Xpress-MP

mmxslp

(Mosel-SLP)

Release version 1.70
iXSLPcommand .................................................. 43
rXSLPcommand .................................................. 44
setdblcontrol .................................................... 45
sethidden .......................................................... 46
setintcontrol ..................................................... 47
setinitval .......................................................... 48
settype ............................................................. 49
SLPcalluserfunc .................................................. 50
SLPDATA ............................................................ 52
SLPexportprob .................................................... 53
SLPgetfuncinfo ................................................... 54
SLPgetindex ........................................................ 55
SLPgetname ........................................................ 56
SLPglobal ........................................................... 57
SLPloadprob ....................................................... 58
SLPlogfile ........................................................... 59
SLPmaximize, SLPminimize ....................................... 60
SLPprintmemory ................................................... 61
SLPscale ............................................................. 62
SLPsetcallback .................................................... 63
SLPsetprobstat ..................................................... 65
SLPsetuserfuncaddress ............................................ 66
SLPsetuserfuncinfo ............................................... 67
SLptime .............................................................. 68
tolsetstatus ........................................................ 69
XSLPcommand ...................................................... 71
XVitem ............................................................... 72

5 The SLPDATA Procedure ......................................... 73
  SLPDATA("DR") .................................................... 74
  SLPDATA("EC") .................................................... 75
  SLPDATA("IV") ..................................................... 76
  SLPDATA("SB") ....................................................... 77
  SLPDATA("TOLSET") ............................................... 78
  SLPDATA("UF") .................................................... 79
  SLPDATA("XV") ..................................................... 81

6 Functions and Callbacks ....................................... 82
  User function (simple form) ..................................... 84
  User function (general form) ..................................... 85
  Func (General SLP function) .................................... 88
  Callbacks .................................................................. 90

7 Error Messages .................................................. 92

Index ..................................................................... 94
Chapter 1
Overview

This guide is intended to be used in conjunction with the Xpress-Mosel Reference Manual. Only those items which are specific to Xpress-SLP and related systems will be included in this guide.

Xpress-SLP is a mathematical programming system for modeling and solving large-scale non-linear problems by using Successive Linear Programming. The extensions within Mosel allow the modeling of any type of non-linear expression and constraint, as well as all the facilities already provided for linear and integer modeling and solution.

Xpress-SLP is provided in Mosel as two Dynamic Shared Objects (DSO):

- \texttt{mmxs1p} is the DSO containing all of the SLP functionality
- \texttt{mmx.cmd} provides a mechanism for using the facilities of the XSLP console-based interface to the Xpress Optimizer and Xpress-SLP.

Chapter 2
Special Object Types

The following special types are provided for Xpress-SLP modeling:

- **gexp**: a general expression
- **genctr**: a general constraint
- **xvitem**: an element of an Xpress-SLP XV

### 2.1 General Expressions

A *general expression* is an expression which involves decision variables (type `mpvar`), not necessarily in a linear way. `mpvar` objects are recognized as such and are treated differently from ordinary real numbers. General expressions are made up of the following:

- The arithmetic operators `+`, `−`, `∗`, `/` and `^` (exponentiation)
- Real numbers
- `mpvar` objects
- SLP functions (both internal and user-defined)
- Linear expressions
- Other general expressions

Any linear expression is also a general expression. However, expressions involving `mpvar` objects in a nonlinear way can only be used in general expressions. This includes Mosel functions used on items which are not constant, except for the aggregation functions `sum` and `prod`. Many of the Mosel mathematical functions can be used directly on non-constant expressions; more complex functions, including those defined by the user, must use the special SLP function form `Func`.

**Examples:**

```plaintext
\[ t + u \ast v \div w + x^0.5 + y \div z \]
\[ \text{sum}(i \in 1..10) x(i) \ast y(i) \]
\[ \sin(x \div y) \]
```

where \( t, u, v, w, x, y, z \) are variables

`gexps` can be used in printing statements. For example:

```plaintext
writeln("myGexp = ", myGexp)
```

where `myGexp` is a general expression. After a solution has been found, the value of the expression will be printed where possible. Because of limitations in recalculating functions, this facility is generally not available for `gexps` that contain function references.
2.2 General Constraint Expressions

A general constraint is built up from general expressions. The allowable forms are:

\[
\begin{align*}
\text{GenExp1} & \quad \text{Ctr}_\text{cmp} \quad \text{GenExp2} \\
\text{LinExp1} & \quad \text{Ctr}_\text{cmp} \quad \text{GenExp2} \\
\text{GenExp1} & \quad \text{Ctr}_\text{cmp} \quad \text{LinExp2}
\end{align*}
\]

where GenExp1 and GenExp2 are general expressions; LinExp1 and LinExp2 are linear expressions involving constants or mpvars; and Ctr_cmp is one of the constraint comparators ≥, = and ≤.

Examples:

\[
\begin{align*}
gx & \leq 42 \quad \text{! where gx is an object of type gexp} \\
x^2 + y^2 & \leq 4 \quad \text{! where x and y are objects of type mpvar} \\
x/y & = a/b \quad \text{! where x, y, a, b are objects of type linctr, gexp or mpvar}
\end{align*}
\]

A general constraint expression can be assigned to an identified object of type genctr but can also be stated on its own, as an anonymous constraint. Values (for example, dual or slack values, or coefficients) of named genctr objects can be accessed but no values can be obtained for anonymous constraints.

Examples:

\[
\begin{align*}
gc := gx & \leq 42 \quad \text{! assigns the constraint to the genctr object gc} \\
gx & \leq 42 \quad \text{! has the same effect, but the constraint is anonymous}
\end{align*}
\]

If the left hand side or right hand side of a general constraint is a linear expression, then the modeling of the linear expression will be retained when the matrix is constructed. That is, the coefficients of the variables in the linear expression will appear as constant coefficients in the matrix. Any items which are in a general expression will be aggregated into a single formula.

The following example shows how different implementations of a constraint arise from different ways of formulating it:

\[
\begin{align*}
\text{declarations} \\
& \quad c: \text{array (1..4) of genctr} \\
& \quad x, y: \text{mpvar} \\
\text{end-declarations} \\
& \quad x \leq 10 \\
& \quad y \leq 10 \\
& \quad c(1):= x^2 + y^2 \quad \leq 4 - x \\
& \quad c(2):= x^2 + y^2 + x \quad \leq 4 \\
& \quad c(3):= x^2 + y^2 + x - 4 \quad \leq 0 \\
& \quad c(4):= x^2 + y^2 \quad \leq 4 - x^2 \\
\text{SLPloadprob}(x+y) \\
\text{SLPmaximize} \\
\text{forall (i in 1..4)} \\
& \quad \text{writeln(}"c(\text{"},i,\text{"}),'="\text{getact}(c(\text{\textbf{i}}))," (c(\text{"},i,\text{"}),x)='\text{getcoeff}(c(\text{\textbf{i}}),x)," (c(\text{"},i,\text{"}),)=''\text{getcoeff}(c(\text{\textbf{i}}))," (c(\text{"},i,\text{"}),nonlin)=''\text{getnlcoeff}(c(\text{\textbf{i}}))"
\) \\
\text{Constraints c(1) to c(4) are mathematically identical. However, because of the different ways in which the expression is split into linear and nonlinear parts, the ways in which they are loaded into the problem are different. With an optimal solution of x=0.962666 and y=1.452960, the printed results are as follows:}
\end{align*}
\]

\[
\begin{align*}
c(1)=4 \quad \{c(1),x\}=1 \quad \{c(1),\}=4 \quad \{c(1),\text{nonlin}\}=3.03782
\end{align*}
\]
The explanation is as follows: all constraints (for example $X \leq Y$) are rewritten with a right hand side of zero (for example $X - Y \leq 0$). It is this latter form which is used when returning values through `getsol`, `getcoeff` or `getnlcoeff`. In more detail:

**c(1)** This is loaded as the constraint

$$c(1):= ( x^2 + y^2 ) + ( x ) \leq 4$$

because the original right hand side is a linear expression which can be separated. It has a constant term of -4 (always interpreted as if it was on the left hand side). The first bracketed term on the left hand side is a (nonlinear) $gexp$ and the second is a linear term arising from the rearrangement of the linear right hand of the original formulation. The coefficient of $x$ is therefore 1 because $x$ appears explicitly in a linear term. The activity of the constraint is 4 (the value of the left hand side of the rearrangement). The value of the nonlinear part is the value of $x^2 + y^2$.

**c(2)** This is loaded as the constraint

$$c(2):= ( x^2 + y^2 + x ) \leq 4$$

because the original right hand side is a constant, and the left hand side is a $gexp$. It has a constant term of -4 (always interpreted as if it was on the left hand side). The bracketed term on the left hand side is a (nonlinear) $gexp$. The coefficient of $x$ is therefore 0 because $x$ appears only in a $gexp$. The activity of the constraint is 4 (the value of the left hand side of the rearrangement). The value of the nonlinear part is also 4, within the tolerance of the convergence.

**c(3)** This is loaded as the constraint

$$c(3):= ( x^2 + y^2 + x - 4) \leq 0$$

because the original left hand side is a $gexp$. It has a constant term of 0. The bracketed term on the left hand side is a (nonlinear) $gexp$. The coefficient of $x$ is therefore 0 because $x$ appears only in a $gexp$. The activity of the constraint is 0 (the value of the left hand side of the rearrangement). The value of the nonlinear part is also 0, within the tolerance of the convergence.

**c(4)** This is loaded as the constraint

$$c(4):= ( x^2 + y^2 ) + ( x - 4) \leq 0$$

because the original left hand side and right hand side are both $gexps$ (although the right hand side is actually linear, the presence of the exponent for an mpvar forces Mosel to treat it as a nonlinear expression). It has a constant term of 0. The bracketed terms on the left hand side are both $gexps$, which are amalgamated into one general expression. The coefficient of $x$ is therefore 0 because $x$ appears only in a $gexp$. The activity of the constraint is 0 (the value of the left hand side of the rearrangement). The value of the nonlinear part is also 0, within the tolerance of the convergence.

`genctrs` can be used in printing statements. For example:

```plaintext
writeln("myGenctr = ",myGenctr)
```

where `myGenctr` is a general constraint. After a solution has been found, the value of the constraint will be printed where possible. Because of limitations in recalculating functions, this facility is generally not available for `genctrs` that contain function references. A `genctr` contained in the problem will return its activity, slack and dual values.
2.3 xvitem Objects

An xvitem corresponds to a single item in an Xpress-SLP entity of type XV (an extended variable array). The main purpose of the XV is to pass complicated lists of parameters to external functions, but the structure does have wider application. Within Mosel, the fundamental object is the xvitem, although most of the data handling is done with arrays of xvitems. Following the convention of Xpress-SLP, we shall refer to an array of objects of type xvitem as an XV.

Because an XV is a (one-dimensional) array of xvitem objects, an array of XVs is represented as a multi-dimensional array of xvitem objects. In such a case, an individual XV is described by using the reference of the first item in the XV, rather than the name of the XV array itself (see the example below in 2.4).

An xvitem object is initialized using the XVitem function.

Examples:

XX(1) := XVitem(C, "Capital")
XX(2) := XVitem("Rate", I+2)
XX(3) := XVitem("Term", 3.5)

XX(1) is defined as the mpvar C which is given the name Capital for the function
XX(2) is defined as the formula I+2 where i is an mpvar which is given the name Rate for the function
XX(3) is defined as the constant 3.5 which is given the name Term for the function

Although it is premature to explain fully about the way user functions are implemented in Mosel, the following function shows how variable names can be used:

function Interest(Values:array(vRange:range) of real,
                 FuncInfo:array(fRange:range) of integer,
                 Names:array(nRange:range) of string): real
declarations
  n: integer
  Capital, Rate, Term: real
end-declarations
Capital := 100000
Rate := 5
Term := 25
n := FuncInfo(fRange(1))
forall (i in 1..n) do
  case Names(nRange(i)) of
    "Capital" : Capital := Values(vRange(i))
    "Rate" : Rate := Values(vRange(i))
    "Term" : Term := Values(vRange(i))
  end-case
end-do
returned := Capital*(Rate/100)*Term
end-function

Without going into detail about the function, it is easy to see that it calculates a total amount of simple interest payable on a given amount of capital, at a given rate, for a given amount of time. If any of these pieces of information is missing, then a default value is used. The function is flexible because it does not require the information in any particular order and is robust because it will always give an answer. If additional information is given, it simply ignores it.

2.4 Extended variable arrays (XVs)

An XV is an array of xvitem objects. As well as allowing features such as named arguments for functions, an XV provides a convenient way of passing the same information to more than one function. For example, there might be a number of different ways of borrowing money, with different repayment conditions. The same basic information (amount, interest rate and term)
is always required, but the calculations are different (for example, simple interest, compound interest, partial repayments of capital). In a model, the same data can be passed to several functions, for example:

```mosel
declarations
  XX: array(1..n) of xvitem
  C, I: impvar ! capital and interest rate
  Cost: impvar
end-declarations
...
Cost is_free
Cost >= -Interest1(XX)
Cost >= -Interest2(XX)
Cost >= -Interest3(XX)
...
SLPloadprob(Cost)
SLPminimize
```

Defining the members of XX as in the previous section, functions Interest1 etc calculate the interest payable on different types of loan. The objective Cost is minimized and is therefore the greatest of all the "- interest" values - that is, -Cost is the smallest interest payable.

An XV is a 1-dimensional array of xvitems. An array of XVs must therefore be represented by a multi-dimensional array of xvitems — for example, a 2-dimensional array of XVs is represented by a 3-dimension array of xvitems. Extending the earlier example to (say) three different loans, we would have:

```mosel
declarations
  XVA: array(1..3,1..n) of xvitem
  C, I: array(1..3) of mpvar ! capital borrowed and interest rates
end-declarations
forall (i in 1..3) do
  XVA(i,1) := XVitem(C(i), "Capital")
  XVA(i,2) := XVitem("Rate", I(i)+2)
  XVA(i,3) := XVitem("Term", 3.5)
  XVA(i,4) := XVitem("Repayment", 50)
end-do
```

When referring to one of the XVs in an array, use the first item of the XV. For example, to refer to XV number 2 in the array XVA use XVA(2,1). The cost constraints might then be similar to the following:

```mosel
Cost >= -sum(i in 1..3) Interest1(XVA(i,1))
```

or

```mosel
Cost(i) >= -(Interest1(XVA(i,1)) +
             Interest2(XVA(i,1)) +
             Interest3(XVA(i,1)))
```

### 2.5 Arrays and XVs

A real array can be used in place of an XV (array of xvitems) whenever the XV would contain only constant values (no names, formulae or variables). Normally, only 1-dimensional arrays are supported so, if there is a need for multi-dimensional references, they must still be done as XVs.

**Example:**

```mosel
declarations
  XVA: array(1..n) of xvitem
  RA: array(1..n) of real
  X: mpvar
end-declarations
```
In the example, XVA is declared as an XV and initialized with values from the real array RA. It is then used in the constraint C1 inside a function which takes one variable (X) as well as the XV of constants. Because XVA contains only constants, the same constraint can be written using the array RA instead.

Because there is no provision for the use of real arrays as multi-dimensional XVs, the following example cannot be rewritten to avoid the use of XVs:

Normally, only 1-dimensional real arrays are used as substitutes for XVs. However, it is recognized that there are occasions when it is easier to maintain data in a multi-dimensional array, and use it directly as an XV. If this feature is used, the data will be converted into a 1-dimensional XV by storing the individual elements of the array in order with the right-hand index changing fastest. So, with a 2-dimensional array A(1..3,1..2), the values are stored in the order A(1,1), A(1,2), A(2,1), A(2,2), A(3,1), A(3,2).

To prevent the system rejecting the use of a multi-dimensional array in this way, the parameter \texttt{xslp\_max\_x\_v\_array\_dimension} must be set to a value greater than or equal to the dimension of the array (in this case, 2).

Example:
Chapter 3
Control parameters and constants

3.1 Control Parameters

Most of the Xpress-MP and Xpress-SLP control parameters can be used by name through the setparam procedure and the getparam function. Any real or integer parameter which cannot be accessed in this way (for example, it is a recent addition to the optimizer which has not yet been included in Mosel) can be set or changed using the get...control or set...control functions.

In addition to the optimizer parameters there are a few specific to SLP within Mosel as follows:

- **xslp_colorder**: Reorder matrix columns before loading the problem.
  - **Type**: Integer, read/write
  - **Values**:
    - 0: Mosel implicit ordering
    - 1, 3: Reorder using a numeric criterion
    - 2: Alphabetical order of the variable names (this requires the names to be available)
    - 4: Random order
  - **Default value**: 0

- **xslp_loadnames**: Enable/disable loading of MPS names into the Optimizer.
  - **Description**: Enable/disable loading of MPS names into the Optimizer.
  - **Type**: Integer, read/write
  - **Values**:
    - 0: Mosel implicit ordering
    - 1, 3: Reorder using a numeric criterion
    - 2: Alphabetical order of the variable names (this requires the names to be available)
    - 4: Random order
  - **Default value**: 0
Type: Boolean, read/write

Values:
- true: Enable loading of names
- false: Disable loading of names

Default value: false

Affects routines: SLPloadprob, SLPmaximize, SLPminimize, SLPexportprob

**xslp_maxxvarraydimension**

Description: Maximum number of dimensions for real arrays used as arguments to user functions

Type: Integer

Notes: If a real array with more than one dimension is to be used as an argument to a user function in a formula (that is, it is being used as an XV), then `xslp_maxxvarraydimension` must be set before the formula is encountered, to a value at least equal to the number of dimensions of the array. Arrays with larger numbers of dimensions will be faulted.

It is good practice to reset the value to 1 after the expression so that any inadvertent use of multi-dimensional real arrays is identified.

Default value: 1

Affects routines: SLPDATA("XV"), formulae containing user functions

**xslp_problem**

Description: The SLP problem pointer. This attribute is only required in applications using both Mosel and SLP at the C level.

Type: String, read only

**xslp_verbose**

Description: Enable/disable message printing by SLP

Type: Integer, read/write

Values:
- 0: Only error messages are printed
- 1: All messages are printed

Bit 1 = 0: Error messages are printed to the error output stream
Bit 1 = 1: Error messages are printed to the standard output stream

Default value: 0

Notes: `xslp_verbose` is a bit map. Bit 0 controls the amount of output; bit 1 controls where error messages are printed.
If bit 1 is not set, it is possible for error messages to appear out of place in the output; set bit 1 to redirect error messages to the same output stream as other information.
3.2 Constants

The following constants are defined:

**Xpress-SLP callbacks:**

- `xslp_cascadeend` at end of SLP variable cascading
- `xslp_cascadestart` at start of SLP variable cascading
- `xslp_cascadevar` after cascading of each SLP variable
- `xslp_intsol` in MISLP when an integer solution is found
- `xslp_iterend` at end of each SLP iteration
- `xslp_iterstart` at start of each SLP iteration
- `xslp_itervar` after testing for convergence of each SLP variable
- `xslp_optnode` in MISLP when an optimal solution is found
- `xslp_prenode` in MISLP before a node is optimized
- `xslp_slpdestroy` when an SLP problem is being destroyed
- `xslp_slpend` when SLP terminates
- `xslp_slpmessage` message printing
- `xslp_slpnode` in MISLP after a node has been solved
- `xslp_slpstart` when SLP optimization starts

For compatibility with previous versions, callback constants of the form `XSLPxxxx` are accepted as equivalent to `xslp_xxxx`.

**Xpress-SLP SLPgetindex types:**

- `xslp_cvnames` Character variable
- `xslp_internalfuncnames` Internal function (case sensitive)
- `xslp_internalfuncnamesnocase` Internal function (case insensitive)
- `xslp_userfuncnames` User function (case sensitive)
- `xslp_userfuncnamesnocase` User function (case insensitive)
- `xslp_xvnames` XV (extended variable array)

**Xpress-SLP user function constants:**

- `xslp_funcinfosize` Required size of function and argument information array
- `xslpinfo_callerflag` Caller flag
- `xslpinfo_funcnum` Index number of called function
- `xslpinfo_instance` Instance number of called function
- `xslpinfo_ndelta` Number of sets of partial derivatives required
- `xslpinfo_ninput` Number of input variables provided
- `xslpinfo_ninstring` Number of items in list of input variable names
- `xslpinfo_noutput` Number of output variables required
- `xslpinfo_noutstring` Number of items in list of output variable names
## Chapter 4
### Functions and Procedures

All of the functions and procedures of the `mmxprs` module are available to SLP users. In addition, the following functions and procedures are provided for use in SLP:

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>chgtolset</td>
<td>Change values in an SLP tolerance set or create a new set</td>
<td>13</td>
</tr>
<tr>
<td>chgvartolset</td>
<td>Set or change the tolerance set for an SLP variable</td>
<td>14</td>
</tr>
<tr>
<td>clearinitvals</td>
<td>Clear initial value settings for all SLP variables</td>
<td>15</td>
</tr>
<tr>
<td>copySolToInit</td>
<td>Set the initial values of all SLP variables to their value in the current solution</td>
<td>16</td>
</tr>
<tr>
<td>cXSLPcommand</td>
<td>Execute an SLP console command which returns a string result</td>
<td>17</td>
</tr>
<tr>
<td>Func</td>
<td>Include an SLP or user function in an expression</td>
<td>18</td>
</tr>
<tr>
<td>getact</td>
<td>Returns the value of a general constraint or of a general expression</td>
<td>19</td>
</tr>
<tr>
<td>getcoeff</td>
<td>Return the coefficient or constant term in a constraint</td>
<td>20</td>
</tr>
<tr>
<td>getconverged</td>
<td>Return the convergence status of a variable</td>
<td>21</td>
</tr>
<tr>
<td>getcsol</td>
<td>Return the activity or reduced cost of a variable</td>
<td>22</td>
</tr>
<tr>
<td>getdblattrib</td>
<td>Get the value of an SLP double (real) attribute</td>
<td>23</td>
</tr>
<tr>
<td>getdblcontrol</td>
<td>Get the value of an SLP double (real) control parameter</td>
<td>24</td>
</tr>
<tr>
<td>getdual</td>
<td>Returns the dual solution value of a general constraint</td>
<td>25</td>
</tr>
<tr>
<td>gethistory</td>
<td>Return the history value of a variable</td>
<td>26</td>
</tr>
<tr>
<td>getintattrib</td>
<td>Get the value of an SLP integer attribute</td>
<td>27</td>
</tr>
<tr>
<td>getintcontrol</td>
<td>Get the value of an SLP integer control parameter</td>
<td>28</td>
</tr>
<tr>
<td>getname</td>
<td>Return the name in the problem of the requested item</td>
<td>29</td>
</tr>
<tr>
<td>getnlcoeff</td>
<td>Return the current value of the nonlinear part of a constraint</td>
<td>30</td>
</tr>
<tr>
<td>getrsol</td>
<td>Return the slack or dual solution value of a constraint</td>
<td>31</td>
</tr>
<tr>
<td>getslack</td>
<td>Return the slack value of a general constraint</td>
<td>32</td>
</tr>
<tr>
<td>getsol</td>
<td>Return the primal solution value of a constraint, or the calculated value of a general expression</td>
<td>33</td>
</tr>
<tr>
<td>getstatus</td>
<td>Return the status value of an mpvar</td>
<td>34</td>
</tr>
<tr>
<td>getstepbound</td>
<td>Return the current value of the step bound of an mpvar</td>
<td>35</td>
</tr>
</tbody>
</table>
gettolset  Get the values for an SLP tolerance set.  p. 36
gettype    Returns the type of a general constraint         p. 37
getvarnum  Return the index of an mpvar in the problem p. 38
getvartolset Get the tolerance set for an SLP variable p. 39
interp     General-purpose linear interpolation function p. 41
ishidden   Test whether a general constraint is hidden p. 40
iXSLPcommand Execute an SLP console command which returns an integer result p. 43
rXSLPcommand Execute an SLP console command which returns a real result p. 44
setdblcontrol Set the value of an SLP double (real) control parameter p. 45
sethidden   Hide or unhide a general constraint         p. 46
setinitval  Provide an initial value for an SLP variable p. 48
setintcontrol Set the value of an SLP integer control parameter p. 47
settype    Set the type of a general constraint         p. 49
SLPcalluserfunc Call a user function directly from Mosel p. 50
SLPDATA     Provide SLP-specific problem information   p. 52
SLPexportprob Write the SLP problem to file in extended MPS or text format p. 53
SLPgetfuncinfo Obtain additional information for a user function p. 54
SLPgetindex  Obtain the index of an SLP item from its name p. 55
SLPgetline   Get the name of an SLP item corresponding to a given index p. 56
SLPglobal   Call the Xpress-SLP mixed integer optimizer (MISLP) p. 57
SLPloadprob  Loads the nonlinear problem into the Xpress-SLP optimizer p. 58
SLPlogfile   Specify the name of a file to receive any output messages from the Xpress-SLP program p. 59
SLPmaximize, SLPminimize Maximize or minimize the current SLP problem p. 60
SLPprintmemory Print memory usage for Xpress-SLP and the mmxslp module. p. 61
SLPscaling   Call the Xpress-SLP scaling function to report on the range of values in the problem p. 62
SLPsetcallback Set a Mosel function to be used as an SLP callback p. 63
SLPsetprobstat This function sets the problem status parameters after an optimization p. 65
SLPsetuserfuncaddress Set the address of an external user function p. 66
SLPsetuserfuncinfo Set the function information array before calling a user function p. 67
SLPtime      Get a time reference value                    p. 68
SLPvalidate  Validate an SLP solution                    p. 69
tolsetstatus Return the bitmap status of an SLP tolerance set p. 70
XSLPcommand Execute an SLP console command and optionally return a value p. 71
XVitem       Set the values for an xvitem object          p. 72
chgtolset

Purpose
Change values in an SLP tolerance set or create a new set

Synopsis
function chgtolset(TolSet:integer, Status:integer, Tols:array of real) :integer

Arguments
TolSet the number of the tolerance set to be changed. If TolSet is zero, a new set will be created.
Status an integer containing a bitmap of the tolerances being set. For details of these, see the Xpress-SLP Reference Manual.
Tols an array of 9 real values, containing the tolerances.

Return value
The number of the affected tolerance set. If TolSet is zero, this will be the number of the tolerance set created; otherwise it will be equal to TolSet.

Example
The following example creates a new tolerance set (the first argument is zero) and sets tolerances TA and TM (bits 1 and 2 of Status) to 0.005 and 0.008 respectively. Other tolerances remain at the default. If there are already K tolerance sets in existence, the return value will be K+1.

    declarations
    Tols: array(1..9) of real
    n: integer
    end-declarations

    Tols(2) := 0.005
    Tols(3) := 0.008
    n := chgtolset(0,6,Tols)

Related topics
gettollset, tolsetstatus
**chgvartolset**

**Purpose**
Set or change the tolerance set for an SLP variable

**Synopsis**
procedure chgvartolset(Var:mpvar, TolSet:integer)

**Arguments**
- **Var** The SLP variable whose tolerance set is to be changed
- **TolSet** The index of the tolerance set to be used

**Example**
The following code fragment creates a new tolerance set, and applies it to variable `Var(3)`:  
```
declarations
  Var: array (1..10) of mpvar
  Tols: array(1..9) of real
  TolSet: integer
end-declarations

Tols(2) := 0.005
Tols(3) := 0.008
TolSet := chgtolset(0,6,Tols)
chgvartolset(Var(3),TolSet)
```

**Further information**
If the variable is not currently an SLP variable, it will be marked as an SLP variable

**Related topics**
chgtolset, gettolset, getvartolset
clearinitvals

**Purpose**
Clear initial value settings for all SLP variables

**Synopsis**
procedure clearinitvals

**Example**
The following code fragment assigns the initial values 1, 1.1, 1.2 etc to the variables $V(i)$, solves the problem, clears the initial values and solves again:

```
declarations
    V: array(1..10) of mpvar
end-declarations

forall (i in 1..10) setinitval(V(i),1+(i-1)/10)
...
SLPmaximize
...
clearinitvals
SLPmaximize
```

**Further information**
This function clears any initial values already set, so that a future optimization uses default values or any that are set subsequently.

**Related topics**
setinitval, copiesoltoinit, SLPDATA("IV")
copysoltoinit

**Purpose**
Set the initial values of all SLP variables to their value in the current solution

**Synopsis**
procedure copysoltoinit

**Example**
The following code fragment assigns the initial values 1, 1.1, 1.2 etc to the variables \( V(i) \), solves the problem, sets the initial values to those from the solution and solves again:

```plaintext
declarations
  V: array(1..10) of mpvar
end-declarations

forall (i in 1..10) setinitval(V(i),1+(i-1)/10)
...
SLPmaximize
...
copysoltoinit
SLPmaximize
```

**Further information**
This function sets the initial value of each SLP variable to its current solution value. A good starting point can help to find a solution and the values from one solution are often suitable as a starting point for a similar problem. Individual values can still be assigned after `copysoltoinit` and before the optimization.

**Related topics**
clearinitvals, setinitval, SLPDATA("IV")
cXSLPcommand

Purpose
Execute an SLP console command which returns a string result

Synopsis
function cXSLPcommand(cmd1:string [,cmd2:real|string] [,cmd3:real|string]) :string

Arguments
- cmd1 : String containing the first part or all of the SLP console command
- cmd2 : Optional string or value to be appended to cmd1
- cmd3 : Optional string or value to be appended to cmd2

Return value
The result of the console command if it is a string, or an empty string otherwise.

Further information
See XSLPcommand for more details.

Related topics
XSLPcommand
Func

Purpose
Include an SLP or user function in an expression

Synopsis
Func (fName:string [, arg1:real|mpvar|linctr|XV|gexp [, arg2:real|mpvar|linctr|XV|gexp]] [, rIndex:string]) :gexp

Arguments
- **fName** string containing the name of the function being evaluated. This can be an Xpress-SLP internal function or a user-defined function
- **arg1** (optional) first argument to the function. This can be of any type (real, mpvar or expression) acceptable to the function. Arguments of type XV are always acceptable, provided the items in the XV are of the correct type
- **arg2** (optional) second argument to the function. The requirements are the same as for arg1
- **rIndex** (optional) string indicating the index or name of the return value from the function. This is only required when the function returns an array of values

Return value
For most purposes, **Func** can be regarded as returning a gexp which contains the requested evaluation of the function.
Strictly speaking, **Func** returns a gexp describing how to call the specified function, and the evaluation of the function then returns a real value.

Further information
See **Func (General SLP function)** for a fuller description of **Func**.

Related topics
**Func (General SLP function)**
getact

Purpose
Returns the value of a general constraint or of a general expression

Synopsis
function getact(c:genctr | g:gexp) :real

Arguments
c a general constraint (genctr)
g a general expression (gexp)

Return value
For a general constraint the left hand side of the constraint or 0;
for a general expression, the evaluation of the expression using the solution values of the
variables.

Example
The following example solves a problem containing the two constraints c1 and c2, and then
prints the values (left hand sides) of the constraints:

declarations
c1, c2: genctr
x, y: mpvar
end-declarations

c1:= x^2 + y^2 <= 4
c2:= x*y >= 1

SLPloadprob(x+y)
SLPmaximize
writeln("c1=",getact(c1)," c2=",getact(c2))

Further information
Because of limitations in recalculating functions, this facility is only available for gexp that do
not contain function references.
genctr contained in the problem will always return their left-hand side value.

Related topics
getdual, getslack, getsol
getcoeff

Purpose
Return the coefficient or constant term in a constraint

Synopsis
function getcoeff(c:genctr [, x:mpvar]) :real

Arguments
  c      a general constraint
  x      a decision variable

Return value
If a variable is specified as well as a constraint, the return value is the coefficient in that variable
and constraint.
If no variable is specified, the return value is the constant term in the constraint

Example
The following example retrieves a coefficient and the constant term of a constraint:

declarations
  c1, c2: genctr
  x, y: mpvar
end-declarations

  c1:= x^2 + y^2 <= 4 - x
  c2:= x*y >= 1

SLPloadprob(x+y)
SLPmaximize
writeln("(c1,x)=" , getcoeff(c1,x)," c2=" , getact(c2))

Further information
See the discussion on General Constraint Expressions for more details on the way in which the
formulation of a constraint affects the way it is loaded into the problem.

Related topics
getnlcoeff
getconverged

**Purpose**
Return the convergence status of a variable

**Synopsis**

```plaintext
function getconverged(v:mpvar | i:integer) :integer
```

**Arguments**

- `v` an mpvar
- `i` an integer, being the index of an mpvar

**Return value**
The convergence status of the variable

**Example**
The following example shows part of a callback function which is used to check the convergence status of a variable before attempting potentially more expensive analysis:

```plaintext
function ConvergeTest(V: integer) : integer
  if getconverged(V) = 0 then
    returned := 0 ! variable has converged
  else
    ! ... additional testing on variable
    ! ...
    returned := 99
  end-if
end-function
```

**Further information**
This function returns the convergence status of a variable. A status of zero means the variable is not converged. Values 1 to 9 are reserved for convergence on system-defined convergence criteria. Values of 10 or more are for convergence on user-defined criteria.
getcsol

Purpose
Return the activity or reduced cost of a variable

Synopsis
function getcsol(cIndex:integer, vType:integer) :real

Arguments
  cIndex      the index of an mpvar or a column in the problem
  vType       integer indicating the type of information returned:
               0        activity
               1        reduced cost

Return value
The activity (if vType=0) or the reduced cost (if vType=1) of the mpvar or column whose index
is given as cIndex. Zero will be returned if there is no solution or if cIndex is outside the
range of valid indices.

Example
The following code fragment solves a problem and retrieves the values of column number 42:

    SLPmaximize
    writeln("Column 42 activity=",getcsol(42,0))
    writeln("Column 42 reduced cost=",getcsol(42,1))

Further information
The column index can be obtained by getvarnum or SLPgetindex.

Related topics
getact, getrsol, getsol
getdblattrib

Purpose
Get the value of an SLP double (real) attribute

Synopsis
function getdblattrib(Type:string, Param:integer|string) :real

Arguments
Type A character string beginning with "L" for the linear problem or "M" for the MIP (integer) problem in MISLP
Param The number or name of the required parameter. The number is given in the header file for the corresponding Xpress module (for example, xprs.h or xslp.h). If the name is used, it must be preceded by "XSLP_" or "XPRS_" as appropriate (for example, "xslp_errorcosts" or "xprs_lpobjval").

Return value
The current value of the attribute (parameter set by the system) in the appropriate problem

Example
The following code fragment from a callback within a MISLP optimization prints the current best bound:

    writeln("Best bound = ",getdblattrib("M",2004)))

2004 is parameter xprs_bestbound

Further information
The get...control and get...attrib functions are identical to getparam for the linear problem (Type="L"). During MISLP, MIP-specific information must be retrieved from the MIP problem (Type="M") because the linear problem is solving only subproblems and is not in MIP mode. Note that the required parameter can be specified as an integer or as a name. Specifying as an integer is generally more efficient. The value corresponding to a particular parameter can be found in the header file for the appropriate module (xprs.h or xslp.h). This function is also useful for setting parameters which have not been defined by name in the Mosel system.

Related topics
getdblcontrol, setdblcontrol, getintattrib
getdblcontrol

Purpose
Get the value of an SLP double (real) control parameter

Synopsis
function getdblcontrol(Type:string, Param:integer|string) :real

Arguments
Type    A character string beginning with "L" for the linear problem or "M" for the MIP (integer) problem in MISLP
Param   The number or name of the required parameter. The number is given in the header file for the corresponding Xpress module (for example, xprs.h or xslp.h). If the name is used, it must be preceded by "XSLP_" or "XPRS_" as appropriate (for example, "xslp_errorcost" or "xprs_feastol").

Return value
The current value of the parameter in the appropriate problem

Example
The following code fragment from a callback within a MISLP optimization prints the current value of \texttt{mipabscutoff}:

\begin{verbatim}
writeln("Abs CutOff = ",getdblcontrol("M",7013))
\end{verbatim}

7013 is parameter \texttt{xprs_mipabscutoff}

Further information
The \texttt{get...control} and \texttt{get...attrib} functions are identical to \texttt{getparam} for the linear problem (\texttt{Type="L"}). During MISLP, MIP-specific information must be retrieved from the MIP problem (\texttt{Type="M"}) because the linear problem is solving only subproblems and is not in MIP mode. Note that the required parameter can be specified as an integer or as a name. Specifying as an integer is generally more efficient. The value corresponding to a particular parameter can be found in the header file for the appropriate module (\texttt{xprs.h} or \texttt{xslp.h}). This function is also useful for setting parameters which have not been defined by name in the Mosel system.

Related topics
\texttt{getintattrib}, \texttt{getintcontrol}, \texttt{setdblcontrol}
getdual

Purpose
Returns the dual solution value of a general constraint

Synopsis
function getdual(c:genctr) :real

Argument
c a general constraint

Return value
The dual solution value of the constraint, or zero if the problem has not been optimized or the
constraint is not part of the current problem.

Example
The following example solves a problem and retrieves the dual value of a constraint:

declarations
  c1, c2: genctr
  x, y: mpvar
end-declarations

  x <= 10
  y <= 10
  c1:= x^2 + y^2 <= 4 - x
  c2:= x*y >= 1

  SLPloadprob(x+y)
  SLPmaximize
  writeln("c1 dual value=",getdual(c1))

Related topics
getact, getslack, getsol
gethistory

Purpose
Return the history value of a variable

Synopsis
function gethistory(v:mpvar | i:integer) :integer

Arguments
v
an mpvar
i
an integer, being the index of an mpvar

Return value
The history value of the variable, or zero if the variable does not have a history or is not part of the current problem.

Example
The following example shows part of a callback function which is used to check the progress of the convergence of a variable:

```mosel
function ConvergeTest(V: integer) : integer
declarations
    History:integer
end-declarations
History := gethistory(V)
if History > 1 then
    writeln(SLPgetname(2,V)," has increased ",
         History," times in succession")
elsif History < -1 then
    writeln(SLPgetname(2,V)," has decreased ",
         History," times in succession")
end-if
end-function
```

Further information
This function returns the history value of an mpvar. The value describes the number of consecutive movements in the same direction which reached the step bound. Positive values of history are for positive movements (the value of the variable increases), negative values of history are for negative movements. When the direction changes sign, the history will be reset to +1 or -1.

If the argument is an integer, it is interpreted as the SLP index of the mpvar.
getintattrib

Purpose
Get the value of an SLP integer attribute

Synopsis
function getintattrib(Type:string, Param:integer|string) :integer

Arguments
Type A character string beginning with "L" for the linear problem or "M" for the MIP (integer) problem in MISLP
Param The number or name of the required parameter. The number is given in the header file for the corresponding Xpress module (for example, xprs.h or xslp.h). If the name is used, it must be preceded by "XSLP_" or "XPRS_" as appropriate (for example, "xslp_ufs" or "xprs_cols").

Return value
The current value of the attribute (parameter set by the system) in the appropriate problem

Example
The following code fragment from a callback within a MISLP optimization prints the current node and its parent:

 writeln("Node: ",getintattrib("M",1013))
 writeln("Parent: ",getintattrib("M",1027))

1013 is parameter xprs_nodes
1027 is parameter xprs_parentnode

Further information
The \texttt{get...control} and \texttt{get...attrib} functions are identical to \texttt{getparam} for the linear problem (Type="L"). During MISLP, MIP-specific information must be retrieved from the MIP problem (Type="M") because the linear problem is solving only subproblems and is not in MIP mode. Note that the required parameter can be specified as an integer or as a name. Specifying as an integer is generally more efficient. The value corresponding to a particular parameter can be found in the header file for the appropriate module (xprs.h or xslp.h). This function is also useful for setting parameters which have not been defined by name in the Mosel system.

Related topics
getdblattrib, getintcontrol, setintcontrol
getintcontrol

Purpose
Get the value of an SLP integer control parameter

Synopsis
function getintcontrol(Type:string, Param:integer|string) :integer

Arguments
Type
A character string beginning with "L" for the linear problem or "M" for the MIP
(integer) problem in MISLP

Param
The number or name of the required parameter. The number is given in the
header file for the corresponding Xpress module (for example, xprs.h or xsip.h).
If the name is used, it must be preceded by "XSLP_" or "XPRS_" as appropriate
(for example, "xslp_algorithm" or "xprs_lpiterlimit").

Return value
The current value of the parameter in the appropriate problem

Example
The following code fragment from a callback within a MISLP optimization prints the current
value of presolve:

    writeln("Presolve = ", getintcontrol("M",8011))

8011 is parameter xprs_presolve

Further information
The get...control and get...attrib functions are identical to getparam for the linear
problem (Type="L"). During MISLP, MIP-specific information must be retrieved from the MIP
problem (Type="M") because the linear problem is solving only subproblems and is not in
MIP mode. Note that the required parameter can be specified as an integer or as a name.
Specifying as an integer is generally more efficient. The value corresponding to a particular
parameter can be found in the header file for the appropriate module (xprs.h or xsip.h).
This function is also useful for setting parameters which have not been defined by name in the
Mosel system.

Related topics
getdblcontrol, getintattrib, setintcontrol
getname

Purpose
Return the name in the problem of the requested item

Synopsis
function getname(c:genctr | xv:array of xvitem | xvi:xvitem) :string

Arguments
- c a genctr
- xv an array of xvitem objects (an XV)
- xvi an xvitem object

Return value
The name in the problem of the requested item, or an empty name if the item is not in the current problem.

Example
The following example loads a problem and then writes the names of some of the items in the problem:

declarations
rx: array(A) of genctr
XVA: array(A,1..10) of xvitem
end-declarations
forall (a in A) writeln(getname(rx(a)))
forall (a in A) writeln(getname(XVA(a,1)))

The first usage writes the names of the members of an array of genctrs. The second example shows how to handle an array of XVs. A 1-dimensional array of XVs is described as a 2-dimensional array of xvitems (since each XV is itself a 1-dimensional array of xvitems). Use the first item in the XV to access data for the XV.

Further information
This function is only effective when a problem has been loaded.

Related topics
SLPgetname, SLPgetindex
**getnlcoeff**

**Purpose**
Return the current value of the nonlinear part of a constraint

**Synopsis**
function getnlcoeff(c:genctr) :real

**Argument**
c a general constraint

**Return value**
The current value of the nonlinear coefficient term in the constraint

**Example**
The following example solves a problem and retrieves the optimal value of the nonlinear coefficient of a constraint:

```mosel
declarations
c1, c2: genctr
x, y: mpvar
end-declarations

x <= 10
y <= 10
c1:= x^2 + y^2 <= 4 - x
c2:= x*y >= 1

SLPloadprob(x+y)
SLPmaximize
writeln("c1 nonlinear coefficient value=",getnlcoeff(c1))
```

**Further information**
See the discussion on General Constraint Expressions for more details on the way in which the formulation of a constraint affects the way it is loaded into the problem.

**Related topics**
getcoeff
getrsol

Purpose
Return the slack or dual solution value of a constraint

Synopsis
function getcsol(rIndex:integer, vType:integer) :real

Arguments
rIndex  
the index of a constraint in the problem
vType   
integer indicating the type of information returned:
        0.slack value
        1.dual solution value

Return value
The slack (if vType=0) or the dual solution value (if vType=1) of the constraint whose index is
given as rIndex. Zero will be returned if there is no solution or if rIndex is outside the range
of valid indices.

Example
The following code fragment solves a problem and retrieves the values of constraint number
42:

SLPmaximize
writeln("Constraint 42 slack=",getrsol(42,0))
writeln("Constraint 42 dual value=",getrsol(42,1))

Further information
The constraint index can be obtained from the constraint name by using SLPgetindex.

Related topics
getslack, getcsol, getsol
getslack

Purpose
Return the slack value of a general constraint

Synopsis
function getslack(c:genctr) :real

Argument
c
a general constraint

Return value
The slack value of the constraint, or zero if the constraint is not part of the current problem.

Example
The following example solves a problem and retrieves the slack value of a constraint:

declarations
c1, c2: genctr
x, y: mpvar
end-declarations

x <= 10
y <= 10
c1:= x^2 + y^2 <= 4 - x
c2:= x*y >= 1

SLPloadprob(x+y)
SLPmaximize
writeln("c1 slack value=",getslack(c1))

Further information
This function returns the slack value of the constraint if the problem has been solved successfully and the constraint is in the problem, or zero otherwise.

Related topics
getact, getdual, getsol
**getsol**

**Purpose**
Return the primal solution value of a constraint, or the calculated value of a general expression.

**Synopsis**

```plaintext
function getsol(c:genctr | g:exp) :real
```

**Arguments**
- `c` a general constraint
- `g` a general expression

**Return value**
For a general constraint, the primal solution value of the constraint, or zero if there is no solution or the constraint is not part of the current problem.
For a general expression, the evaluation of the expression using the solution values of the variables; zero is used as the value of a variable if it is not in the current problem, or if there is no solution.

**Example**
The following example solves a problem and retrieves the values of a constraint and of a general expression:

```plaintext
declarations
  c1, c2: genctr
  x, y: mpvar
end-declarations

  x <= 10
  y <= 10
  c1:= x^2 + y^2 <= 4 - x
  c2:= x*y >= 1

SLPloadprob(x+y)
SLPmaximize
writeln("c1 solution value=", getsol(c1))
writeln("x^2=", getsol(x^2))
```

**Further information**
This function requires the existence of a solution to the SLP problem. If it is called at an intermediate point during an optimization, `SLPsetprobstat` may be needed to ensure that solution values can be obtained.

**Related topics**
- `getact`, `getdual`, `getslack`
getstatus

Purpose
Return the status value of an mpvar

Synopsis
function getstatus(v:mpvar | i:integer) :integer

Arguments
v an mpvar
i an integer, being the index of an mpvar

Return value
The SLP status value of the mpvar

Example
The following code fragment checks the status of variable v(i) and if the variable has a step bound, the value of the bound is retrieved:

```
if bittest(getstatus(v(i)),131072) > 0 then
  sb := getstepbound(v(i))
```

Further information
For SLP variables, the status contains information about the variable as a bit pattern. The following bits are potentially useful:

16 (=65536) Variable has converged
17 (=131072) Variable has an active step bound
20 (=1048576) Variable is not an SLP variable

Other bits are set for internal use, so the bits should always be tested using the bittest function.
If the argument is an integer, it is interpreted as the index of an mpvar.
getstepbound

**Purpose**
Return the current value of the step bound of an mpvar

**Synopsis**
function getstepbound(v:mpvar | i:integer) :real

**Arguments**
v an mpvar
i an integer, being the index of an mpvar

**Return value**
The value of the step bound of the mpvar or zero if there is no step bound

**Example**
The following code fragment checks the status of variable v(i) and if the variable has a step bound, the value of the bound is retrieved:

```plaintext
if bittest(getstatus(v(i)),131072) > 0 then
    sb := getstepbound(v(i))
```

**Further information**
gettolset

Purpose
Get the values for an SLP tolerance set.

Synopsis
procedure gettolset(TolSet:integer, Tols:array of real)

Arguments
TolSet the number of the SLP tolerance set to be obtained
Tols an array of 9 real values to contain the tolerance values

Example
The following example obtains the values and status for tolerance set number 1 and prints the
tolerance values which have been set:

declarations
   Tols: array(1..9) of real
   Status: integer
end-declarations
gettolset(1,Tols)
Status := tolsetstatus(1)
forall (i in 1..9) do
   if bittest(Status,1) <> 0 then
      writeln("Tolerance ",i," = ",Tol(i))
   end-if
   Status := Status / 2
end-do

Further information
Use tolsetstatus to determine which values in Tols are activated.

Related topics
chgtolset, tolsetstatus
gettype

Purpose
Returns the type of a general constraint

Synopsis
function gettype(c:genctr) :integer

Argument
 c        a general constraint

Return value
The type of the constraint. This will be one of:

XPRM_CTYPE_LEQ       ≤
XPRM_CTYPE_EQ        =
XPRM_CTYPE_GEQ       ≥
XPRM_CTYPE_UNCONS    not constraining or not defined

Example
The following code fragment retrieves the type of a previously-defined constraint:

declarations
   c1, c2: genctr
   x, y: mpvar
end-declarations

c1:= x^2 + y^2 <= 4 - x
...
case (gettype(c1)) of
   XPRM_CTYPE_LEQ: writeln("c1 is less than or equal")
   XPRM_CTYPE_EQ: writeln("c1 is equal")
   XPRM_CTYPE_GEQ: writeln("c1 is greater than or equal")
   XPRM_CTYPE_UNCONS: writeln("c1 is not constraining")
end-case

Further information
The problem does not need to be loaded in order to retrieve the type of a constraint.

Related topics
settype
getvarnum

Purpose
Return the index of an mpvar in the problem

Synopsis
function getvarnum(v:mpvar) :integer

Argument
v an mpvar

Return value
The index number of the mpvar or -1 if the problem has not been loaded or the variable is not in the current problem.

Example
The following code fragment obtains the index for variable v(3) and then uses it to obtain the SLP status of the variable:

declarations
    vNum, Status: integer
    v: array(1..10) of mpvar
    OBJ: linctr
end-declarations
...
SLPloadprob(OBJ)
SLPmaximize
vNum := getvarnum(v(3))
Status := getstatus(vNum)

Related topics
SLPgetindex
getvartolset

Purpose
Get the tolerance set for an SLP variable

Synopsis
function getvartolset(Var:mpvar) :integer

Argument
Var The SLP variable whose tolerance set is to be returned

Return value
The index of the tolerance set for the variable. If the variable is not an SLP variable, or has the
default tolerances, the return value is zero.

Example
The following code fragment finds the tolerance set for variable Var(3):

declarations
Var:array (1..10) of mpvar
Status, TolSet: integer
end-declarations

Status := getstatus(Var(3))
if bittest(Status,1048576) <> 0 then
  writeln("Var(3) is not an SLP variable")
else
  TolSet := getvartolset(Var(3))
  writeln("Var(3) has tolerance set number ",TolSet)
end-if

Further information
Use getstatus and test bit 20 (=1048576) to check if the variable is an SLP variable.

Related topics
chgtolset, chgvartolset, getstatus, gettolset
ishidden

Purpose
Test whether a general constraint is hidden

Synopsis
function ishidden(cName:genctr) :boolean

Argument
cName     name of a genctr

Return value
ture     cName is hidden
false      cName is not hidden

Example
The following example tests the status of the genctr OptRow and prints an appropriate message.

if ishidden(OptRow) then
    writeln("OptRow is hidden")
else
    writeln("OptRow is not hidden")
end-if

Further information
This function returns the "hidden" status of a genctr. A hidden genctr still exists in the model, but will be omitted from the SLP problem which is loaded and solved.

Related topics
sethidden
**interp**

**Purpose**
General-purpose linear interpolation function

**Synopsis**

1. `function interp(VR:real, XY: array(...) of real) :real`
2. `function interp(VR:real, X: array(...) of real,
   Y: array(...) of real) :real`
3. `function interp(VC:linctr, XY: array(...) of real) :gexp`
4. `function interp(VC:linctr, XV: array(...) of xvitem) :gexp`
5. `function interp(VG:gexp, XY: array(...) of real) :gexp`
6. `function interp(VG:gexp, XV: array(...) of xvitem) :gexp`

**Arguments**

- **VR**: real value to be interpolated
- **VC**: linear expression to be interpolated
- **VG**: general expression to be interpolated
- **XY**: real array of pairs of values with which to interpolate
- **X**: real array of X-values with which to interpolate
- **Y**: real array of Y-values with which to interpolate
- **XV**: array of pairs of xvitems with which to interpolate

**Return value**

Forms (1) and (2) return the interpolated value. The remaining forms return the formula for interpolating the required value.

**Example**

Linear interpolation is typically used to estimate the value of a function at a point from known values at two adjacent points. As an example, suppose we want to estimate \(x^2\), but are given values only at \(x = 1\), \(2\), \(3\), \(4\), and \(5\). Then, to estimate the square of \(x = 2.5\) we would take the value half way between \(x = 2\) and \(x = 3\) and obtain the result 6.5. The `interp` function performs the same operation. Its first argument is the value (constant or variable) that we are trying to interpolate. The other argument(s) contain the given values, either in a 2-dimensional real array of corresponding X and Y values (as in XY2 below), two 1-dimensional real arrays of X and corresponding Y values (as in X and Y below), or in a 1-dimensional array of values which contain successive pairs of values X and its corresponding Y (as in the remaining arrays below).

```mosel
parameters
  n = 5
end-parameters
declarations
  XY2: array(1..n,1..2) of real
  XY: array(1..n*2) of real
  X, Y: array(1..n) of real
  XV: array(1..n*2) of xvitem
  V: mpvar
  R: real
dend-declarations
X := [1, 2, 3, 4, 5]
Y := [1, 4, 9, 16, 25]
XY:=[1,1, 2,4, 3,9, 4, 16, 5,25]
forall (i in 1..n) do
  XY2(i,1) := X(i)
  XY2(i,2) := Y(i)
end-do
forall (i in 1..n*2) do
```

---

Functions and Procedures 41 Xpress-Mosel Program Reference Manual (Xpress-SLP Section)
\[ XV(i) := XVitem(XY(i)) \]
\[ \text{end-do} \]
\[ R := 2.5 \]
\[ \text{writeln("Estimate (XY) for \"R\" is \", interp(R, XY))} \]
\[ \text{writeln("Estimate (XY2) for \"R\" is \", interp(R, XY2))} \]
\[ \text{writeln("Estimate (X,Y) for \"R\" is \", interp(R, X, Y))} \]

\[ \text{setparam("xslp\_MaxXVArrayDimension",2)} \]
\[ \text{interp(V,XY2) <= 10} \]
\[ \text{setparam("xslp\_MaxXVArrayDimension",1)} \]
\[ \text{interp(V+6,XY) <= 16} \]
\[ \text{interp(V^2,XV) <= 20} \]

The first three uses of \texttt{interp} calculate an immediate value (all the arguments are constant). The last three uses all produce constraints (at least one of the arguments is not constant). The real arrays are effectively being used as XVs. Normally, only 1-dimensional arrays can be used. If a multi-dimensional array is to be used, then the parameter \texttt{xslp\_maxxvarraydimension} must be set before the array is first referenced as an XV The parameter should be returned to its standard value (1) afterwards so that any inadvertent uses of multi-dimensional arrays are identified.

**Further information**

More complex uses of \texttt{interp} may require the use of the \texttt{Func} wrapper and custom-built XVs

**Related topics**

\texttt{Func}
iXSLPcommand

Purpose
Execute an SLP console command which returns an integer result

Synopsis
function iXSLPcommand(cmd1:string [,cmd2:real|string] [,cmd3:real|string]) :integer

Arguments
- **cmd1**: String containing the first part or all of the SLP console command
- **cmd2**: Optional string or value to be appended to **cmd1**
- **cmd3**: Optional string or value to be appended to **cmd2**

Return value
The result of the console command if it is an integer, or zero otherwise.

Further information
See XSLPcommand for more details.

Related topics
XSLPcommand
**rXSLPcommand**

**Purpose**
Execute an SLP console command which returns a real result

**Synopsis**
function rXSLPcommand(cmd1:string [,cmd2:real|string] [,cmd3:real|string]) :real

**Arguments**
cmd1 String containing the first part or all of the SLP console command
cmd2 Optional string or value to be appended to cmd1
cmd3 Optional string or value to be appended to cmd2

**Return value**
The result of the console command if it is a real number, or zero otherwise.

**Further information**
See XSLPcommand for more details.

**Related topics**
XSLPcommand
setdblcontrol

Purpose
Set the value of an SLP double (real) control parameter

Synopsis
procedure setdblcontrol(Type:string, Param:integer|string, Value:real)

Arguments
Type   A character string beginning with "L" for the linear problem or "M" for the MIP (integer) problem in MISLP
Param  The number or name of the required parameter. The number is given in the header file for the corresponding Xpress module (for example, xprs.h or xslp.h). If the name is used, it must be preceded by "XSLP_" or "XPRS_" as appropriate (for example, "xslp_errorcosts" or "xprs_feastol").
Value  The value to which the parameter is to be set

Example
The following code fragment from a callback within a MISLP optimization sets the value of mipabscutoff to infinity (1.0e+20):

    setdblcontrol("M",7013,1.0e+20))

7013 is parameter xprs_mipabscutoff

Further information
The set...control functions are identical to setparam for the linear problem (Type="L"). During MISLP, MIP parameters must be set in the MIP problem (Type="M") because the linear problem is solving only subproblems and is not in MIP mode. Note that the required parameter can be specified as an integer or as a name. Specifying as an integer is generally more efficient. The value corresponding to a particular parameter can be found in the header file for the appropriate module (xprs.h or xslp.h). This function is also useful for setting parameters which have not been defined by name in the Mosel system.

Related topics
getdblcontrol
sethidden

Purpose
Hide or unhide a general constraint

Synopsis
procedure sethidden(cName:genctr, tf:boolean)

Arguments
cName name of a genctr
tf constraint status
true Hide the constraint
false Unhide the constraint

Example
The following example hides Constraint(1) to Constraint(10) so that they will not appear in the generated problem, and then unhide Constraint(5) so that it will appear in the generated problem.

forall (i in 1..10) sethidden(Constraint(i),true)
sethidden(Constraint(5),false)

Further information
This procedure hides (tf=true) or unhide (tf=false) a genctr. A hidden genctr still exists in the model, but will be omitted from the SLP problem which is loaded and solved.

Related topics
ishidden
**setintcontrol**

**Purpose**  
Set the value of an SLP integer control parameter

**Synopsis**  
```plaintext
procedure setintcontrol(Type:string, Param:integer|string, Value:integer)
```

**Arguments**

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
<td>A character string beginning with &quot;L&quot; for the linear problem or &quot;M&quot; for the MIP (integer) problem in MISLP</td>
</tr>
<tr>
<td>Param</td>
<td>The number or name of the required parameter. The number is given in the header file for the corresponding Xpress module (for example, <code>xprs.h</code> or <code>xslp.h</code>). If the name is used, it must be preceded by &quot;XSLP&quot; or &quot;XPRS&quot; as appropriate (for example, &quot;xslp_algorithm&quot; or &quot;xprs lpiterlimit&quot;).</td>
</tr>
<tr>
<td>Value</td>
<td>The value to which the parameter is to be set</td>
</tr>
</tbody>
</table>

**Example**

The following code fragment from a callback within a MISLP optimization sets the value of `defaultalg` in the MIP problem to 2:

```plaintext
setintcontrol("M",8023,2))
```

**Further information**

The `set...control` functions are identical to `setparam` for the linear problem (`Type="L"`). During MISLP, MIP parameters must be set in the MIP problem (`Type="M"`) because the linear problem is solving only subproblems and is not in MIP mode. Note that the required parameter can be specified as an integer or as a name. Specifying as an integer is generally more efficient. The value corresponding to a particular parameter can be found in the header file for the appropriate module (`xprs.h` or `xslp.h`). This function is also useful for setting parameters which have not been defined by name in the Mosel system.

**Related topics**

`getdblcontrol`
**setinitval**

**Purpose**
Provide an initial value for an SLP variable

**Synopsis**

```
procedure setinitval(Var:mpvar, Value:real)
```

**Arguments**
- **Var**       the variable being assigned an initial value
- **Value**    initial value to be assigned to the variable

**Example**
The following code fragment assigns the initial values 1, 1.1, 1.2 etc to the variables \( V(i) \):

```
declarations
  V: array(1..10) of mpvar
end-declarations

forall (i in 1..10) setinitval(V(i),1+(i-1)/10)
```

**Further information**
This function assigns Value to Var as the initial value for the SLP optimization. In general, all SLP variables (threadvars which have nonlinear coefficients, or which are involved in nonlinear coefficients) should be given initial values, because the default values used by Xpress-SLP may not be very good.

**Related topics**
clearinitvals, copy_sol_to_init, SLPDATA("IV")
settype

Purpose
Set the type of a general constraint

Synopsis
procedure settype(c:genctr, type:integer)

Arguments
- c: a general constraint
- type: the type of the constraint. This will be one of:
  - XPRM_CTYPE_LEQ \leq
  - XPRM_CTYPE_EQ =
  - XPRM_CTYPE_GEQ \geq
  - XPRM_CTYPE_UNCONS not constraining or not defined

Example
The following example sets the genctr area(i) to be of type "less than or equal to".

  settype(area(i),XPRM_CTYPE_LEQ)

Related topics
gettype
SLPcalluserfunc

**Purpose**
Call a user function directly from Mosel

**Synopsis**
```plaintext
function SLPcalluserfunc(FuncName:string|FuncNumber:integer,
iArray:array(Ri:range) of real,
FuncInfo:array(RA:range) of integer,
InString:array(RI:range) of string,
OutString:array(RO:range) of string,
dArray:array(Rd:range) of real
[, rArray:array(Rr:range) of real]) :real
```

**Arguments**
- **FuncName**  The name of the user function to be called
- **FuncNumber**  The index of the user function to be called
- **iArray**  A real array containing the input values to the function to be called
- **FuncInfo**  An integer array containing the function and argument information array for the function to be called. This must always be dimensioned at least 1..xslp_funcinfosize.
- **InString**  A string array containing the names of the input arguments for the function to be called
- **OutString**  A string array containing the names of the output arguments for the function to be called
- **dArray**  A real array containing the partial derivative flags for the function to be called
- **rArray**  A real array to hold the values returned from the function to be called

**Return value**
The first (or only) value returned by the function being called

**Example**
The following example initializes data for a call to the user function called MoselArea and then calls the function

```plaintext
declarations
    InArray:array(1..100) of real
    FuncInfo:array(1..xslp_funcinfosize) of integer
    RetArray:array(1..200) of real
    NullString: dynamic array(1..1) of string
    NullReal: dynamic array(1..1) of real
end-declarations

SLPDATA("UF","MoselArea","","DOUBLE,INTEGER","DLL","MyLib")

Count := 5
N := 1
forall (i in 1..Count) do
    InArray(N) := rho(i)
    InArray(N+1) := theta(i)
    N := N+2
end-do

SLPsetuserfuncinfo(FuncInfo,1,Count,1,0,0,0)
Result := SLPcalluserfunc("MoselArea",
    InArray, FuncInfo, NullString,
    NullString, NullReal, RetArray)
```

Functions and Procedures 50 Xpress-Mosel Program Reference Manual (Xpress-SLP Section)
Note the use of empty dynamic arrays where `SLPcalluserfunc` requires an argument but the user function does not.

**Further information**

`SLPcalluserfunc` can take the name or the index of the user function as its first argument. Use of the index (obtained by using `SLPgetindex`) is more efficient when there are repeated calls to the same function.

`SLPcalluserfunc` always has arguments representing all the possible input arguments to the user function. If the user function does not use an argument (it is missing in the `SLPDATA` declaration), then an empty dynamic array of the appropriate type should be used, as in the above example.

If the user function requires the use of the return array argument (`rArray`) then this must be included in the `SLPcalluserfunc` statement. If the user function does not require it, then its presence in the `SLPcalluserfunc` statement is optional. If it is present, it will be filled with the value(s) returned by the function.

If the function returns more than one value, the first one will be used as the return value of `SLPcalluserfunc`.

**Related topics**

`SLPDATA`, `SLPsetuserfuncinfo`
SLPDATA

Purpose
Provide SLP-specific problem information

Further information
See *The SLPDATA Procedure*
SLPexportprob

Purpose
Write the SLP problem to file in extended MPS or text format

Synopsis
procedure SLPexportprob(FileName:string [, Options:string])

Arguments
FileName    the name of the file to be written.
Options     string holding any options required. These are the same ones as defined for the XSLPwriteprob function in the Xpress-SLP Reference Manual. The most common ones are:
l    write the linearized form of the problem (the default is to write the nonlinear problem)
o    write one matrix element on each record (the default is two elements)
t    write in "text" format (the default is to write in Extended MPS format)

Example
The following example writes the nonlinear problem to file MyFile.txt in text format:
   SLPexportprob("MyFile","t")

Further information
The extension .mat (for Extended MPS format) or .txt (for text format) is automatically appended to the given file name.
Note that, where an SLP problem is being created, the standard Mosel procedure exportprob will write out only the linear part of the problem.

Related topics
SLPloadprob
**SLPgetfuncinfo**

**Purpose**
Obtain additional information for a user function

**Synopsis**
function SLPgetfuncinfo(FuncInfo:array(FR:range) of integer, Param: integer) :integer

**Arguments**
- **FuncInfo**
  The function information array for the function. When SLPgetfuncinfo is used within a user function, FuncInfo is normally the second argument to the function.
- **Param**
  The item required. The value is one of the following:
  - `xslpinfo_callerflag` Returns the CallerFlag parameter
  - `xslpinfo_ninput` Returns the number of input variables
  - `xslpinfo_noutput` Returns the number of return variables
  - `xslpinfo_ninstring` Returns the number of items in the input string array
  - `xslpinfo_noutstring` Returns the number of items in the output string array
  - `xslpinfo_funcnum` Returns the number of the function being called
  - `xslpinfo_instance` Returns the instance number of the function being called

**Return value**
the value indicated by Param

**Example**
The following code fragment (from within a Mosel user function) uses SLPgetfuncinfo to find the number of input variables (the number of items actually used in InArray):

```mosel
function MyArea(InArray:array(IR:range) of real, nArray:array(NR:range) of integer): real
...  N := SLPgetfuncinfo(nArray, xslpinfo_ninput)
```

**Related topics**
- SLPcalluserfunc, SLPsetuserfuncinfo
**SLPgetindex**

**Purpose**
Obtain the index of an SLP item from its name

**Synopsis**

```pascal
function SLPgetindex(Name:string, Type:integer) : integer
```

**Arguments**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>The name of the item whose index is to be found</td>
</tr>
<tr>
<td>Type</td>
<td>The type of item required. The following are pre-defined:</td>
</tr>
<tr>
<td>1</td>
<td>Row names</td>
</tr>
<tr>
<td>2</td>
<td>Column names</td>
</tr>
<tr>
<td>xslp_internalfuncnames</td>
<td>Internal (pre-defined) function names (case sensitive)</td>
</tr>
<tr>
<td>xslp_internalfuncnamesnocase</td>
<td>Internal (pre-defined) function names (not case sensitive)</td>
</tr>
<tr>
<td>xslp_userfuncnames</td>
<td>User function names (case sensitive)</td>
</tr>
<tr>
<td>xslp_userfuncnamesnocase</td>
<td>User function names (not case sensitive)</td>
</tr>
<tr>
<td>xslp_xvnames</td>
<td>XV names</td>
</tr>
</tbody>
</table>

There are other types, but they are not usually required for Mosel

**Return value**
The index of the item requested.
Except for row and column names, a value of zero means that the item was not found. For row and column names, a negative value means that the item was not found.

**Example**
The following code fragment gets the index of the user function *MoselArea*. The first form is case-sensitive, and will find the name exactly as given; the second form is not case-sensitive, and will find the first name regardless of the case of the individual letters. The final call finds the name of the internal (pre-defined) function *sin*

```pascal
FuncNum := SLPgetindex("MoselArea",xslp_userfuncnames)
FuncNum := SLPgetindex("MOSELarea",xslp_userfuncnamesnocase)
FuncNum := SLPgetindex("SIN",xslp_internalfuncnamesnocase)
```

**Further information**
Use *SLPgetindex* to obtain the index of a user function which is used in more than one *SLPcalluserfunc* statement, because using the index is more efficient than using the name when it is required more than once.

**Related topics**
*SLPgetname*
**SLPgetname**

**Purpose**
Get the name of an SLP item corresponding to a given index

**Synopsis**

```
function SLPgetname(Type:integer, Index:integer) : string
```

**Arguments**

<table>
<thead>
<tr>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
<td>The type of item required. The full list is in the XSLP Function Reference manual; the following are those most commonly used:</td>
</tr>
<tr>
<td>1</td>
<td>Row names</td>
</tr>
<tr>
<td>2</td>
<td>Column names</td>
</tr>
<tr>
<td>xslp_internalfuncnames</td>
<td>Internal (pre-defined) function names</td>
</tr>
<tr>
<td>xslp_userfuncnames</td>
<td>User function names</td>
</tr>
<tr>
<td>xslp_xvnames</td>
<td>XV names</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Index</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Index</td>
<td>The index number of the item required</td>
</tr>
</tbody>
</table>

**Return value**
The name of the indexed item in the problem of the appropriate type. An empty value means that the requested item was not found.

**Example**
The following code fragment writes the names of the row, column, internal function and user function with index 2:

```plaintext
i := 2
writeln("Row ",i, " is ",
        SLPgetname(1,i))
writeln("Column ",i, " is ",
        SLPgetname(2,i))
writeln("Internal function ",i, " is ",
        SLPgetname(xslp_internalfuncnames,i))
writeln("User function ",i, " is ",
        SLPgetname(xslp_userfuncnames,i))
```

**Further information**
This function is used principally to find the name in the problem of an item referenced by number. It is only useful where the problem is generated using meaningful names (set `xslp_loadnames` to true). Items are referenced by number particularly when they are passed as data to SLP callbacks.

**Related topics**

getline, SLPgetindex
SLPglobal

Purpose
Call the Xpress-SLP mixed integer optimizer (MISLP)

Synopsis
procedure SLPglobal

Example
The following example first calls the Xpress-SLP maximization procedure to solve the linear relaxation and then the mixed integer optimizer to solve the integer problem.

SLPmaximize
SLPglobal

Further information
SLPglobal can only be used after a call to SLPmaximize or SLPminimize to solve the linear relaxation of the problem. If a problem is known to be an integer problem, then SLPmaximize and SLPminimize automatically include the "l" (linear relaxation) option which therefore does not have to be explicitly included.
SLPglobal can be called automatically after the linear relaxation by including the "g" option for SLPmaximize or SLPminimize.

Related topics
SLPmaximize, SLPminimize
SLPloadprob

Purpose
Loads the nonlinear problem into the Xpress-SLP optimizer

Synopsis
procedure SLPloadprob(Name:linctr | gexp [Options:string])

Arguments
Name              name of the objective function
Options           set to "n" to force the use of the linear SLP optimizer even if the objective is quadratic.

Further information
Unless otherwise specified on the call to the optimization function, the linear optimizer will be used to solve all problems except those with a quadratic objective. If Name contains only quadratic (and possibly linear) terms, then the quadratic optimizer will be used by default.

Any objective can be structured into a linear objective function by modelling it in an equality constraint, and transferring the resulting value to the (linear) objective. For example, if OBJECTIVE is the required nonlinear objective, then:
create a free mpvar OBJ
set OBJ = OBJECTIVE
and then do SLPloadprob(OBJ)

An example of this technique is shown in the Xpress-SLP User Guide. If a non-quadratic, nonlinear, objective is provided, then a transfer structure of this type will be created automatically. If the "n" option is included, then the transfer structure will also be provided if the objective is quadratic.

SLPloadprob loads the nonlinear problem into the Xpress optimizer. The linear equivalent (loadprob) will load only the linear part of the problem.

Example
The following examples of the SLPloadprob procedure illustrate the different types of objective which can be defined:

SLPloadprob(OBJ) ! where OBJ is a linctr
SLPloadprob(OBJcol) ! This can be used if OBJcol is a column whose
! activity contains the value of the objective
SLPloadprob(quadexp) ! This can be used if quadexp is a gexp
! containing a quadratic objective
SLPloadprob(nlexp) ! If nlexp is a general (non-quadratic,
! nonlinear) function, additional structure
! will be created to allow solution using
! the linear optimizer

Related topics
SLPexportprob, SLPmaximize, SLPminimize
**SLPlogfile**

**Purpose**
Specify the name of a file to receive any output messages from the Xpress-SLP program

**Synopsis**
procedure SLPlogfile([FileName:string] [, Option:integer])

**Arguments**
- **FileName** the name of the file to be written.
- **Options** integer specifying where the output is written. If **Option** is omitted, the output is written to both the file and to standard output.
  - 0 only to the file specified
  - 1 to the file specified and to the standard output

**Example**
The following example sets the SLP log file to be `MyLog.log` and specifies that the output should be written to the file and displayed on the program’s standard output.

```
SLPlogfile("MyLog.log",1)
```

**Further information**
The messages include error, warning and information messages as well as any other data which is passed to the Xpress-SLP message routine (for example, from a user function).

**Related topics**
- xslp_verbose
SLPmaximize, SLPminimize

**Purpose**
Maximize or minimize the current SLP problem

**Synopsis**

```plaintext
procedure SLPmaximise([Objective:linctr|genctr] [Options:string])
procedure SLPminimise([Objective:linctr|genctr] [Options:string])
procedure SLPmaximize([Objective:linctr|genctr] [Options:string])
procedure SLPminimize([Objective:linctr|genctr] [Options:string])
```

**Arguments**
- **Objective**
  The name of the objective function. See SLPloadprob for more details.
- **Options**
  String containing any options as used in the Xpress-SLP or Xpress-Optimizer maximize or minimize procedures. In addition, the following are specific to SLP:
  - `g`
    Perform SLPglobal immediately after a successful optimization
  - `n`
    Use the linear optimizer even if the objective is quadratic

**Example**
The following examples load a nonlinear problem and then maximize it using the Newton barrier code in the Xpress-SLP optimizer:

1. Separate load and solve
   ```plaintext
   SLPloadprob(OBJ)
   SLPmaximize("b")
   ```

2. Combined load and solve
   ```plaintext
   SLPmaximize(OBJ,"b")
   ```

**Further information**
These functions call Xpress-SLP to optimize the problem previously loaded by SLPloadprob. If Objective is omitted, SLPloadprob must be called explicitly. If SLPloadprob is called explicitly, then parameter changes and other functions and procedures can be called after the problem is loaded and before it is optimized. Otherwise, all settings have to be made before the call to optimize.

Both the English and American spellings are accepted.

Note that the linear procedures maximize and minimize will not solve a nonlinear problem. Instead, they will solve only the linear part of the nonlinear problem. This can be useful because if the linear part of the problem is infeasible, there is no point in trying to solve the whole problem.

**Related topics**
SLPglobal, SLPloadprob
SLPprintmemory

Purpose
Print memory usage for Xpress-SLP and the mmxslp module.

Synopsis
procedure SLPprintmemory

Example
The following example loads a nonlinear problem and then reports on the memory usage.

SLPloadprob(OBJ)
setparam("xslp_verbose",true)
SLPprintmemory

Further information
Because the output from the procedure is information, it will be suppressed if xslp_verbose is FALSE.

The output shows the current memory usage and the values of the memory parameters (XSLP_MEM... and XPRM_MEM...). Memory can be used more efficiently by presetting some of these values when the problem sizes are known in advance.
SLPscaling

Purpose
Call the Xpress-SLP scaling function to report on the range of values in the problem

Synopsis
procedure SLPscaling

Example
The following example loads a nonlinear problem and then reports on the ranges of values in
the rows and columns of the problem, and the largest and smallest elements.

SLPloadprob(OBJ)
setparam("xslp_verbose", true)
SLPscaling
SLPsetcallback

Purpose
Set a Mosel function to be used as an SLP callback

Synopsis
procedure SLPsetcallback(cbNumber:integer, cbFunc:string)

Arguments
- cbNumber the number of the callback. This can be one of the Xpress-SLP constants named to match the callbacks.
- cbFunc string holding the name of the Mosel function or procedure to receive the callback. An empty string removes any previously-defined callback of this type.

Example
The following is a small example of a callback function. The name of the function (myIterStart) is used in the SLPsetcallback procedure.

```mosel
function myIterStart : integer
    writeln("SLP iteration ", getparam("xslp_iter")
    returned := 0
end-function
...
SLPsetcallback(2, "myIterStart")
SLPsetcallback(xslp_iterstart, "myIterStart")
```

The first usage of SLPsetcallback sets SLP callback number 2 to be the Mosel function myIterStart in the current model. The second usage does the same thing, but is more readable because it uses the constant xslp_iterstart instead of the number 2. In addition, the second example is more robust because it will continue to work even if the value of the parameter changes.

The next example shows a callback function to be used within global optimization. It tests the value of the parameter xslp_errorcosts which is the total feasibility error cost of the solution. If it is significant (>10) then the node is declared infeasible (the function returns a nonzero value). The name of the function (MyNodeCB) is used in the SLPsetcallback procedure.

```mosel
function MyNodeCB(feas:integer) : integer
    if getparams("xslp_errorcosts") > 10 then
        returned := 1
        writeln("Terminated on feasibility error costs")
    else
        returned := 0
    end-function
...
SLPsetcallback(xslp_slpnode, "MyNodeCB")
```

Further information
This function defines a Mosel function or procedure to be the callback function at a defined point in the Xpress-SLP optimization. The callback type is given in the first argument, either as a number or — better — by using one of the defined Xpress-SLP constants. The name of the function or procedure is given as a string in the second argument.

If the function cannot be identified as a Mosel function, it will be checked against the user function list. If it is found there, then the user function will be used as the callback. This extension can only be used if the user function has the correct linkage to be a callback (an Excel spreadsheet, for example, would not qualify). It is primarily intended to be used where the callback is in a compiled external library.

The function itself must behave like the corresponding Xpress-SLP callback function: in general, this means that the functions are declared as of type integer; the normal return value is zero.
and the abnormal return value is nonzero. See the Xpress-SLP Reference Manual for more details. Most of the Mosel callback functions do not take arguments. The exceptions are:

- `XSLPcascadevar(VarNum:integer) : integer`
- `XSLPitervar(VarNum:integer) : integer`
  which have one integer argument holding the number of the variable being processed.

- `XSLPoptnode(feas:integer) : integer`
- `XSLPprenode(feas:integer) : integer`
- `XSLPslpnode(feas:integer) : integer`
  which have one integer argument holding the current value of the feasibility flag. The value of the flag can be checked, but cannot be changed. It is set to 1 if the callback function returns a nonzero value.

- `XSLPslpmessage(Message:string, MsgLen:integer, MsgType:integer)`
  which has arguments holding the message, message length and message type. This should be defined as a procedure rather than as a function, because there is no return value.

Note that if this callback is used, then the normal SLP message handling by Mosel will be overridden.
SLPsetprobstat

Purpose
This function sets the problem status parameters after an optimization.

Synopsis
procedure SLPsetprobstat

Example
The following example sets the problem status before obtaining the solution value for a variable.

    SLPsetprobstat
    x := getact(col1)

Further information
The problem status is set automatically after a problem has been solved. If solution information is required while the optimization is in progress (for example, in a callback), then it may be necessary to set the problem status. If the status is not set, then solution values will be returned as zero (or "unavailable").
SLPsetuserfuncaddress

Purpose
Set the address of an external user function

Synopsis
procedure SLPsetuserfuncaddress(FuncIndex:integer, Address:integer|string)

Arguments
FuncIndex The index of the user function (as obtained by SLPgetindex)
Address The address of the function, given as an integer or (in character form) as a string. The integer form cannot be used in a 64-bit environment.

Example
The following code fragment sets the address of the user function ExtFunc to ExtAddress. Assuming that Mosel has been called from Visual Basic and that the address of the Visual Basic function has been passed to Mosel in ExtAddress, any calls from Mosel to the user function ExtFunc will be to the Visual Basic function.

SLPDATA("UF", "ExtFunc", ",", "DOUBLE, INTEGER", "VB", ",", ")
FuncNum := SLPgetindex("ExtFunc", xslp_funcnamesnocase)
SLPsetuserfuncaddress(FuncNum, ExtAddress)

Further information
The integer form of the address is provided for compatibility with earlier versions. The string form is always to be preferred.

Related topics
SLPcalluserfunc, SLPgetindex
SLPsetuserfuncinfo

Purpose
Set the function information array before calling a user function

Synopsis
procedure SLPsetuserfuncinfo(FuncInfo:array(AR:range) of integer,
   CallerFlag:integer, nInput:integer, nOutput:integer,
   nDelta:integer, nInString:integer, nOutString:integer)

Arguments
FuncInfo  The argument information array to be set up. This must be dimensioned at least
   xslp_funcinfosize
CallerFlag  an integer value which can be used for any purpose in the calling and called
   function
nInput  The number of values in the array of input arguments for the function to be
called
nOutput  The number of values required from the function to be called
nDelta  The number of sets of partial derivatives to be calculated by the function to be
called
nInString  The number of strings in the input string argument (argument 4 to the function
   being called)
nOutString  The number of strings in the output string argument (argument 5 to the func-
tion being called)

Example
The following code fragment caller flag to 99 (this can be picked up and interpreted by the
called function), the number of input values to 5, the number of return values required to 1,
and sets the remaining values to zero.

   declarations
   nArray:array(1..xslp_funcinfosize) of integer
   end-declarations

   SLPsetuserfuncinfo(nArray,99,5,1,0,0,0)

Further information
This function is used in conjunction with SLPcalluserfunc to allow direct calls to external
functions from within Mosel. If CallerFlag is set to a nonzero value, the calling program
can identify that it has been called directly rather than from within an optimization, when
CallerFlag is always zero.

Related topics
SLPgetfuncinfo, SLPcalluserfunc
SLPtime

Purpose
Get a time reference value

Synopsis
function SLPtime :real

Return value
An elapsed time reference in seconds (to two or three decimal places, depending on the accuracy of the system clock.

Example
The following code fragment prints the elapsed time for loading an SLP problem:

```
declarations
    StartTime: real
end-declarations
StartTime := SLPtime
SLPloadprob(OBJ)
writeln("Time to load problem = ", SLPtime-StartTime)
```

Further information
SLPtime returns the current value of the system clock, and is not initialized to zero at the start of model execution. It is therefore necessary to calculate all elapsed times by difference.
SLPvalidate

Purpose
Validate an SLP solution

Synopsis
procedure SLPvalidate

Example
The following code fragment optimizes an SLP problem and then validates the solution:

```mosel
SLPmaximize
SLPvalidate
```

Further information
SLPvalidate is equivalent to the Xpress-SLP console command VALIDATE or the library function XSLPvalidate. It calculates the values of the constraints using the solution values of the variables, and reports any which are outside the validation feasibility tolerances. Because SLP uses linear approximations, the approximation may be feasible but the nonlinear problem may be slightly infeasible at the same set of solution values. The validation, and the related validation indices (xslp_validationindex_a and xslp_validationindex_r) give a measure of the practical feasibility of the solution.
tolsetstatus

Purpose
Return the bitmap status of an SLP tolerance set

Synopsis
function tolsetstatus(TolSet:integer) :integer

Argument
TolSet the number of the tolerance set whose status is to be obtained

Return value
The bitmap status of the tolerance, indicating which of the tolerances are activated. For details of these, see the Xpress-SLP Reference Manual.

Example
The following example obtains the values and status for tolerance set number 1 and prints the tolerance values which have been set:

declarations
Tols: array(1..9) of real
Status: integer
end-declarations
gettolset(1,Tols)
Status := tolsetstatus(1)
forall (i in 1..9) do
  if bittest(Status,1) <> 0 then
    writeln("Tolerance ",i," = ",Tol(i))
  end-if
  Status := Status / 2
end-do

Related topics
chgtolset, gettolset
**XSLPcommand**

**Purpose**
Execute an SLP console command and optionally return a value

**Synopsis**

```plaintext
procedure XSLPcommand(cmd1:string [,cmd2:real|string] [,cmd3:real|string])
function cXSLPcommand(cmd1:string [,cmd2:real|string] [,cmd3:real|string]) :string
function iXSLPcommand(cmd1:string [,cmd2:real|string] [,cmd3:real|string]) :integer
function rXSLPcommand(cmd1:string [,cmd2:real|string] [,cmd3:real|string]) :real
```

**Arguments**
- `cmd1` String containing the first part or all of the SLP console command
- `cmd2` Optional string or value to be appended to `cmd1`
- `cmd3` Optional string or value to be appended to `cmd2`

**Return value**
- `XSLPcommand` does not return a value.
- The other forms of the statement return the value of the command. A function will generally return zero or an empty string if the return value of the command is of a different type from that of the function.

**Example**
The following code fragment uses the SLP console commands to retrieve the value of `presolve` and reverse its sense:

```plaintext
declarations
  Presolve: integer
end-declarations
Presolve := iXSLPcommand("presolve")
writeln("Presolve was ",Presolve)
XSLPcommand("presolve = ",(1.0-Presolve))
Presolve := iXSLPcommand("presolve")
writeln("Presolve is now ",Presolve)
```

**Further information**
These statements use the `mmxcmd` module, which must be included in a `uses` statement in the model.
XVitem

Purpose
Set the values for an xvitem object

Synopsis
function XVitem([Name:mpvar|XV], [sName:string],
    [Value:real|linctr|mpvar|gexp|XV]) :xvitem

Arguments
Name  an object of type mpvar or XV. An mpvar is used when the item takes the value of the variable. An XV is used to include all the defined elements of the XV at this point in the array. Name must be omitted when the item is a constant or more complex expression
sName  character string containing the name of the item as known to the function. This is required only when the xvitem is part of an XV which is an argument to a function, and the function receives its arguments by name rather than by position.
Value  real constant, mpvar, linear expression, general expression or an XV. These can be used when the xvitem is not simply an mpvar. Using an XV or mpvar as the third argument is equivalent to using it as the first argument. Value must be omitted if Name is used.

Return value
xvitem data of the appropriate form.

Example
The following code fragment sets 3 elements of the XV XX and the first element of the XV XY.

    declarations
    XX, XY:array(1..3) of xvitem
    C, I: mpvar
    end-declarations
    XX(1) := XVitem(C, "Capital")
    XX(2) := XVitem("Rate", I+2)
    XX(3) := XVitem("Term", 3.5)
    XY(1) := XVitem(XX)

Further information
Name and Value are mutually exclusive. Exactly one of these must appear as an argument.
All elements of an XV must be initialized either by using the function XVitem or by direct assignment of another xvitem.
Chapter 5

The SLPDATA Procedure

SLPDATA is used to provide most of the information specific to SLP models. The type of information is indicated in the first parameter to the procedure. The meaning of the remaining parameters depends on the type of information being provided.
SLPDATA(“DR”)  

Purpose  
Assign a constraint to be the determining row for a variable  

Synopsis  
procedure SLPDATA(DR:string, Var:mpvar, Row:linctr|genctr)  

Arguments  
DR string containing "DR", indicating the SLPDATA type  
Var the variable being assigned a determining row  
Row the linear (linctr) or general (genctr) constraint which acts as the determining row for the variable  

Example  
The following code fragment assigns the constraints RVPq(P) to be the determining rows of the variables RVP(P)  

declarations  
RVP, RVI:array(Prod) of mpvar  
RVPq:array(Prod) of genctr  
end-declarations  

forall (P in Prod) RVPq(P):= RVP(P) = RVI(i)^1.25  
forall (P in Prod) SLPDATA("DR",RVP(P),RVPq(P))  

Further information  
In the example shown, there is an implicit assumption that in some sense RVP is calculated from RVI rather than the other way around. Therefore, for a given value of RVI there is a corresponding value of RVP. However, the linear approximation to $RVI^{1.25}$ will not generally be accurate, and so it is beneficial to be able to recalculate a correct value for RVP from the actual value of RVI, using the formula rather than the linear approximation. This is the purpose of the determining row which allows the SLP optimizer to recalculate values which depend in a nonlinear way on other values.
SLPDATA(“EC”)  

Purpose  
Defines a constraint as "enforced"  

Synopsis  
procedure SLPDATA(EC:string, Row:linctr|genctr)  

Arguments  
EC string containing "EC", indicating the SLPDATA type  
Row the linear (linctr) or general (genctr) constraint which is to be enforced  

Example  
The following code fragment assigns the constraints RVPq(P) to be enforced  

```plaintext
declarations  
    RVP, RVI:array(Prod) of mpvar  
    RVPq:array(Prod) of genctr  
end-declarations  

forall (P in Prod) RVPq(P):= RVP(P) = RVI(i)^1.25  
forall (P in Prod) SLPDATA("EC",RVP(P),RVPq(P))  
```

Further information  
The standard augmentation procedure (which creates a linear approximation from a nonlinear problem) adds penalty vectors to all nonlinear constraints so that the linear approximation can be violated at a cost, even though the original nonlinear constraint itself might still be satisfied. In some cases, such a violation is undesirable and can cause difficulties in obtaining a feasible solution to the original problem. An enforced constraint does not have penalty vectors and the linear approximation must always be satisfied in any feasible solution.
**SLPDATA(“IV”)**

**Purpose**
Provide an initial value for an SLP variable

**Synopsis**

```
procedure SLPDATA(IV:string, Var:mpvar, Value:real)
```

**Arguments**

- **IV** string containing "IV" indicating the SLPDATA type
- **Var** the variable being assigned an initial value
- **Value** initial value to be assigned to the variable

**Example**
The following code fragment assigns the initial values 1, 1.1, 1.2 etc to the variables V(i):

```
declarations
  V: array(1..10) of mpvar
end-declarations

forall (i in 1..10) SLPDATA("IV", V(i), 1+(i-1)/10)
```

**Further information**
This function assigns Value to Var as the initial value for the SLP optimization. In general, all SLP variables (mpvars which have nonlinear coefficients, or which are involved in nonlinear coefficients) should be given initial values, because the default values used by Xpress-SLP may not be very good.
SLPDATA("SB")

Purpose
Assign an initial value for the step bound of a variable

Synopsis
procedure SLPDATA(SB:string, Var:mpvar, Value:real)

Arguments
SB string containing "SB" indicating the SLPDATA type
Var the variable being assigned an initial step bound
Value initial value to be assigned to the step bound

Example
The following code fragment assigns initial step bounds of 1.2 to all the variables RVP(P):

```mosel
declarations
  RVP, RVI:array(Prod) of mpvar
  RVPq:array(Prod) of genctr
end-declarations

forall (P in Prod) RVPq(P) := RVP(P) = RVI(i)^1.25
forall (P in Prod) SLPDATA("SB",RVP(P),1.2)
```

Further information
Step bounds are used in SLP optimization to restrict the amount by which the value of a variable can change between successive SLP iterations. A smaller value means that the linearization is more accurate but that it takes more iterations to change a value by a large amount; a larger value means that it is easier for the value of the variable to change but the linear approximations are less accurate.

The SLP optimizer will estimate step bounds for all variables once optimization has begun. If step bounds are required from the beginning, or if the estimates are not appropriate, then initial values should be provided.
SLPDATA("TOLSET")

Purpose
Assign a special SLP tolerance set for a variable

Synopsis
procedure SLPDATA(TOLSET:string, Var:mpvar, TolSet:integer)

Arguments
- TOLSET: string containing "TOLSET", indicating the SLPDATA type
- Var: the variable being assigned a special tolerance set
- TolSet: number of the tolerance set assigned to the variable

Example
The following code fragment assigns tolerance set number 3 to the variables RVP(P):

```
declarations
  RVP: array(Prod) of mpvar
end-declarations

forall (P in Prod) SLPDATA("TOLSET", RVP(P), 3)
```

Further information
Tolerance sets are created and modified by chgtolset.
The current tolerance set for a variable is obtained from getvartolset.
The default tolerance set is number zero; this uses the system-defined values which are set
using the control parameters xslp_atol_a etc.
**SLPDATA("UF")**

**Purpose**
Define a user function

**Synopsis**

```
procedure SLPDATA(UF:string, Name:string, fName:string, ArgList:string,
                 fType:string [,Param1:string] [,Param2:string] [,Param3:string])
```

**Arguments**

- **UF** string containing "UF", indicating the SLPDATA type
- **Name** name of the function (as used within a Func expression)
- **fName** name of the function to be used when it is called. This may be different from Name (for example, it may be decorated or have a special prefix). fName can be left blank if it is the same as Name.
- **ArgList** list of the argument types to the function, as described in the Xpress-SLP Reference Manual. These must match exactly the declaration of the function.
- **fType** the function type as described in the Xpress-SLP Reference Manual. Possible types are:
  - **DLL** Compiled in an external library or DLL
  - **XLS** Excel spreadsheet with optional macro
  - **XLF** Excel macro
  - **VB** Visual basic
  - **COM** COM
  - **MOSEL** Mosel function in the current model
- **Param1-3** optional strings giving additional parameter information as required by the particular function type. Details are in the Xpress-SLP Reference Manual.

**Example**

The following defines the user function PolyArea as an Excel spreadsheet (type XLS) on Sheet1 of C:\Progs\Polycalc.xls (notice the use of the single quote to surround the string, because "\\ has a special meaning). It takes two arguments, both as arrays of type VARIANT: the list of input values and the Function Information Array.

```
SLPDATA("UF", "PolyArea", "", 
        "VARIANT,VARIANT", "XLS",
        'C:\Progs\Polycalc.xls', 'Sheet1')
```

The following defines the user function myUF as a C or Fortran routine in an external library (type DLL) in file myProgLib. It takes two arguments: the list of input values as an array of type DOUBLE (Mosel type real) and the Function Information Array as an array of type INTEGER.

```
SLPDATA("UF", "myUF", "DOUBLE,INTEGER", 
        ",", "DLL", "myProgLib")
```

The final example is similar, except that the function will be used as an external callback function. In such a case, the linkage is determined by the type of callback, and is not provided through the SLPDATA procedure.

```
SLPDATA("UF", "myCB", "", 
        ",", "DLL", "myProgLib")
```

The callback itself can then be defined using the SLPsetcallback procedure.

```
SLPsetcallback(xslp_iterstart, "myCB")
```
Further information

This procedure defines a user function for use in the Xpress-SLP optimization process. Once a function has been defined, it can be used in `Func` expressions. Because there is no "forward declaration" of user functions, you must put any `SLPDATA` procedures which define user functions at the beginning of the model.

The `SLPDATA` procedure is also used to define external functions for use as callbacks. In this form, the `ArgList` string is empty, because the arguments required are determined by the nature of the callback. The function type and any other appropriate parameters (for example, the location of the function) are still required.
SLPDATA("XV")

Purpose
Define an array of xvitems as an XV

Synopsis
procedure SLPDATA(XV:string, XArray:array(XR) of xvitem)

Arguments
XV  string containing "XV", indicating the SLPDATA type
XArray an array of xvitems to be defined as an XV

Example
The following example declares two arrays of xvitem:
- rTheta is a 1-dimensional array, which will correspond to a single XV with (in this case) up to 50 items;
- XXA is a 2-dimensional array, which will correspond to an array of nBox XVs each with up to (in this case) 10 items.

The first two SLPDATA statements define two user functions. The remaining two SLPDATA statements define rTheta and XXA as XVs.

The XVs are used in the user functions through the general-purpose Func function, which allows an XV to be used in place of an array of values. rTheta can appear directly because it is 1-dimensional. The relevant XV from the array XXA is referenced using the first item in the appropriate part of the array so that XXA(B,1), being the first item in XXA(B), refers to the Bth XV in XXA.

declarations
  rTheta: array (1..50) of xvitem
  XXA: array(1..nBox,1..10) of xvitem
end-declarations
SLPDATA("UF", "Area", "DOUBLE,INTEGER", "", "DLL", "myProgLib")
SLPDATA("UF", "ValueCalc", "DOUBLE,INTEGER", "", "DLL", "myProgLib")
SLPDATA("XV", rTheta)
SLPDATA("XV", XXA)

... Obj = Func("Area",rTheta)
forall (B in 1..nBox) Value(B) = Func("ValueCalc",XXA(B,1))

Further information
An array of xvitems must be declared as an XV before it can be used in a statement where an XV is required — for example, in a Func reference.

An array of XVs is declared as an array of xvitem objects with one extra dimension (since each XV is itself an array of xvitem objects). Thus, a 1-dimensional array of XVs with dimension 1..nBox is described as a 2-dimensional array of xvitem objects, with its first dimension 1..nBox and the second dimension large enough for the largest of the XVs in the array. When an XV in an array of XVs is used, it is referred to by using its first xvitem object. So, in the example above, to refer to XXA number B, use XXA(B,1).
User functions and callbacks are the two mechanisms which allow the user program to control the way in which the optimization proceeds. They have different purposes and are not interchangeable.

A callback is called at a specific point in the SLP optimization process (for example, at the start of each SLP iteration). It has full access to all the problem data and can, in principle, change the values of any items — although not all such changes will necessarily be acted upon immediately or at all. Callbacks are implemented in Mosel as functions or procedures and are registered with the optimizer by using the setcallback or SLPsetcallback procedures.

A user function is essentially the same as any other mathematical function, used in a formula to calculate the current value of a coefficient. The function is called when a new value is needed; for efficiency, user functions are not usually called if the value is already known (for example, when the function arguments are the same as on the previous call). Therefore, there is no guarantee that a user function will be called at any specific point in the optimization procedure or at all. User functions are implemented in Mosel as functions with a pre-defined pattern of arguments; they are registered with the optimizer by using the SLPDATA("UF") procedure and included in constraints by using the Func function.

A user function appears in formulae with one or more arguments (and possibly with a return value reference if the function returns an array of values). These arguments can be any mixture of constants, mpvars, formulae or XVs. Regardless of how the arguments appear, they are converted into a single array of values for input to the implementation of the user function, so that the user function itself always deals with an array of values and returns a value or an array of values. The user function itself can be written in a number of different languages — for example: in Mosel as a real function; in Excel as a set of spreadsheet formulae or in a macro; in C or Fortran, compiled into a library. The SLPDATA("UF") procedure describes the linkage and location of the function so that it can be called as necessary.

In skeleton form, the components of a user function implementation are:

1. The user function used in formulae.
   For example, if Product is a user function:
   
   ```
   declarations
   X, Y: mpvar
   Z: array(1..n) of xvitem
   R: real
   end-declarations
   Product(X,Y,R) >= 10
   Product(X^R,Y,X+2,X-2) <= 20
   Product(X,Z) >= 100
   ``

   The first usage shows the function called with 3 arguments: 2 mpvars and a constant; the second usage shows the function called with 4 arguments including linear and non-linear formulae; the third usage shows the function called with an mpvar and an XV. An XV (extended variable array) is equivalent to an array of mpvars, formulae or constants.
2. The implementation of the user function.
   In Mosel, a simple function takes an array of values and an integer (with the number of
   values) as its arguments and returns a single real value:

   ```
   function Product(Values:array(vRange:range) of real, N:integer):real
       returned := 1
       forall (i in 1..N) returned := returned * Values(vRange(i))
   end-function
   ```

   This function works independently of the number of arguments used in the original for-
   mula and returns the product of the first $N$ values in the array.

3. Declaration of the function linkage using SLPDATA.
   All the information necessary for calling the function is contained in this statement:
   the name of the function (both within the Mosel program and, if necessary, the actual
   name used in the implementation);
   the type of function (C, Fortran, Mosel, etc) which determines the linkage mechanism;
   where the function is (user library, Mosel program, spreadsheet, etc);
   what data the program expects (values, names) and what it returns (multiple values,
   derivatives).

   ```
   SLPDATA("UF","Product", ",", "DOUBLE,INTEGER",
   "Mosel", "Model", "BigArray")
   ```

   This declares the user function Product as a Mosel function in the model Model which
   takes input values and the number of values and returns a single result. BigArray is
   additional information specific to Mosel functions, and is the name of an array used to
   transfer data to and from the program.
User function (simple form)

Purpose
In many cases, a user function takes one or more parameters and returns a single value. Such a function does not need the full power of the user function structure as described in User function (general form) (below), and it is possible to simplify the function definition.

Synopsis
function fName (aName:array(aRange:range) of real, nName:integer) :real

Arguments
aName an array of type real containing the actual values of the parameters to the function
nName an integer containing the actual number of parameters

Return value
The calculated value of the function

Example
The following example of a user function calculates the product of an array of values:

```mosel
function Product(Values:array(vRange:range) of real, nValue:integer) :real
declarations
Value: real
end-declarations
Value := 1
forall (i in 1..nValue) do
  Value := Value * Values(vRange(i))
end-do
returned := Value
end-function
```

Notice that the function uses nValue rather than vRange to step through the array. This is because not all the values in the array will necessarily be set or used on each function call.

Related topics
User function (general form), Func
User function (general form)

Purpose
The general form of the user function includes possibilities for calculating and returning an array of values, including partial derivatives if appropriate. Additional features include optional information about the calling program.

Synopsis
function fName (aName:array(aRange:range) of real,
nName:array(nRange:range) of integer
[,iName:array(iRange:range) of string]
[,oName:array(oRange:range) of string]
[,dName:array(dRange:range) of string]
[,rName:array(rRange:range) of real]) : real

Arguments
aName an array of type real containing the actual values of the parameters to the function
nName an array of type integer containing the function information array
iName an array of type string containing the array of input names
oName an array of type string containing the array of output names
dName an array of type real containing the array of partial derivative flags
rName an array of type real to contain the array of results from the function

Example
The following is an example of a general Mosel user function which calculates the product of a set of numbers and also the partial derivatives with respect to each of them. It takes four parameters: the input values, the function information array, the array of partial derivative flags, and an array to hold the return values. The function itself returns zero for success (and would return nonzero for failure).

function Product(Values:array(vRange:range) of real,
FuncInfo:array(fRange:range) of integer,
Deltas:array(dRange:range) of real,
Return:array(rRange:range) of real) :real
nValue := FuncInfo(fRange(1))
declarations
Value: real
Derivatives: array(1..nValue) of real
end-declarations
Value := 1
forall (i in 1..nValue) Derivatives(i) := 1
forall (i in 1..nValue) do
  Value := Value * Values(vRange(i))
  forall (j in 1..nValue | j <> i)
    Derivatives(j) := Derivatives(j)* Values(vRange(i))
end-do
Return(rRange(1)) := Value
j := 1
forall (i in 1..nValue) do
  if Deltas(dRange(i)) <> 0 then
    j += 1
    Return(rRange(j)) := Derivatives(i)
  end-if
end-do
returned := 0
end-function
The number of input values is given in the first item in the FuncInfo array. For ease of explanation in this case, the partial derivatives are calculated for all the input variables (that is, for each item in the Values array). Deltas(i) is zero if a derivative is not required for the i^{th} item, and nonzero if a derivative is required. This is tested in the final loop, when the Derivatives array is packed into the Return array.

The function is declared by using the SLPDATA("UF") procedure and can then be called using the Func wrapper as in the following example:

```mosel
declarations
  X,Y: mpvar
  Values, Deltas, Results: array(1..100) of real
  FuncInfo: array(1..xslp_funcinfosize) of integer
  NullStr: dynamic array(1..1) of string
  XArray: array(1..6) of string
end-declarations
XArray := [
  "Values", "FuncInfo", "NullStr", "NullStr", "Deltas", "Results"]
SLPDATA("UF", "Product", ",", ",DOUBLE,INTEGER,,,DOUBLE,DOUBLE",
  "Mosel", "Model", "XArray")
Func("Product", X,Y) <= 10
```

The presence of the Deltas argument in the function definition defines the function as calculating its own derivatives. There is no change in the way the function is written in constraints, and the system will decide when derivatives are required.

Further information

The order and purpose of the arguments to a Mosel user function follow the standard layout for Xpress-SLP user functions as described in the Xpress-SLP reference manual.

- **aName**: The actual values of the parameters to the function are passed in the array aName.
- **nName**: This holds various information items about the function and its arguments. In particular:
  - **nName(1)** is the number of parameters (the actual number of values in the array aName).
  - **nName(2)** is the number of return values required. This is only important if the function returns multiple values and the actual number or order required may vary.
  - **nName(3)** is the number of sets of partial derivatives required. If the function is declared as providing its own derivatives, then **nName(3)** is the number required on this call to the function, which may be zero.
- **iName**: This is an optional argument, which is used if the function receives values by name rather than by position. If it is used, then the names of the arguments are provided in the same order as the values.
- **oName**: This is an optional argument, which is used if the function returns multiple values by name rather than by position. If it is used, then the names of the values are provided in the order in which the values are required.
- **dName**: This argument is only used when the function is declared as providing its own partial derivatives. If it is used, it will have one entry for each item in aName. The value in dName is nonzero if a partial derivative is required for the corresponding item in aName, and zero otherwise. A nonzero value can be used as a basis for numerical derivatives if required. The number of nonzero entries in dName is in nName(3).
- **rName**: This argument is normally used when the function returns multiple values, either because it is a multi-valued function, or because it calculates its own derivatives.
On return, `rName` must contain the required values. The return value must be 0 if the function has completed successfully, or a positive value (≥ 1) if it failed.
**Func (General SLP function)**

**Purpose**

*Func* provides a uniform syntax for implementing functions (particularly user functions) within an SLP problem.

**Synopsis**

```
Func (fName:string [, arg1:real|mpvar|linctr|XV|gexp [, arg2:real|mpvar|linctr|XV|gexp]] [, rIndex:string}) :gexp
```

**Arguments**

- **fName** (string) containing the name of the function being evaluated. This can be an Xpress-SLP internal function or a user-defined function
- **arg1** (optional) first argument to the function. This can be of any type (real, mpvar or expression) acceptable to the function. Arguments of type XV are always acceptable, provided the items in the XV are of the correct type
- **arg2** (optional) second argument to the function. The requirements are the same as for arg1
- **rIndex** (optional) string indicating the index or name of the return value from the function. This is only required when the function returns an array of values

**Return value**

For most purposes, *Func* can be regarded as returning a *gexp* which contains the requested evaluation of the function.

Strictly speaking, *Func* returns a *gexp* describing how to call the specified function, and the evaluation of the function then returns a *real* value.

**Example**

The following code fragment shows two user functions:

- **mySin** takes one input value and returns its sine
- **myArea** takes pairs of values in (r,theta) co-ordinates and calculates the area of the polygon whose vertices are the origin and these points in order.

```plaintext
function mySin(Values:array(vRange:range) of real, nValue:integer) :real
    returned:= sin(Values(vRange(1)))
end-function

function myArea (Values:array(vRange:range) of real, nValue:integer) :real
    declarations
        r1,r2: real ! distances
        t1,t2: real ! angles
        area: real
    end-declarations
    area := 0
    forall (n in 1..nValue-3) do
        r1 := Values(vRange(n))
        t1 := Values(vRange(n+1))
        r2 := Values(vRange(n+2))
        t2 := Values(vRange(n+3))
        area += 0.5*r1*r2*sin(t2-t1)
    end-do
    returned := area
end-function
```

These functions can be used in formulae within the model. Formulae like
declarations
A:array(1..n) of real
V:real
end-declarations
V := mySin(A,1)
V := myArea(A,6)

are ordinary instances of a Mosel function, and will calculate a value based on the values in A
at the time the function is encountered. If we want to include the functions inside a constraint,
with values taken from mpvars, then the function needs to be wrapped so that its formula is
retained. Func is used for this purpose as the following examples show:

declarations
A:array(1..n) of xvitem
R,Theta:array(1..m) of mpvar
X, Obj:mpvar
V:real
BigArray:array(1..n) of real
end-declarations
SLPDATA("UF","mySin","","DOUBLE,INTEGER","MOSEL","Model","BigArray")
SLPDATA("UF","myArea","","DOUBLE,INTEGER","MOSEL","Model","BigArray")
cos(X) <= 0.9
Func("cos",X) <= 0.9
Func("mySin",X) >= 0.5
forall (i in 1..m) do
  A(i*2-1) := R(i)
  A(i*2) := Theta(i)
end-do
Obj = Func("myArea",A)

The constraint
\[ \cos(X) \leq 0.9 \]
is interpreted as a formula rather than a value because (a) the argument \(X\) is not a constant
and (b) \(\cos\) is recognized as an SLP intrinsic function.

The first usage of Func is effectively the constraint
\[ \cos(X) \leq 0.9 \]
that is, it is the same as the previous constraint. Func is not required because \(\cos\) is already
known as an SLP intrinsic function.

The second usage of Func is effectively the constraint
\[ \sin(X) \geq 0.5 \]
Func is necessary here, because \(\text{mySin}\) is not known as an SLP function when the model is compiled
(it is defined at execution time by the SLPDATA procedure). Although \(\text{mySin}\) is defined as
receiving an array of values, there is no need to supply an array in the Func implementation.

The third usage sets the variable \(\text{Obj}\) equal to the area of the polygon whose vertices are
defined by the pairs of values \((R(i),\Theta(i))\). Because there are more than two values
being passed to the function, the information is passed using an XV (array of xvitems) which
can hold any number of items of any type.
**Callbacks**

**Purpose**
A callback is a user-specified function called at a specific point or points in the SLP optimization procedure, and which allows the user to intervene in the optimization process.

**Synopsis**
1. procedure CbMessage(Message:string, MsgLen:integer, MsgType:integer)
2. function CbType1:integer
3. function CbType2(VarNum:integer):integer
4. function CbType3(Flag:integer):integer

**Arguments**
- **Message**: Message text to be printed
- **MsgLen**: Length of message text
- **MsgType**: Type of message
  - 4 Error
  - 3 Warning
  - 0,1,2 Information
  - negative Program is finishing
- **VarNum**: Number of the variable being processed
- **Flag**: Feasibility flag

**Return value**
The procedure form of the callback is only used for `xslp_slpmessage`. All other forms return a value to the calling program.

- 0 Normal
- 1 Mark node as infeasible (MISLP callbacks) or terminate current SLP procedure

**Example**
The following example defines a callback for `xslp_optnode` which marks the node as infeasible (so that the branch is not worth pursuing further) if the error costs are too high.

```
SLPsetcallback(xslp_optnode,"MyOptNode")
function MyOptNode(Flag:integer):integer
if getparam("xslp_errorcosts") > 1000 then
  returned := 1
else
  returned := 0
end-if
end-function
```

**Further information**
All callbacks are of type `CbType1` except the following:
- `xslp_cascadevar` `CbType2`
- `xslp_itervar` `CbType2`
- `xslp_optnode` `CbType3`
- `xslp_prenode` `CbType3`
- `xslp_slpmessage` `CbMessage`
- `xslp_slpnode` `CbType3`

If a function of type `CbType1` returns a nonzero value then generally the current part of the SLP optimization procedure is terminated as follows:
- `xslp_cascadeend` No effect
Functions of type CbType2 receive the number of the variable which is currently being processed.

`xslp_itervar` returns the convergence status of the variable:

- 0: No change to the convergence status
- < 0: Variable has not converged
- n > 0: Variable has converged with status n

`xslp_cascadevar` can change the cascaded value for the variable. Its return value is ignored.

Functions of type CbType3 receive the feasibility status flag. This is normally zero (node is feasible) on entry. The functions cannot change the value of the flag, but the flag is set to a nonzero value if the function returns a nonzero value. A nonzero value for the flag means that the node has been declared infeasible and will not be examined further during the MISLP search.

Related topics

SLPsetcallback
Chapter 7
Error Messages

If a procedure or function encounters an error, then it normally terminates with a nonzero
return code and prints an error message.

The following is a list of the error messages. For reference purposes, each message is num-
bered. Errors are prefixed by \texttt{E-} and warnings by \texttt{W-}.

\textbf{E-02}  \textit{Unknown function \texttt{func}}
This message is printed if a function referenced in a \texttt{Func} statement cannot be
identified as an Xpress-SLP internal function or as a user function.

\textbf{E-03}  \textit{The linear part of the problem is infeasible}
This message is printed when the problem is loaded if the linear constraints can be
identified as infeasible without any optimization (typically because of inconsistent
bounds).

\textbf{E-04}  \textit{SLPloadprob failed with unrecognised return code \texttt{num}}
This message is printed when the problem is loaded and finds an error other than
E-03 (linear part infeasible). There is usually a preceding error message with more
details. Contact Dash Optimization if the reason is not apparent.

\textbf{W-05}  \textit{SLPglobal not executed - relaxed problem is not optimal}
This message is printed when global (MISLP) optimization is attempted (either ex-
plicitly with \texttt{SLPglobal} or implicitly using the "g" option in the optimization pro-
cedure) and the linear relaxation is infeasible.

\textbf{E-06}  \textit{SLPglobal returned error code \texttt{num}}
This message is printed when global (MISLP) optimization fails with an unrecover-
able error. The code number is that returned by the Optimizer or Xpress-SLP.

\textbf{E-07}  \textit{part of the XVitem has not been created}
This message is printed if a dynamic variable or expression has not been created
before it is used in an \texttt{XVitem} function.

\textbf{W-08}  \textit{Values not available}
This message is printed if an attempt is made to write the value of an expression
before the problem is optimized.

\textbf{W-12}  \textit{Unable to calculate user functions outside optimisation}
This message is printed if an attempt is made to write the value of an expression
including a user function. Only expressions of this type which appear as constraints
in the problem can be calculated.

\textbf{E-15}  \textit{Error: Inconsistent bounds for variable \texttt{var}}
This message is printed in conjunction with message E-03 when inconsistent bounds
are identified. The message is suppressed if \texttt{xslp_verbose} is FALSE.
E-23  **Division by zero in objective function**
This message is printed when checking a nonlinear objective which includes division by a (constant) zero.

E-24  **Memory allocation error**
This message occurs when the program is unable to allocate enough memory for the current operation. If no more memory is available on the machine, it may still be possible to run the model by pre-allocating some of the memory areas. Use the procedure SLPprintmemory immediately before the command which fails and use the output to preset some of the memory areas with the XSLP_MEM or XPRM_MEM parameters.

E-25  **Unidentified callback function func**
This message is printed when a function named `func` is included in a call to SLP-setcallback but cannot be identified as a function within the Mosel model nor as an external user function.

E-26  **Callback function func is not accessible**
This message is printed when a function named `func` is included in a call to SLP-setcallback and has been identified as an external user function, but has an invalid address, probably supplied by SLPsetuserfuncaddress or an equivalent external function using XSLPsetuserfuncaddress.

E-27  **Problem not loaded - use SLPloadprob first**
This message is printed when a call to SLPmaximize or SLPminimize does not include the definition of the objective, and the problem has not already been loaded using SLPloadprob. Include the objective in the call to the optimization procedure or use an explicit SLPloadprob first.

E-28  **Relaxed problem not solved - use SLPmaximize/minimize first**
This message is printed when a call to SLPglobal is not preceded by a call to SLPmaximize or SLPminimize to solve the relaxed problem.

E-29  **Error in definition of XV**
This message is printed when an array of `xvitems` is used as an XV without having been defined through the SLPDATA("XV") procedure.

E-30  **func is not available in this version of Xpress-SLP**
This message is printed when an attempt is made to access a function which is not included in the version of the Xpress-SLP optimizer being used. If the function is required, then Xpress-SLP must be upgraded to a later version.

E-31  **Unidentified user function func**
This message is printed by SLPcalluserfunc when the user function named has not been defined through SLPDATA("UF").

E-32  **Incorrect dimension for array arr in function func**
This message is printed when real arrays are being used in the interp function but have more than two dimensions in total.

E-33  **Incorrect dimension for array arr used as XV**
This message is printed when a real array is being used as an XV but has more than XSLP_maxxvarraydimension dimensions.

E-34  **func error accessing data in array arr**
This message is printed when data is unobtainable from a real array, probably because the requested data is out of range.
Index

A
Anonymous constraint, 3
Array
  multidimensional real used as XV, 7
    real, used as XV, 6, 7
    XV, 5, 6

C
Callback
to external function, 80
  Callbacks, 90
    chgtolset, 13
    chgvartolset, 14
    clearinitvals, 15
Column order, 8
Constraint
  anonymous, 3
    enforced, 75
    general, 3
  copyssoltointinit, 16
  cXSLPcommand, 17, 71

D
Determining row, 74
  DR, 74

E
E-02, 92
E-03, 92
E-04, 92
E-06, 92
E-07, 92
E-15, 92
E-23, 93
E-24, 93
E-25, 93
E-26, 93
E-27, 93
E-28, 93
E-29, 93
E-30, 93
E-31, 93
E-32, 93
E-33, 93
E-34, 93

EC, 75
Enforced constraint, 75
Error Messages, 92
  redirection, 9
Expression, general, 2

F
Func, 18
  Func (General SLP function), 88

G
genctr, 3
General constraint, 3
  anonymous, 3
    formulation, 3
    printing value, 4
General expression, 2
  printing value, 2
  getact, 19
  getcoeff, 20
  getconverged, 21
  getcsol, 22
  getdblattrib, 23
  getdblcontrol, 24
  getdual, 25
  gethistory, 26
  getintattrib, 27
  getintcontrol, 28
  getname, 29
  getnlcoeff, 30
  getrsol, 31
  getslack, 32
  getsol, 33
  getstatus, 34
  getstepbound, 35
  gettolset, 36
  gettype, 37
  getvarnum, 38
  getvartolset, 39
  gexp, 2

I
Initial value, 48, 76
  clearing, 15
    copying from solution, 16
  interp, 41
  ishidden, 40
IV, 76
  iXSLPcommand, 43, 71

M
Matrix
  column order, 8
    constraint formulation, 3

N
Names
  loading, 8

O
Optimizer
  loading names, 8

P
Problem pointer