	reads in a QP model from standard input data and converts it to the external interface form - together with a starting basis if specified.
<i>QPDINPC</i>	Does the same as <i>QP2INPC</i> and, additionally, reads in the dictionary of row and column names together with look-up tables.

FortMP Callable Library and DLL

3.3 Entries to solve a problem and report the solution to the user

Entries for FORTRAN-users are:-

- SUBLP2*** This entry supplies all problem data for an LP or MIP model, solves the problem and returns the solution to the calling program – FORTRAN-users.
- SUBQP2*** This entry supplies all problem data for a QP or QMIP model, solves the problem and returns the solution to the calling program – FORTRAN-users.

Note that these entries replace former library entries ***SUBLP1*** and ***SUBQP1***, which remain available for users of existing systems.

Entries for C and Visual Basic users are:-

- SUBLP2C*** This entry supplies all problem data for an LP or MIP model, solves the problem and returns the solution to the calling program – C-users.
- SUBQP2C*** This entry supplies all problem data for a QP or QMIP model, solves the problem and returns the solution to the calling program – C-users.

FortMP Callable Library and DLL

3.4 Call-backs and the Corresponding Set-up Entries

A general description of the technique and use of Call-backs is given later in section 8. Entries for FORTRAN-users are:-

BMSGCB	This entry is used to set the MSG (message) call-back.
BPH1CB	This entry is used to set the PH1 (SSX Phase 1) call-back.
BPH2CB	This entry is used to set the PH2 (SSX Phase 2) call-back.
BNODCB	This entry is used to set the NOD (MIP node) call-back.
BIPMCB	This entry is used to set the IPM (IPM iteration) call-back.
BINTCB	This entry is used to set the INT (interrupt) call-back.

Entries for C-users are:-

BMSGCBC	This entry is used to set the MSG (message) call-back.
BPH1CBC	This entry is used to set the PH1 (SSX Phase 1) call-back.
BPH2CBC	This entry is used to set the PH2 (SSX Phase 2) call-back.
BNODCBC	This entry is used to set the NOD (MIP node) call-back.
BIPMCBC	This entry is used to set the IPM (IPM iteration) call-back.
BINTCBC	This entry is used to set the INT (interrupt) call-back.

Separate entry for Visual Basic users:-

BMSGCBV	This entry is used to set the MSG call-back.
----------------	--

Visual Basic users employ the C entries for other call-backs. The exception for MSG call-back occurs because Visual Basic converts messages when it call a DLL but does not convert messages when it itself is called from a DLL. The separate entry is to distinguish the library's message event-handler so that the proper conversion can be performed.

4. Library Start-up Call Specifications

4.1 Start-up Entries for FORTRAN users:

The entries described here are provided for systems programmed in FORTRAN (77 or 90). They initialise the solver, set up default controls and any necessary call-backs to the user program.

BLDFMP

This entry must be used before any other call to FortMP. It initialises FortMP execution and sets defaults for all controls. The 'Call-backs' are cleared so that default action will occur unless changed later (messages are sent to the console).

Synopsis:- **CALL BLDFMP (TCTN)**

Where the parameter is:

Parameter	Type	Description
TCTN	INTEGER	Return code (zero if OK, non-zero if locked or otherwise in error)

SPECMD

This entry passes one SPECS command in order to change the value of some control parameter, over-riding the default set by **BLDFMP**.

Synopsis (FORTRAN):- **CALL SPECMD (command, TCTN)**

Where the parameters are:

Parameter	Type	Description
command	CHARACTER*(*)	Text of a SPECS command
TCTN	INTEGER	Return code (zero if OK)

TCTN should be preset to zero before calling **SPECMD**. No action takes place if **TCTN** is non-zero on entry, and after a series of calls one test is enough to verify that all have succeeded.

FortMP Callable Library and DLL

SPECIN

This entry reads the SPECS command-file (*fortmp.spc*) and stores it internally to save repeated re-reading when the solver has to be called a large number of times

Synopsis:- **CALL SPECIN(TCTN)**

Where the parameter is:

Parameter	Type	Description
TCTN	INTEGER	Return code (zero if OK, non-zero otherwise)

BKDINI

This entry resets the defaults for all SPECS commands to the original – as set up by **BLDFMP**.

Synopsis:- **CALL BKDINI**

With no parameters

UNBFMP

This entry must be used in a multi-threaded system at the completion of FortMP usage for one thread. It releases the lock set by **BLDFMP** allowing **BLDFMP** to be used again for another thread. There is no need to use this entry in a single-threaded system.

Synopsis:- **CALL UNBFMP(TCTN)**

Where the parameter is:

Parameter	Type	Description
TCTN	INTEGER	Return code (zero if OK, non-zero otherwise)

FortMP Callable Library and DLL

4.2 Start-up Entries for C-language users

The entries described here are provided for systems programmed in C (or C++). They initialise the solver, set up default controls and any necessary call-backs to the user program.

BLDFMPC

This entry is used to initiate FortMP execution and set defaults for all controls. No 'Call-backs' are set up and messages are sent by default to the console.

Prototype:- `void std_call BLDFMPC(int *TCTN);`

Where:- **TCTN** Is a return code (zero if OK, non-zero locked or otherwise in error)

SPECMDC

This entry passes one SPECS command in order to change the value of some control parameter, over-riding the default set by **BLDFMPC**.

Prototype:- `void std_call SPECMDC(char *command, int *TCTN);`

Where:-

command is the text of a SPECS command (NULL-terminated)
TCTN is the return-code (zero if OK, non-zero otherwise)

TCTN should be preset to zero before calling **SPECMDC**. No action takes place if **TCTN** is non-zero on entry, and after a series of calls one test is enough to verify that all have succeeded.

SPECINC

This entry reads the SPECS command-file (**fortmp.spc**) and stores it internally to save repeated re-reading when the solver has to be called a large number of times.

Prototype:- `void std_call SPECINC(int *TCTN);`

Where:- **TCTN** Is a return code (zero if OK, non-zero otherwise)

FortMP Callable Library and DLL

BKDINIC

This entry resets the defaults for all SPECS commands to the original – as set up by **BLDFMPC**.

Prototype:- `void std_call BKDINIC(void);`

With no parameters

UNBFMPC

This entry must be used in a multi-threaded system at the completion of FortMP usage for one thread. It releases the lock set by **BLDFMPC** allowing **BLDFMPC** to be used again for another thread. There is no need to use this entry in a single-threaded system.

Prototype:- `void std_call UNBFMPC(int *TCTN);`

Where:- **TCTN** Is a return code (zero if OK, non-zero otherwise)

4.3 Visual Basic Usage

For Visual Basic users the C-language entries are used. Note that all numeric parameters are passed 'ByRef' (the default mode) but text-type parameters must be passed 'ByVal'.

FortMP Callable Library and DLL

5. The External Data Interface

The external data interface and certain other arguments are described here in order to avoid repeating parameter descriptions many different times. Note that solver inputs to *SUBLP2* etc. are actually outputs of the data-read entries *LP2INP* etc.

5.1 Solver Input scalar arguments:-

Variable	Type	Description
<i>MR</i>	INTEGER	number of rows - <i>m</i> .
<i>NC</i>	INTEGER	number of columns - <i>n</i> .
<i>NAIJ</i>	INTEGER	number of non-zero entries in the A -matrix.
<i>NQIJ</i>	INTEGER	number of non-zero entries in the Q -matrix.
<i>NSET</i>	INTEGER	Number of special ordered sets (LP only)
<i>KOFF</i>	REAL*8	constant offset to the objective - <i>k</i> .
<i>PNAME</i>	CHARACTER*8	model name. Only the first 8 characters are used
<i>SPID</i>	CHARACTER*(*)	identity of the BEGIN-line in the SPECS file (see below in section ---).

5.2 Solver Input array arguments:-

Array(Dimension)	Type	Description
<i>AIJ (NAIJ)</i>	REAL*8	Non-zero elements of the A -matrix.
<i>AROW (NAIJ)</i>	INTEGER	Row indices of the A-matrix belonging to the corresponding entries in AIJ .
<i>ACOL (NAIJ)</i>	INTEGER	Column indices of the A-matrix belonging to the corresponding entries in AIJ .
<i>UPB (NC)</i>	REAL*8	Upper bound vector - u . Any value greater than or equal to 10^{31} indicates that no upper bound exists.
<i>LOB (NC)</i>	REAL*8	Lower bound vector - l . Any value less than or equal to -10^{31} indicates that no lower bound exists.
<i>URHS (MR)</i>	REAL*8	Upper RHS vector - U . Any value greater than or equal to 10^{31} indicates that no upper bound exists.
<i>LRHS (MR)</i>	REAL*8	Lower RHS vector - L . Any value less than or equal to -10^{31} indicates that no lower bound exists.
<i>COST (NC)</i>	REAL*8	Cost vector - c .
<i>MITYP (NC)</i>	INTEGER	Variable type code for MIP – see below

FortMP Callable Library and DLL

Solver Input Array Arguments (continued)

Array(Dimension)	Type	Description
<i>QIJ</i> (<i>NQIJ</i>)	REAL*8	Non-zero elements of the <i>Q</i> -matrix.
<i>QROW</i> (<i>NQIJ</i>)	INTEGER	Row indices of the <i>Q</i> -matrix belonging to the corresponding entries in <i>QIJ</i> .
<i>QCOL</i> (<i>NQIJ</i>)	INTEGER	Column indices of the <i>Q</i> -matrix belonging to the corresponding entries in <i>QIJ</i> .
<i>SREF</i> (<i>NSET</i>)	INTEGER	Reference row numbers of each special ordered set.
<i>SFUN</i> (<i>NSET</i>)	INTEGER	Function row numbers of each special ordered set.
<i>SBEG</i> (<i>NSET</i>)	INTEGER	First column number in each special ordered set.
<i>SEND</i> (<i>NSET</i>)	INTEGER	Last column number in each special ordered set.

Arrays *AIJ*, *AROW* and *ACOL* do not need to be organised in any particular sequence so long as the entries correspond to each other.

Coding in array *MITYP* is as follows:

CODE	MEANING
0	Continuous variable
1	Binary Variable
2	Integer Variable
3	Semi-continuous Variable
4	Member of SOS type 1
5	Member of SOS type 2

Codes 3, 4 and 5 may not be used in a QP model.

Arrays *QIJ*, *QROW* and *QCOL* do not need to be organised in any particular sequence so long as the entries correspond to each other. However the full *Q*-matrix should be supplied, and if user states the quadratic objective as an algebraic expression then its coefficients are to be doubled in *Q*. The transposed positions of the upper and lower triangle need not be identical, and may be added and put in as a single entry with indices in either order (FortMP does this operation in any case).

Special Ordered Sets are described in the manual sections 6.2.3 and 6.2.4. When this feature is not used (with no *MITYP* codes 4 or 5), the cell *NSET* should be one (1), rather than zero, as some FORTRAN compilers do not accept zero as a dimension. In this case arguments *SREF*, *SFUN*, *SBEG* and *SEND* are mere dummies.

FortMP Callable Library and DLL

5.3 Solver Output scalar arguments:-

Variable	Type	Description
OBJ	REAL*8	Objective value of the solution.
STSL	INTEGER	Solution status. Standard values are shown below
TCTN	INTEGER	Terminate criterion. A value zero indicates that the execution was successful, subject to the value of STSL.

Standard values for solution status **STSL** are:-

STSL = 0	No solution has been obtained
STSL = 1	The problem is infeasible
STSL = 2	The problem is unbounded
STSL = 3	Continuous optimum solution
STSL = 4	Integer feasible solution
STSL = 5	Integer optimum solution

5.4 Solver Output array arguments:-

In the tables which transfer output values back to the user, both logical and structural variables are represented in this order, plus in addition one extra position for the objective function at the beginning. Table-size is therefore

$$1 + MR + NC$$

where:-

Position **1** refers to the objective

Positions **2** to **(1+MR)** refer to logicals

Positions **(2+MR)** to **(1+MR+NC)** refer to structurals

The tables are:-

Array(Dimension)	Type	Description
SOL (1+MR+NC)	REAL*8	Primal solution values.
DSL (1+MR+NC)	REAL*8	Dual solution values.
BAS (1+MR+NC)	INTEGER	Code value for the basis status of each variable in the final solution. Codes are shown below

Codes for each entry in array **BAS** are:-

CODE	MEANING
0	basic variable
-1	Variable is at its lower bound
+1	Variable is at its upper bound

FortMP Callable Library and DLL

BAS may also be an INPUT argument supplying an advanced starting basis. This technique is often used when the user system calls FortMP several times to solve similar problems. It is also used when a basis has been read from the basis input file.

FortMP Callable Library and DLL

5.5 Dictionary of Row and Column Names

During the input process a dictionary of row and column names is formed together with hash-tables and a binary look-up tree permitting names to be identified rapidly. Library entries such as LP2INP and QP2INP simply allocate space for this before input and release it afterwards. Entries LPDINP and QPDINP (with their C-equivalents) assume that the necessary space is pre-allocated and so allow user to make further use of the dictionary after input.

Scalar parameters for the dictionary are:

Variable	Type	Description
<i>MXN</i>	INTEGER	Maximum number of names – must be at least MR+NC
<i>MXH</i>	INTEGER	Size of the Hash-tables. This must be a power of 2, the normal value used is 2^{12} , ie 4096 .
<i>NNAM</i>	INTEGER	Number of names in the table, this need not concern the user.

Array parameters for the dictionary are:

Array(Dimension)	Type	Description
<i>ROWHSH (MXH)</i>	INTEGER	Hash-table for row-names
<i>COLHSH (MXH)</i>	INTEGER	Hash-table for column-names
<i>NMTREE (2, MXN)</i>	INTEGER	Binary look-up tree
<i>NAMTAB (MXN)</i>	CHARACTER*8	Table of names, row-names first followed by column-names

5.6 Model Descriptors

Certain extra data describing the model are necessary in order to obtain a solution. These descriptors are not passed via arguments to the solver subroutines but are given as SPECS commands (see section 8 below). They are listed in the following table

Description	SPECS Command(s)	Default
Specifies the sense of optimisation	<i>MAXIMIZE²</i> <i>MINIMIZE</i>	Minimise
Specifies wither array BAS supplies a starting basis	<i>SSX START INPUT</i> <i>BASIS</i>	Starting basis not supplied
Specifies if Q is given in full, or only the diagonal and one symmetric half	<i>QMATRIX FULL</i> <i>QMATRIX HALF</i>	Full Q-matrix is given.

² At the moment ***MAXIMIZE*** cannot be used for QP models

FortMP Callable Library and DLL

6. Entries to read data from file to the external data interface

Arguments used in this section are all described above in the External Data Interface

The following entries permit MPS-standard files (with various extensions) to be read in and stored as the external interface. Control of precise file-type, location and file name, together with other options is provided by SPECS commands. These commands are to be given by previous calls to SPECMD or via the SPECS command file *fortmp.spc*, which is read as the first step.

Dimension-sizes are found by the first call (*MP2SIZ* or *MP2SIZC*) and these dimensions may be used to allocate storage or verify that existing storage is enough for the model. They are then passed as input to the second call (*LP2INP* etc) where they are used to check that no overflow occurs before the actual sizes are returned as outputs.

Entries *LPDINP*, *QPDINP* and their C-equivalents load dictionary parameters which are to be given for these entries. Utilities to employ the dictionary are described later.

It is possible to read in a basis as well as the actual model data. In order to do this the appropriate SPECS command is used (i.e. '*SSX START INPUT BASIS*').

6.1 Data Read Entries for the FORTRAN user

MP2SIZ

Performs the initial stage of input, establishing the dimensions of the input model so that user can allocate space for the external interface.

Synopsis:

```
CALL MP2SIZ (MR, NC, NAIJ, NQIJ, NSET, SPID, TCTN)
```

Where the arguments are described as a part of the external interface.

FortMP Callable Library and DLL

LP2INP

Reads in an LP model from standard input data and converts it to the external interface form - together with a starting basis if specified.

Synopsis:

```
CALL LP2INP(MR, NC, NAIJ, NSET, PNAME, SPID,  
*          AIJ, AROW, ACOL, UPB, LOB,  
*          URHS, LRHS, COST, MITYPE,  
*          SREF, SFUN, SBEG, SEND,  
*          KOFF, BAS, TCTN)
```

Note that when the number of special ordered sets is actually zero, the size **NSET** should have the value 1 on input. This is because some FORTRAN systems do not accept zero as a valid dimension-size. Codes in **MITYP** determine if one set is actually present.

LPDINP

Reads in an LP model as in **LP2INP**, retaining the dictionary and look-up tables for subsequent use.

Synopsis:

```
CALL LPDINP(MR, NC, NAIJ, NSET, PNAME, SPID,  
*          AIJ, AROW, ACOL, UPB, LOB,  
*          URHS, LRHS, COST, MITYPE,  
*          SREF, SFUN, SBEG, SEND,  
*          KOFF, BAS, MXN, MXH, NNAM,  
*          ROWSH, COLSH, NMTREE, NAMTAB, TCTN)
```

Note that **NSET** should not be zero as before.

QP2INP

Reads in a QP model from standard input data and converts it to the external interface form - together with a starting basis if specified.

Synopsis:

```
CALL QP2INP(MR, NC, NAIJ, NQIJ, PNAME, SPID,  
*          AIJ, AROW, ACOL, QIJ, QROW, QCOL,  
*          UPB, LOB, URHS, LRHS, COST, MITYPE,  
*          KOFF, BAS, TCTN)
```

FortMP Callable Library and DLL

QPDINP

Reads in a QP model as in *QP2INP*, retaining the dictionary and look-up tables for subsequent use.

Synopsis:

```
CALL QPDINP(MR, NC, NAIJ, NQIJ, PNAME, SPID,  
*          AIJ, AROW, ACOL, QIJ, QROW, QCOL,  
*          UPB, LOB, URHS, LRHS, COST, MITYPE,  
*          KOFF, BAS, MXN, MXH, NNAM,  
*          ROWSH, COLSH, NMTREE, NAMTAB, TCTN)
```

FortMP Callable Library and DLL

6.2 Data Read Entries for the C-language and Visual Basic user

MP2SIZC

Performs the initial stage of input, establishing the dimensions of the input model so that user can allocate space for the external interface.

Prototype:

```
void std_call MP2SIZC(int *MR, int *NC, int *NAIJ,  
                     int *NQIJ, int *NSET,  
                     char *SPID, int *TCTN);
```

LP2INPC

Reads in an LP model from standard input data and converts it to the external interface form - together with a starting basis if specified.

Prototype:

```
void std_call LP2INPC(int *MR, int *NC, int *NAIJ,  
                     int *NSET, char *PNAME, char *SPID,  
                     double *AIJ, int *AROW, int *ACOL,  
                     double *UPB, double *LOB, double *URHS,  
                     double *LRHS, double *COST, int *MITYPE,  
                     int *SREF, int *SFUN, int *SBEG, int *SEND,  
                     double *KOFF, int *BAS, int *TCTN);
```

LPDINPC

Reads in an LP model as in *LP2INPC*, retaining the dictionary and look-up tables for subsequent use.

Synopsis:

```
void std_call LPDINPC(int *MR, int *NC, int *NAIJ,  
                     int *NSET, char *PNAME, char *SPID,  
                     double *AIJ, int *AROW, int *ACOL,  
                     double *UPB, double *LOB, double *URHS,  
                     double *LRHS, double *COST, int *MITYPE,  
                     int *SREF, int *SFUN, int *SBEG, int *SEND,  
                     double *KOFF, int *BAS, int *MXN, int *MXH,  
                     int *NNAM, int *ROWSH, int *COLSH,  
                     int *NMTREE, char *NAMTAB, int *TCTN);
```

Note that *NSET* should not be zero as before. All space for the dictionary parameters must be pre-allocated.

FortMP Callable Library and DLL

QP2INPC

Reads in a QP model from standard input data and converts it to the external interface form - together with a starting basis if specified.

Prototype:

```
void std_call QP2INPC(int *MR, int *NC, int *NAIJ,  
    int *NQIJ, char *PNAME, char *SPID,  
    double *AIJ, int *AROW, int *ACOL,  
    double *QIJ, int *QROW, int *QCOL,  
    double *UPB, double *LOB, double *URHS,  
    double *LRHS, double *COST, int *MITYPE,  
    double *KOFF, int *BAS, int *TCTN);
```

QPDINPC

Reads in a QP model as in *QP2INPC*, retaining the dictionary and look-up tables for subsequent use.

Synopsis:

```
void std_call QPDINPC(int *MR, int *NC, int *NAIJ,  
    int *NQIJ, char *PNAME, char *SPID,  
    double *AIJ, int *AROW, int *ACOL,  
    double *QIJ, int *QROW, int *QCOL,  
    double *UPB, double *LOB, double *URHS,  
    double *LRHS, double *COST, int *MITYPE,  
    double *KOFF, int *BAS, int *MXN, int *MXH,  
    int *NNAM, int *ROWSH, int *COLHSH,  
    int *NMTREE, char *NAMTAB, int *TCTN);
```

All space for the dictionary parameters must be pre-allocated.

7. Entries to solve a problem and report solution to the user

Arguments used in this section are all described above in the External Data Interface. For all these entries, argument **BAS** becomes an input (as well as an output), supplying the starting basis when the SPECS command '**SSX START INPUT BASIS**' has been given.

7.1 Solver entries for the FORTMP user

SUBLP2

This entry supplies all problem data for an LP or MIP model, solves the problem and returns the solution to the calling program.

Synopsis:

```
CALL SUBLP2(MR, NC, NAIJ, NSET, PNAME, SPID,  
*          AIJ, AROW, ACOL, UPB, LOB, URHS, LRHS,  
*          COST, MITYPE, SREF, SFUN, SBEG, SEND,  
*          KOFF, OBJ, SOL, DSL, BAS, STSL, TCTN)
```

Note that when the number of sets is actually zero, the size **NSET** should have the value 1 on input. This is because some FORTRAN systems do not accept zero as a valid dimension-size. Codes in **MITYP** determine if one set is actually present.

SUBQP2

This entry supplies all problem data for a QP or QMIP model, solves the problem and returns the solution to the calling program.

```
CALL SUBQP2(MR, NC, NAIJ, NQIJ, PNAME, SPID,  
*          AIJ, AROW, ACOL, QIJ, QROW, QCOL,  
*          UPB, LOB, URHS, LRHS, COST, MITYPE,  
*          KOFF, OBJ, SOL, DSL, BAS, STSL, TCTN)
```

FortMP Callable Library and DLL

7.2 Solver entries for the C-language and Visual Basic user

SUBLP2C

This entry supplies all problem data for an LP or MIP model, solves the problem and returns the solution to the calling program.

Prototype:

```
void std_call SUBLP2C(int *MR, int *NC, int *NAIJ,  
    int *NSET, char *PNAME, char *SPID,  
    double *AIJ, int *AROW, int *ACOL,  
    double *UPB, double *LOB, double *URHS,  
    double *LRHS, double *COST, int *MITYPE,  
    int *SREF, int *SFUN, int *SBEG, int *SEND,  
    double *KOFF, double *OBJ, double *SOL,  
    double *DSL, int *BAS, int *STSL, int *TCTN);
```

SUBQP2C

This entry supplies all problem data for a QP or QMIP model, solves the problem and returns the solution to the calling program.

Prototype:

```
void std_call SUBQP2C(int *MR, int *NC, int *NAIJ,  
    int *NQIJ, char *PNAME, char *SPID,  
    double *AIJ, int *AROW, int *ACOL,  
    double *QIJ, int *QROW, int *QCOL,  
    double *UPB, double *LOB, double *URHS,  
    double *LRHS, double *COST, int *MITYPE,  
    double *KOFF, double *OBJ, double *SOL,  
    double *DSL, int *BAS, int *STSL, int *TCTN);
```

8. Call-backs and the Corresponding Set-up Entries

8.1 Technique and Usage of ‘Call-backs’

A ‘Call-back’ is a technique whereby library procedures can invoke subroutines or functions in the user’s system whenever certain events occur during the library-call. Events of particular interest are:-

- Whenever a message is displayed
- At regular intervals during any long-running algorithm

If the static library is used then the user can simply, in his own code, incorporate alternatives to those library subroutines that handle these events. However when the DLL is used this will not work. Instead the user must invoke a special library-call to pass an entry-point in his own code designed to handle such an event, and this will then pre-empt the library’s default action (if any).

Call-backs available are listed in the following table:

Call-back	Event	Default Action	When Pre-empted
MSG	A message is issued for display to the user	Message is written to the console	Console Write is suppressed. User-code receives the message.
PH1	Break-point at each iteration in Phase 1 of the SSX (Primal and Dual) algorithms.	None	User-code receives iteration data
PH2	Break-point at each iteration in Phase 2 of the SSX (Primal and Dual) algorithms.	None	User-code receives iteration data
NOD	Break-point at each node in Branch and Bound (MIP) algorithms.	None	User-code receives node data
IPM	Break-point at each iteration in the IPM algorithms.	None	User-code receives iteration data
INT	Break-point in algorithms not served by any other call-back.	None	No data is passed.

An important task for the user-code to perform is management of the windows currently on display, and the handling of outstanding Windows events to prevent the screen from ‘freezing’. This is the primary reason for the ‘INT’ call-back. When the ‘INT’ call-back is used (WIN32 DLL versions only) every call-back invokes execution of the Windows event-handler so that the screen does not freeze.

Call-backs other than MSG have a return-code that user-code may set non-zero in order to cancel the current run of the solver and obtain the latest solution.

FortMP Callable Library and DLL

When the static library is used call-backs as such do not exist. The corresponding subroutines within the library are dummies that only execute default action, that is only message-display in the MSG call-back. Specifications of these subroutines are given below (section 8.5). The user may simply encode his own versions that will over-ride the library dummies.

8.2 Call-back entries for the FORTMP user (DLL)

BMSGCB, BINTCB, BPH1CB, BPH2CB, BNODCB, BIPMCB

These entries are used to set up FORTRAN call-backs.

Synopses:-	CALL BMSGCB (wrtscb)	Display message call-back
	CALL BPH1CB (ph1cbk)	SSX Phase 1 iteration call-back
	CALL BPH2CB (ph2cbk)	SSX Phase 2 iteration call-back
	CALL BNODCB (nodcbk)	MIP node call-back
	CALL BIPMCB (ipmcbk)	IPM iteration call-back
	CALL BINTCB (intcbk)	Interrupt call-back (Invoking Windows event-handler for all call-backs)

Where each argument is the (type EXTERNAL) entry-point in the user's system.

Arguments passed to the call-back subroutines are as follows. For each call-back they are given in order in the table below:

Call-back	Argument	Type	Description
Wrtscb	msg	CHARACTER*(*)	Display message
Ph1cbk	iter	INTEGER	Iteration number
	obj	REAL*8	Objective value
	sinf	REAL*8	Infeasibility sum
	info	INTEGER	Signal to 'Cancel'
Ph2cbk	iter	INTEGER	Iteration number
	obj	REAL*8	Objective value
	info	INTEGER	Signal to 'Cancel'
Nodcbk	node	INTEGER	Node number
	iter	INTEGER	Iteration number
	nni	INTEGER	Number non-integer
	lproj	REAL*8	LP objective value
	ipobj	REAL*8	IP objective value
	info	INTEGER	Signal to 'Cancel'
Ipmcbk	iter	INTEGER	Iteration number
	pobj	REAL*8	Primal objective value
	dobj	REAL*8	Dual objective value
	Info	INTEGER	Signal to 'Cancel'
Intcbk	Info	INTEGER	Signal to 'Cancel'

FortMP Callable Library and DLL

The 'Cancel' signal *info* must normally be set to zero by any call-back. If set non-zero then the system halts execution and returns with the 'Cancel' error code. During MIP or QMIP there is a solution in being, which is the best integer solution so far or the solution to the continuous relaxation if no integer solution has been found. This solution is returned in the output arguments by the solver when it is cancelled.

The call-back name given above is actually the same as that of the corresponding library subroutine called within the FortMP solver. If the user gives these names to his call-back subroutines (with the same arguments), and if the user employs the static library rather than the DLL, then there is no need to employ call-backs at all since the users version simply over-rides the library version.

8.3 Call-back entries for the C user (DLL)

BMSGCBC, BINTCBC, BPH1CBC, BPH2CBC, BNODCBC, BIPMCBC

C-language call-backs differ from FORTRAN call-backs in that arguments are passed by value rather than by reference. The following entries are used:

Prototypes:-

```
Void std_call BMSGCBC(Xmsgcb cb);
Void std_call BPH1CBC(Xph1cb cb);
Void std_call BPH2CBC(Xph2cb cb);
Void std_call BNODCBC(Xnodcb cb);
Void std_call BIPMCBC(Xipmcb cb);
Void std_call BINTCBC(Xintcb cb);
```

Prototype definitions of user's call-back entries are:

```
typedef int (std_call *Xmsgcb)(char *msg);
typedef int (std_call *Xph1cb)(int iter,
                             double obj, double sinf);
typedef int (std_call *Xph2cb)(int iter,
                             double obj);
typedef int (std_call *Xnodcb)(int node,
                             int iter, int nni,
                             double lproj, double ipobj);
typedef int (std_call *Xipmcb)(int iter,
                             double pobj, double dobj);
typedef int (std_call *Xintcb)(void);
```

Here the return-code supplies the '*info*' of the FORTRAN equivalents (ignored by the MSG call-back). When the return-code is non-zero (MSG excepted) FortMP cancels solver execution and returns the latest solution if one such has been reached. Arguments are named the same way as above (in FORTRAN), and are passed by value except for '**msg*' in the display message call-back which is an ordinary, null-terminated string.

FortMP Callable Library and DLL

BFMPCBC

In this single entry, provided only for C-users (not for Visual Basic), any or all of the call-backs specified by individual FORTRAN calls above may be set at once.

Prototype:-

```
Void std_call BFMPCBC(XMSGCB msgcb, XINTCB intcb,  
                      XPH1CB ph1cb, XPH2CB ph2cb,  
                      XNODCB nodcb, XIPMCB ipmcb);
```

Where if '*NULL*' is specified for any argument then that particular call-back will not be used.

8.4 Call-back entries for the Visual Basic user (DLL)

BMSGCBV

Visual Basic users employ the same DLL entries as C for call-back types INT, PH1, PH2, NOD and IPM. For MSG call-back the following is used.

```
Public Declare Sub BMSGCBV Lib _  
    "[<library path>]FortMP.dll" (ptr1 As Long)
```

The call-back function could (e.g.) have the following synopsis:

```
Function MyMsgcbak(ByVal msg As String) As Long
```

and this would be set up as follows (after invoking FortMP start-up):

```
Call BMSGCBV(AddressOf MyMsgcbak)
```

The function return-code of *MyMsgcbak* is ignored by the DLL.

BINTCBC, BPH1CBC, BPH2CBC, BNODCBC, BIPMCBC

These entries are shared by C-language and Visual Basic. The declarations are:

```
Public Declare Sub BPH1CBC Lib _  
    "[<library path>]FortMP.dll" (ptr1 As Long)  
Public Declare Sub BPH2CBC Lib _  
    "[<library path>]FortMP.dll" (ptr1 As Long)  
Public Declare Sub BNODCBC Lib _  
    "[<library path>]FortMP.dll" (ptr1 As Long)  
Public Declare Sub BIPMCBC Lib _  
    "[<library path>]FortMP.dll" (ptr1 As Long)  
Public Declare Sub BINTCBC Lib _  
    "[<library path>]FortMP.dll" (ptr1 As Long)
```

FortMP Callable Library and DLL

The call-back functions could (e.g.) have the following synopsis:

```
Function MyPh1cbak(ByVal iter As Long, _  
                    ByVal obj As Double, _  
                    ByVal sinf As Double) As Long  
Function MyPh2cbak(ByVal iter As Long, _  
                    ByVal obj As Double) As Long  
Function MyNodcbak(ByVal node As Long, _  
                    ByVal iter As Long, _  
                    ByVal nni As Long, _  
                    ByVal lpobj As Double, _  
                    ByVal ipobj As Double) As Long  
Function MyIpmcbak(ByVal iter As Long, _  
                    ByVal pobj As Double, _  
                    ByVal dobj As Double) As Long  
Function MyIntcbak() As Long
```

And these would be set up as follows (after invoking FortMP start-up):

```
Call BPH1CBC(AddressOf MyPh1cbak)  
Call BPH2CBC(AddressOf MyPh2cbak)  
Call BNODCBC(AddressOf MyNodcbak)  
Call BIPMCBC(AddressOf MyIpmcbak)  
Call BINTCBC(AddressOf MyIntcbak)
```

Non-zero returns from these call-back functions cause a solver run to halt with the latest solution returned to the Visual Basic calling function.

FortMP Callable Library and DLL

8.5 Static Library Call-back Subroutines

When the static library is used call-backs are obtained by writing alternatives to the internal call-back routines of the static library. They are coded in FORTRAN but the over-riding user-routine may be coded in FORTRAN or C.

FORTRAN synopses are:

```
SUBROUTINE WRTSCB(msg)  
SUBROUTINE PH1CBK(iter, obj, sinf, info)  
SUBROUTINE PH2CBK(iter, obj, info)  
SUBROUTINE NODCBK(node, iter, nni,  
*                lpobj, ipobj, info)  
SUBROUTINE IPMCBK(iter, pobj, dobj, info)  
SUBROUTINE INTCBK(info)
```

With arguments the same as those for corresponding call-backs in the DLL (see table in section 8.2).

For C-language users the corresponding prototype declarations are:

```
void std_call WRTSCB(char *msg, int mlen);  
void std_call PH1CBK(int *iter, double *obj,  
                    double *sinf, int *info);  
void std_call PH2CBK(int *iter, double *obj,  
                    int *info);  
void std_call NODCBK(int *node, int *iter,  
                    int *nni, double *lpobj,  
                    double *ipobj, int *info);  
void std_call IPMCBK(int *iter, double *pobj,  
                    double *dobj, int *info);  
void std_call INTCBK(int *info);
```

with the same arguments as before.

FortMP Callable Library and DLL

9. SPECS Commands for Control of Solver and Data Input

The text-file '*fortmp.spc*' supplies run-time controls – that is SPECS commands – to the FortMP solver and data input subroutines. This section gives an introduction to SPECS command syntax and describes the most important and frequently used commands. Additional commands are described in the FortMP manual and extensions.

SPECS commands may be introduced by using the entries *SPECMD* and *SPECMDC*, which are described above in section 4. Commands introduced in this way are superseded by commands entered at run-time on the file *fortmp.spc*. There is no difference in syntax, and any command may be entered by either method.

In all the command syntax used below the symbols '<', '>' are used to bracket an item supplied by the user. Character '/' is used to specify alternatives. Optional items are enclosed by square brackets '[' and ']'

9.1 Run-time SPECS control file. The 'SPID' parameter

File '*fortmp.spc*' is split into sections permitting different commands to apply in multiple calls to the solver and data entry routines. A section is delimited by the following two lines:

```
BEGIN [(sectid)]  
END
```

where 'sectid' is a string of up to 8 characters, blank if omitted, that gives an identifier to each section. '**END**' is not necessarily the end of the file but merely terminates the section, and any lines between '**END**' and the next '**BEGIN**' (or end of file) are ignored.

In order to select a particular section the parameter SPID is employed when making a call to any solver or data entry routine. The following special values for 'SPID' should be noted:

SPID	Meaning
Blank	Select the first section. If the first section is 'ALL' then select the first and second sections.
'NOSPECS'	Bypass SPECS-command input altogether. All controls remain at previous settings.
'DEFAULT'	Has a special meaning and should never be used for SPID
'ALL'	Has a special meaning and should never be used for SPID

FortMP Callable Library and DLL

9.2 Most Important SPECS commands

Command	Default
BEGIN [(<i><sectid></i>)]	Section ID is blank
END	

The above commands begin and end each separate *section* of the SPECS file (fortmp.spc). Normally only one section is needed and section identity (with surrounding parentheses) is omitted. Multiple sections are used for systems in which FortMP is called as a sub-system to solve many different models.

Command	Default string
MODEL NAME (<i><modname></i>)	<i>Model</i>

This command supplies a default for the names of all input, output and local scratch files to be used in the run. An extension is added according to file type.

Command	Default String
INPUT FILE NAME (<i><filename></i>)	<i><modname>.MPS</i>
BASIS FILE NAME (<i><filename></i>)	<i><modname>.BAS</i>
OUTPUT FILE NAME (<i><filename></i>)	<i><modname>.RES</i>
LOG FILE NAME (<i><filename></i>)	<i><modname>.LOG</i>

These commands assign a special name to the indicated file, where *<modname>* represents the model name. In the case of '**BASIS**' the filename applies to input only and not to output.

Command	Default String
DIRECTORY (<i><dirname></i>)	Empty
PATH NAME (<i><dirname></i>)	

These two alternative commands supply a prefix added to all filenames other than those introduced with an explicit 'FILE NAME' command.

Command	Default Action
MINIMIZE	This is the default
MAXIMIZE	

These commands specify the sense of optimisation.

FortMP Callable Library and DLL

Command	Default Action
QMATRIX <FULL/HALF>	Full Q-matrix

This command specifies whether the Q-matrix is supplied in full, or has only the diagonal entries and one half – the other half being deduced by symmetry. It applies only to library calls **SUBQP2** and **SUBQP2C**, and will not be needed if the model is read previously with **INPQP2**. **INPQP2** sets the command from the header of the quadratic objective section (**QMATRIX** implies FULL - **QDATA** and **QUADS** imply HALF).

Command	Default Option
SCALE [<ON/OFF>]	ON

This command determines if scaling is to be performed.

Command	Default Option
PRESOLVE [<ON/OFF>]	OFF

This command determines if PRESOLVE is to be executed.

Command	Default Action
ALGORITHM PRIMAL	This is the default
ALGORITHM DUAL	
ALGORITHM IPM	

One of these commands is used to specify the primary solution algorithm for the continuous LP problem. 'PRIMAL' is the default and need not be specified.

Command	Default	Minimum
MAXIMUM SSX ITERATIONS = <intval>	50000	1
MAXIMUM IPM ITERATIONS = <intval>	80	1
MAXIMUM MIP <INTEGER SOLUTIONS /INTSOL> = <intval>	300	1
MAXIMUM MIP NODES = <intval>	50000	1
MAXIMUM MIP TIME = <realval>	50000.0	0.0

These commands specify terminal limits on each major algorithm. When a limit is reached a 'SAVE' is made before exit in order to enable a restart when it is desired to continue the run.

FortMP Callable Library and DLL

Command	Default	Minimum
<i>SSX SAVE FREQUENCY = <intval></i>	<i>10</i> (each 10th re-inversion) or <i>0</i>	<i>0</i>
<i>IPM SAVE FREQUENCY = <intval></i>	<i>10</i> (each 10th iteration) or <i>0</i>	<i>0</i>
<i>MIP SAVE FREQUENCY = <intval></i>	<i>500</i> (each 500th node) or <i>0</i>	<i>0</i>

These commands specify the frequency for making a SAVE in each major algorithm. Zero implies that no regular SAVE is to occur. Default settings are 10, 10, and 500 when FortMP is used as a stand-alone program, and zero (i.e. no SAVE) when using the library.

Command	Default Option
<i>SSX START RESTART [<ON/OFF>]</i>	<i>OFF</i>
<i>IPM RESTART [<ON/OFF>]</i>	<i>OFF</i>
<i>MIP RESTART [<ON/OFF>]</i>	<i>OFF</i>

These commands specify whether a RESTART is to be invoked for the input or for each major algorithm. The INPUT procedure always saves the lengthy 'MPS' stage output to permit rapid restart. For SSX algorithms 'RESTART' is an alternative to other starting basis options.

Command	Default Action
<i>SSX START CRASH</i>	This is the default
<i>SSX START INPUT BASIS</i>	
<i>SSX START UNIT BASIS</i>	

These commands select the mechanism for setting up the starting basis when the main algorithm is PRIMAL or DUAL. '***SSX START RESTART***' can also be used.

In the case of a library call to the solver '***SSX START INPUT BASIS***' invokes use of array ***BAS*** to provide the starting basis.

FortMP Callable Library and DLL

Command	Default option
<i>MIP PREPROCESS</i> [<i><ON/OFF/ROOT ONLY></i>]	<i>OFF</i>

This command controls the use of MIP pre-processing. Option 'ON' invokes execution at the root and at every other node in the Branch and Bound tree.

Command	Default option
<i>MIP PRIORITY UP</i> [<i><ON/OFF></i>]	<i>OFF</i>

This command is an important control on the branching in MIP. Option 'ON' causes variable selection to consider fractional values and node selection to choose direction in the UP sense by preference to the DOWN sense. The feature is very useful for models such as allocation, with a large predominance of simple SOS-type constraints.

Command	Default option
<i>GENERATE CUTS</i> [<i><ON/OFF/ROOT ONLY></i>]	<i>OFF</i>

'ON' activates the cut-generation procedures and the application of strong cuts before and during Branch and Bound (See manual section 6.7.3). With ***ROOT ONLY*** the feature is invoked only for the root node of the branch and bound tree.

FortMP Callable Library and DLL

9.3 External Interface Save and Restart commands

FortMP provides a facility for the user to dump the external data interface, together with its model descriptors, onto a file for separate use. This file can act as an input for restarting (saving the time required for model generation), or it can be submitted for testing purposes to the supplier without compromising original source data. Should there be any difficulty in executing the solver, the dump file enables the precise situation to be duplicated, alternatives can be tried out, and the cause of any eventual bug can be investigated.

The dump-file is named **<modname>.XFC**, where **<modname>** is the model name.

The commands for this are given below.

Command	Default option
EXTERNAL SAVE [<ON/OFF>]	OFF

This command activates the save-dump in library routines **SUBLP2**, **SUBQP2**, **SUBLP2C** and **SUBQP2C**. The file is written immediately after entry and reading the SPECS commands.

Command	Default option
EXTERNAL RESTART [<ON/OFF>]	OFF

This command inputs the model from the save-dump in library routines **INPLP2**, **INQP2**, **INPLP2C** and **INQP2C**.

10. Managing the Log File

During execution of FortMP library subroutines (once the set-up is complete) a number of messages are written onto a log-file in order to permit results to be analysed and any failures to be investigated. Messages displayed or passed to the MSG call-back are a sub-selection of the logged messages. The user may need to manage the log for several reasons:

- To combine messages from the user's system with those of FortMP on the same log file
- To adjust the log 'level' according to the amount of detail required
- To concatenate multiple logs onto the same file, and so avoid the need for repeated 'open' and 'close'.

One problem is that FORTRAN and C have separate mechanisms for referencing file I/O channels. The use of channels in common may be possible in a mixed system, but is not easy to make portable between environments. Accordingly there are separate means to handle log-control for FORTRAN and C users.

10.1 Log Levels

Log priority levels range from zero to 4, the highest being level zero for serious warnings and error diagnostics. Level 1 is the default and provides in addition progress messages allowing user to trace the main events. It is also the default level for display – console write or MSG call-back. Higher levels are for serious investigation, tracing iteration history etc., which will only rarely be of interest to a user.

The different algorithms have their own log-level controls, which can be specified by SPECS commands as follows:

```
SSX LOG LEVEL = <level>  
INVERT LOG LEVEL = <level>  
IPM SSX LOG LEVEL = <level>  
MIP LOG LEVEL = <level>
```

Which are the most important. Others may be found in the manual or the SPECS command document. The highest level to display is specified by:

```
LOG DISPLAY LEVEL = <level>  
LOG DISPLAY
```

Where the latter form displays all logged messages.

FortMP Callable Library and DLL

10.2 Control of the Log in FORTRAN

FortMP uses I/O channel numbers (or 'units') 16 to 29 and any channel outside this range may be selected. To reserve a channel the following command is issued:

```
LOG CHANNEL = <channel>
```

This command prevents FortMP from opening or closing its usual log-channel and directs all log-messages onto the channel that is given. An example of its use as part of the set-up would be:

```
CALL SPECMD('LOG CHANNEL = 15')
```

With this command the responsibility for opening and closing the log channel is entirely with the user. User's own messages may be written to the channel and logs from multiple runs of FortMP will concatenate there if it is not rewound or closed between runs.

10.2 Control of the Log in C

When coding in C-language the above mechanism should also be used to prevent FortMP from opening or closing a FORTRAN channel for the log. It takes the form (e.g.):

```
SPECMDC("LOG CHANNEL = 15");
```

In addition to this the user must open and close a log-file in his own system, and must inform FortMP of its file-pointer as part of the set-up. The prototype for this is as follows:

```
Void std_call FILCLOG(FILE *lfile);
```

Where the argument is a file-pointer that user sets up when opening the log file. Executing this call switches the log-writing from the default FORTRAN method to a C-written subroutine writing onto that file.

11. Dictionary of Row and Column Names

11.1 Name table Read-in

Entries *LPFINP*, *LPDINPC*, *QPDINP* and *QPDINPC* accept as arguments the dictionary-pointers for row and column names (pre-declared) and return the table of names in character-array *NAMTAB*. It contains (*MR+NC*) entries, each of exactly 8 characters without any null-terminators between (this is according to Fortran usage). Row names appear first in *NAMTAB*, followed by Column names. The objective row is always the first entry – if in fact the objective is not the first row of the MPS data file, then it is moved back and earlier rows are shifted up one place.

Together with *NAMTAB*, other arrays *ROWHSH*, *COLHSH* and *NMTREE* are also filled by the input routines. Their use is to permit a fast look-up for the user to obtain the row-index or column-index corresponding to any name. Function-entries in the library for this purpose are *FMP_FNDRNM*, *FMP_FNDRNMC*, *FMP_FNDRNM*, and *FMP_FNDRNMC*.

11.2 Extracting the name corresponding to an index.

One use of this would be to associate names with solution and other values for output. There are examples coded in the tutorial files that accompany the distribution. Remember that solution tables *SOL*, *DSL* and *BAS* have one extra position at the start (for the objective) so that positions 2 to (*1+MR+NC*) correspond to positions 1 to (*MR+NC*) of *NAMTAB*.

11.3 Looking up the Index Corresponding to a Name – FORTRAN users

FMP FNDRNM

This is an integer function returning row-index from 1 to *MR*, or returning zero if the name cannot be found. Synopsis:

```
IROW = FMP_FNDRNM(NAME, MXN, MXH, NNAM,  
* ROWHSH, COLHSH, NMTREE, NAMTAB)
```

FMP FNDCNM

This is an integer function returning column-index from 1 to *NC*, or returning zero if the name cannot be found. Synopsis:

```
JCOL = FMP_FNDCNM(NAME, MR, MXN, MXH, NNAM,  
* ROWHSH, COLHSH, NMTREE, NAMTAB)
```

FortMP Callable Library and DLL

Note that this function has an additional argument **MR** not used for row-index look-up.

11.4 Looking up the Index Corresponding to a Name – C users

FMP FNDRNMC

This is an integer function returning row-index from 1 to **MR**, or returning zero if the name cannot be found. Prototype is:

```
int std_call FMP_FNDRNMC(int *NAME, int *MXN,  
                          int *MXH, int *NNAM,  
                          int *ROWHSH, int *COLHSH,  
                          int *NMTREE, char *NAMTAB)
```

FMP FNDCNMC

This is an integer function returning column-index from 1 to **NC**, or returning zero if the name cannot be found. Prototype is:

```
int std_call FMP_FNDCNMC(int *NAME, int *MR,  
                          int *MXN, int *MXH, int *NNAM,  
                          int *ROWHSH, int *COLHSH,  
                          int *NMTREE, char *NAMTAB)
```

Note that this function has an additional argument **MR** not used for row-index look-up.

FortMP Callable Library and DLL

APPENDIX: Library headers – Compile and Link

The following header files are provided with the distribution to assist C-language users:

Fmphdr.h
Libhdr.h
Libhd2.h

‘Fmphdr.h’ is designed to establish portability between PC_WIN32 and UNIX environments. It should be modified for use by changing the system-define macros at the front. For PC-WIN32 use:

```
#define W32_Standard  
#undef UNX_Standard  
#undef WAT_Standard
```

(WATCOM standard is no longer supported). These macros establish a portable naming standard for the entries, and a portable mechanism for handling the string-lengths of any text-type arguments to FORTRAN entries.

‘***Libhdr.h***’ is for all the FORTRAN static-library entries formerly used and mostly now obsolete and replaced by entries in ‘***Libhd2.h***’.

‘***Libhd2.h***’ provides all of the C-language entries in this manual, plus procedure-pointer ‘***typedef***’ declarations for the call-backs. Because of the complexity involved when using call-backs it provides a very important check, and should be used although the prototypes have also been given here.

A user’s system can be linked either with the static library or in WIN32 environments with the DLL bridging library. Library files are:

<i>Fmp32st.lib</i>	WIN32 static library
<i>libfmp32st.a</i>	SUN-UNIX static library
<i>Fortmp.dll</i>	WIN32 DLL
<i>Fortmp.lib</i>	WIN32 bridging-library for the DLL

Note that entries which set up DLL call-backs (***BFMPCBC*** etc) are only provided in the DLL, so that the simpler, over-riding method must be used with the static library (see 8.5 above).

FortMP Callable Library and DLL

It is also possible in C-language to use the WIN32 DLL without linking the bridging library by using the system feature '*LoadLibrary*'. This gives more autonomy and speeds the linking process. In order for this to succeed the entries to the DLL are not declared as prototypes, but as global procedure-pointers that the user invokes with indirect calls. This type of declaration is provided in the header '*Libhd2.h*' when the user pre-defines macro '*SOLVER_DLL*', replacing the conventional prototypes. There is not the least difference between these indirect calls via procedure-pointers and direct calls to the bridging library when '*SOLVER_DLL*' is not defined. User first executes '*LoadLibrary*', and then '*GetProcAddress*' for each entry in use. A simple example of the coding would be:

```
#ifdef SOLVER_DLL
    HINSTANCE hLIB=NULL;
    hLIB = LoadLibrary("fortmp.dll");
    if (!hLIB) exit(-1);          /* Failed */
    (FARPROC)BLDFMPC = GetProcAddress(hLIB, "BLDFMPC");
    (FARPROC)SUBLP2C = GetProcAddress(hLIB, "SUBLP2C");
    if (!BLDFMPC || !SUBLP2C)      /* Failed */
    {
        FreeLibrary(hLIB);
        exit(-1);
    }
#endif
```

and there is a more advanced example given in the tutorial distributed.