# FICO® Xpress Optimization

# **Mosel Language**

**Cheat Sheet** 

#### Release 6.0

Last update 18 January, 2022

Xpress Mosel language: an algebraic modelling and procedural programming language; first published in 2001 (Dash Optimization), acquired by FICO in 2008, provided as free software since 2018.

# Structure of a Mosel program

Mosel program: text file with the extension .mos of the following form (or package / end-package for a Mosel package = library) that gets compiled to a platform-independent .bim (BIM=binary model) file:

```
model model name
  Compiler directives
 Parameters
 Body
end-model
```

**Compiler directives** Options, loading of libraries, version number

```
options explterm
                   ! Explicit termination with ';'
options noimplicit ! Don't allow implicit declaration
uses "mmxprs", "mmodbc"
version 1.0.0
```

Parameters Scalars of type integer, real, boolean, or string; run-time parameters (models and packages): specified with default value; package parameters (packages only): specify type

```
parameters
 DATAFILE="mydata.txt"
                        ! Runtime parameter
 'myparam': integer
                        ! Define a package parameter
end-parameters
```

Model body Statements other than compiler directives and parameters, including any number of declarations, initializations from/initializations to, functions and procedures

**Declarations** Simple objects can be used without declaring them, if their type is obvious; declarations are private by default

```
declarations
 ONE = 1
                                ! Constant declaration
 public val: real
                                ! Public scalar
 public procedure dosomething ! Public subroutine
end-declarations
```

**Initializations** Data type/access method is specified via I/O drivers (default text format: no prefix to filename; tmp: for temporary directory: bin: binary format)

```
initializations from "mmsheet.xls:mydat.xls"
 [A,B,C] as 'ARangeName'
COST as '[Sheet1$A1:C10]'
end-initializations
initializations to "mmodbc.odbc:mydat.accdb"
SOL as 'SolTable'
end-initializations
```

Functions and procedures Structure similar to a model: can define overloaded versions

```
function multiply(a,b: real): real
 returned:= a*b
end-function
public procedure writesomething
 writeln("something")
end-procedure
```

#### Comments

```
declarations
 make: array (R: range) of mpvar ! Comment on an entity
end-declarations
(! And this is a multi-line
 comment !) forall(t in 1..NT) ...
```

**Annotations** Meta data in a Mosel source file; either global or associated with public globally declared objects (including subroutines); predefined categories mc and doc

```
public declarations
 MYERR=11 !@doc.descr An error code constant
end-declarations
 (!@doc.
  @descr Some short description
  @return Explain the return value
public function getavalue: real
```

Types Built-in types (Mosel core, additional types via modules):

```
any boolean integer linctr mpproblem mpvar real string
```

### Naming conventions

Known/constant values (data): upper case or mixed Unknown values (variables), loop indices: lower case Subroutines: lower case, avoid underscores Constraints: mixed case (CamelCase)

### Data structures

Arrays collections of labeled objects of a given type where the label of an array entry is defined by its index tuple

```
declarations
 A: array(1...5) of real
 B: array(range, set of string) of integer
 x: array(1...10) of mpvar
 C: array(1..5) of real
end-declarations
A:: [4.5, 2.3, 7, 1.5, 10]
A(2) := 1.2
B:: (2..4, ["ABC", "DE"]) [15, 100, 90, 60, 40, 15]
C:= array(i in 1..5) x(i).sol
```

**Sets** collections of objects of the same type without establishing an order among them (as opposed to arrays and lists)

```
declarations
 S: set of string
 R: range
end-declarations
S:= {"A", "B", "C", "D"}
R := 1..10
```

**Lists** collections of objects of the same type

may contain the same element several times; order of elements is specified by construction

```
declarations
 L: list of integer
 M: array(range) of list of string
end-declarations
M:: (2..4)[['A','B','C'], ['D','E'], ['F','G','H','I']]
```

**Records** finite collections of objects of any type

Each component of a record is called a *field* and is characterized by its name and its type.

```
declarations
 ARC: array (ARCSET:range) of record
   Source, Sink: string
                               Source and sink of arc
   Cost: real
                             ! Cost coefficient
```

```
end-record
end-declarations
ARC(1).Source:= "B"
ARC(3).Cost:=1.5
```

**User types** treated in the same way as the predefined types of the Mosel language.

```
declarations
 mvreal = real
 myarray = array(1..10) of myreal
 public arc = record
   public Source,Sink: string ! Source and sink of arc
 end-record
 ARC: array(ARCSET:range) of arc
end-declarations
```

**Union types** Union: container capable of holding an object of one of a predefined set of types.

```
declarations
 u: string or real
                           ! 'string' or 'real' scalar
                           ! Scalar accepting any type
 a: any
! Type name for the union of the 4 basic Mosel types:
 basictype = string or integer or real or boolean
 U: array(range) of basictype ! Array of 'basictype'
end-declarations
```

# **Selection statements**

```
if [... elif] [... else] ... end-if
 if c=1 then
   writeln('c equals 1')
 elif c>1 then
   writeln('c is bigger than 1')
   writeln('c is smaller than 1')
 end-if
```

#### Inline "if" function

```
Inven(t) := stock(t) = buy(t) - sell(t) +
              if(t > 1, stock(t-1), 0)
```

### case ... end-case

```
case c of
 1,2 : writeln('c equals 1 or 2')
  3 : writeln('c equals 3')
  4..6: do
        writeln('c is in 4..6')
        writeln('c is not 1, 2 or 3')
        end-do
else
  writeln('c is not in 1..6')
end-case
```

# Loops

#### forall

```
forall(f in FAC, t in TIME)
 make(f,t) \le MAXCAP(f,t)
forall(t in TIME) do
 use(t) <= MAXUSE(t)
 buy(t) <= MAXBUY(t)
end-do
```

with equivalent to a forall loop stopped after the first iteration

```
with f='F1', t=1 do
   make(f,t) \le MAXCAP(f,t)
 end-do
while
 i := 1
 while (i \leq 10) do
   write(' ', i)
  i += 1
 end-do
```

#### repeat ... until

```
i := 1
repeat
 write(' ', i)
 i += 1
until i > 10
```

break, next break: jumps out of the current (or n) loop(s); next jumps to the beginning of the next iteration of the current loop

```
| 'L1': repeat
 while (condition1) do |
                           'L2': while (condition1) do
   if condition2 then
                                   if condition2 then
     break 2
                                     break 'L1'
   end-if
                                   end-if
 end-do
                           end-do
until condition3
                       | until condition3
```

# as counter

```
cnt:=0.0
writeln("Average of odd numbers in 1..10: ",
 (sum(cnt as counter, i in 1..10 | isodd(i)) i) / cnt)
```

# **Operators**

## Arithmetic operators

```
standard:
                     + - * /
power:
int. division/remainder:
                     mod div
sum:
                     sum(i in 1..10) ...
product:
                     prod(i in 1..10) ...
minimum/maximum:
                     min(i in 1..10) ...
count:
                     count(i in 1..10 | isodd(i))
```

#### Assignment operators

```
i := 10
i += 20
              ! Same as i := i + 20
              ! Same as i := i - 5
i -= 5
```

## Assignment operators with linear constraints

```
C := 5*x + 2*y <= 20
D := C + 7*y
! Same as (constraint type is dropped)
D := 5 * x + 9 * y - 20
C += 7*y
! Same as (constraint type is retained)
C := 5*x + 9*y <= 20
```

### Logical operators

```
constants:
           true, false
standard:
            and, or, not
AND:
            and(i in 1..10) ...
OR:
            or(i in 1..10) ...
comparison: <, >, =, <>, <=, >=
```

### Set operators

```
constants:
             {'A', 'B'}
union:
union:
             union(i in 1..10) ...
intersection:
intersection:
            inter(i in 1..10) ...
difference:
```

#### Set comparison operators

```
subset:
               Set1 <= Set2
               Set1 >= Set2
superset:
equals:
               Set1 = Set2
not equals:
              Set1 <>Set2
element of:
               "Oil5" in Set1
not element of: "Oil5" not in Set1
```

### List operators

```
constants:
               [1, 2, 3]
concatenation:
              +, sum, union
truncation:
equals:
               L1 = L2
               L1 <>L2
not equals:
```

### String expressions

```
"C:\\ddd1\ddd2"
                            ! Results in 'C:\ddd1ddd2'
'C:\\ddd1\ddd2'
                            ! Results in 'C:\\ddd1\ddd2'
`myfile.txt`
                            ! Content of file 'myfile.txt'
"Euro symbol as unicode: \u20AC"
```

### Union and reference operators

```
is:
            u is set of string ! u of union type
is not:
            u is not procedure
reference to: L:= [->cos,->sin,->arctan,->exp]
```

# Reserved words

The following words are reserved in Mosel. The upper case versions are also reserved (i.e. AND and and are keywords but not And).

```
a: and any array as
b: boolean break
   case constant count counter
   declarations div do dynamic
e: elif else end evaluation
f: false forall forward from function
h: hashmap
i: if imports in include initialisations
   initializations integer inter is is_binary
   is_continuous is_free is_integer is_partint
   is_semcont is_semint is_sos1 is_sos2
1: linctr list
m: max min mod model mpproblem mpvar
   namespace next not nsgroup nssearch
o: of options or
   package parameters procedure public prod
r: range real record repeat requirements return
s: set shared string sum
   then to true
   union until uses
v: version
w: while with
```

# **Mosel libraries**

Additional functionality is provided by Mosel libraries, which extend the basic Mosel language; current Mosel distribution:

Solvers	mmxprs, mmnl, mmxnlp, mmrobust, advmod, kalis,
OUIVEIS	minispis, minin, miniship, miniobast, advinod, kans,

nlsolv

Data handling mmodbc, mmsheet, mmoci, mmetc, mmxml,

fssappstudio

System mmsystem, mmhttp, mmssl, deploy, zlib

Model handling mmjobs

GUI, graphics mmsvg, mminsight

aec2, hadoop, s3, dmp, executor Cloud

matlab, r, mosjym, python3, math, random, mmreflect Other

# **Using the Mosel Command Line**

#### Standard model execution from the command line:

```
Execute (=compile/load/run) file 'mymodel.mos' :
mosel exec mymodel.mos
Short form (works with 'mymodel.mos' or 'mymodel.bim'):
mosel mymodel
Setting model runtime parameters:
mosel mymodel NT=5 DATAFILE="mydata.dat"
Compile to a specified BIM file name/location:
mosel comp mymodel.mos -o mybim.bim
Profiler run (output in 'mymodel.mos.prof'):
mosel prof mymodel.mos
```

### Some useful commands:

mosel -h	Command line help text
mosel -V	Mosel version
mosel lslib	List available modules/packages
mosel exam -h	Mosel version info and paths
mosel exam -a mybim.bim	Annotations of 'mybim.bim'
mosel exam -ps mmxprs	Parameters+subroutines of 'mmxprs

# Mosel command line debugger:

mosel debug mymodel.mos	Start Mosel debugger
help	Display debugger commands
break 20	Set breakpoint at line 20
cont	Execute up to the breakpoin
print D	Print out symbol 'D'
cont	Continue model execution
quit	Quit the debugger

moseldoc tool to generate an XML model documentation that is processed into HTML pages:

```
mosel comp -D mymodel.mos
moseldoc mymodel
```

# Links

### Online documentation:

http://www.fico.com/fico-xpress-optimization/docs/latest

Online examples: http://examples.xpress.fico.com/example.pl

# Free Community Edition download:

https://community.fico.com/optimization

Open source components: https://github.com/fico-xpress/mosel

# **Working with Xpress Workbench**

Xpress Workbench is an integrated development environment (IDE) for Mosel models and Xpress Insight applications.

### **Editor**

 $\oplus$ Open a new file/tab T Subdivide and re-arrange panes in the editor window

Code folding for blocks of Mosel statements

Unfold folded code

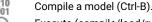
Line position markers during debugging

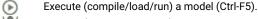
Breakpoints: click onto the gray area (left to the line number if it is displayed) preceding the editor text row



Delete breakpoint/deactivated breakpoint.

**Model execution**: for file name selected in the box next to the buttons



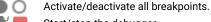


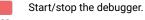


Execute (compile/load/run) a model in debug mode (F5).

Open Compiler Options or Run Dialog windows.

# Navigating in the debugger (select Debugger tab on right border)







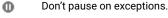
Resume/suspend model execution (F8). Step over an expression (F10).



Step into an expression (F11).



Step out of an expression (Shift-F11).



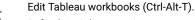
# **Deployment to Xpress Insight** (select *Xpress Insight* tab)

Publish selected model to Insight (Ctrl-Alt-P). Build an Insight app archive (Ctrl-Shift-A).



Debug a scenario.







Xpress Insight settings.