XReflect Module for Mosel

Developers Guide

9.7

Deprecated component

REFERENCE MANUAL

FICO® Xpress Optimization

FICO

FICO® Xpress Optimization 9.7

Deliverable Version: A

Last Revised: 30 Mar, 2022

Contents

1 Introduction 1
- 1.1 Limitations . . . 1
- 1.2 Namespace . . . 2

2 Switching to mmreflect 3
- 2.1 Switching from 'xreflect' to 'mmreflect' . . . 3
  - 2.1.1 The type 'basicvalue' . . . 3
  - 2.1.2 Accessing scalars, retrieving type and structural information . . . 4
  - 2.1.3 Accessing sets . . . 6
  - 2.1.4 Accessing arrays . . . 7
  - 2.1.5 Array iterators . . . 8
  - 2.1.6 Accessing subroutines . . . 9

3 basicvalue Type 10
- 3.1 Subroutines . . . 10
  - getboolvalue . . . 11
  - getintvalue . . . 12
  - getrealvalue . . . 13
  - getstringvalue . . . 14
  - gettextvalue . . . 15
  - gettype . . . 16

4 Accessing Entities 17
- 4.1 Querying Entity Types . . . 17
  - 4.1.1 Subroutines . . . 17
    - getvaluetype . . . 18
    - getstructtype . . . 19
    - getindexsettypes . . . 20
- 4.2 Scalar Entities . . . 21
  - 4.2.1 Subroutines . . . 21
    - getscalarvalue . . . 22
    - setscalarvalue . . . 23
- 4.3 Set Entities . . . 24
  - 4.3.1 Subroutines . . . 24
    - addsetelement . . . 25
    - checksetcontains . . . 26
    - finaliseset . . . 27
    - getindexsetelements . . . 28
    - getsetelements . . . 29
- 4.4 Array Entities . . . 30
  - 4.4.1 Subroutines . . . 30
    - arrayentryexists . . . 31
    - delarray . . . 32

delarrayentry . . . 33
getarrayentry . . . 34
setarrayentry . . . 35

5 arrayiterator Type 36

5.1 Example . . . 36
5.2 Subroutines . . . 36
gethasvalue . . . 37
getindices . . . 38
getvalue . . . 39
iteratorinit . . . 40
nextvalue . . . 41

6 Calling Subroutines 42

6.1 Subroutines . . . 42
callproc . . . 43
findproc . . . 44

Appendix 45

A Contacting FICO 45

FICO Customer Support . . . 45
Documentation . . . 45
FICO Learning . . . 46
Sales and maintenance . . . 46
About FICO . . . 46

Index 47

CHAPTER 1

Introduction

This module is deprecated. Its use is not recommended. New models should use the functionality of the *mmreflect* module instead. Existing models should consider migrating to *mmreflect*.

The *xreflect* module allows a package to programatically access Mosel entities whose names and types were not known at compile-time. For example, this allows a developer to write a function to increment each value in a named array by 1, regardless of the type or index sets of the array.

1.1 Limitations

xreflect can only operate on entities of the following types:

- `boolean`
- `integer`
- `real`
- `string`
- `set of boolean`
- `set of integer`
- `set of real`
- `set of string`
- `array of boolean`
- `array of integer`
- `array of real`
- `array of string`

Additionally, xreflect can only access entities declared as *public*, for example in a *public declarations* block.:

```
public declarations
  myArray: array(range) of string
end-declarations
```

1.2 Namespace

All types and subroutines exported by the *xreflect* module are part of the *xreflect* namespace; your model or package will need to reference this namespace in order to use this functionality. For example:

```
model myModel
 uses 'xreflect'
 nssearch xreflect

 public declarations
   myArray: array(range) of string
 end-declarations

 writeln('Type of values in myArray: ',getvaluetype('myArray'))

end-model
```

CHAPTER 2

Switching to mmreflect

2.1 Switching from 'xreflect' to 'mmreflect'

The module *mmreflect* now provides functional equivalence to *xreflect* (the latter is now deprecated), however, the usage of individual features is different given that the implementation of *mmreflect* relies on the recently introduced concept of unions and the possibility to work with subroutine references. The following listing compares the implementation of specific programming tasks with these two modules, using the examples from the *xreflect* manual. It should be noted that the functionality provided by *mmreflect* is more generic than the cases discussed here which focus on the portion of its functionality that matches what is accessible through *xreflect*. In particular, by working with unions there are no restrictions on the types or data structures that can be handled through *mmreflect*.

All functionality of the module *xreflect* is defined within the namespace `xreflect` to avoid clashes with functionality of the same name in *mmreflect*, the following code examples assume that namespace search has been specified for *xreflect*:

xreflect

```
uses "xreflect"
nssearch xreflect
```

mmreflect

```
uses "mmreflect"
```

2.1.1 The type 'basicvalue'

The module *xreflect* defines the type `basicvalue` for scalar values of Mosel types `boolean`, `integer`, `real`, `string`, `text`. The implementation of *mmreflect* relies on standard Mosel union types that work with all Mosel types and arbitrary data structures. This generic implementation obviously makes it possible for users to define their own union type `basicvalue` (limited to the five scalar types of *xreflect*) with access routines matching closely the restricted functionality available through *xreflect*:

```
uses "mmreflect"

! Defining a union type 'basicvalue' to hold scalars of basic Mosel types or 'text'
public declarations
  basicvalue = string or integer or real or boolean or text
  bvtrue, bvpi, bvhello: basicvalue
end-declarations

! Defining xreflect-style access routines
function getboolvalue(u: basicvalue):boolean
  returned:=u.boolean
end-function

public function gettype(u: basicvalue):string
  case u.typeid of
    boolean.id: returned:="boolean"
    integer.id: returned:="integer"
    real.id: returned:="real"
```

```
    string.id: returned:="string"
    text.id: returned:="string"
    else returned:="unsupported type"
  end-case
end-function

bvtrue := true                ! Same as:  bvtrue := basicvalue(true)
bvpi := 3.14                  ! Same as:  bvpi := basicvalue(3.14)
bvhello := 'hello'            ! Same as:  bvhello := basicvalue('hello')

writeln("***Compatibility routines:")
writeln('bv value=', bvtrue.boolvalue)     ! Output:  bv value=true
writeln('bv type=', bvtrue.type)           ! Output:  bv type=boolean
```

However, unless such restrictive typing is indeed required we strongly recommend to switch to working with the generic union type `any` as shown in the following examples, using standard Mosel functionality and the new set of access routines provided through *mmreflect*.

The following Mosel code example shows correspondence of standard union type handling for

- type `xreflect~basicvalue`
- subroutines `xreflect~get[bool|int|real|string|text]value` and `xreflect~gettype`

xreflect

```
!*** Working with type basicvalue
public declarations
   bvtrue, bvseven, bvpi, bvhello, bvname: basicvalue
end-declarations
bvtrue := basicvalue(true)
bvseven := basicvalue(7)
bvpi := basicvalue(3.14)
bvhello := basicvalue('hello')
bvname := basicvalue(text('my name'))

writeln('boolvalue=', bvtrue.boolvalue)
        ! Output:  boolvalue=true
writeln('is boolean:', bvtrue.type=XREFLECT_TYP_BOOL)
        ! Output:  is boolean:true
writeln('intvalue=', bvseven.intvalue)
        ! Output:  intvalue=7
writeln('is integer:', bvseven.type=XREFLECT_TYP_INT)
        ! Output:  is integer:true
writeln('realvalue=', bvpi.realvalue)
        ! Output:  realvalue=3.14
writeln('is real:', bvpi.type=XREFLECT_TYP_REAL)
        ! Output:  is real:true
writeln('stringvalue=' ,bvhello.stringvalue)
        ! Output:  stringvalue=hello
writeln('is string:', bvhello.type=XREFLECT_TYP_STRING)
        ! Output:  is string:true
writeln('textvalue=', bvname.textvalue)
        ! Output:  textvalue=my name
writeln('is string:', bvname.type=XREFLECT_TYP_STRING)
        ! Output:  is string:true
```

mmreflect

```
!*** Working with union types
public declarations
   bvtrue, bvseven, bvpi, bvhello, bvname: any
end-declarations
bvtrue := true
bvseven := 7
bvpi := 3.14
bvhello := 'hello'
bvname := text('my name')

writeln('boolvalue=', bvtrue.boolean)
        ! Output:  boolvalue=true
writeln('is boolean:', bvtrue.typeid=boolean.id)
        ! Output:  is boolean:true
writeln('intvalue=', bvseven.integer)
        ! Output:  intvalue=7
writeln('is integer:', bvseven is integer)
        ! Output:  is integer:true
writeln('realvalue=', bvpi.real)
        ! Output:  realvalue=3.14
writeln('is real:', bvpi is real)
        ! Output:  is real:true
writeln('stringvalue=', bvhello.string)
        ! Output:  stringvalue=hello
writeln('is string:', bvhello is string)
        ! Output:  is string:true
writeln('textvalue=', bvname.text)
        ! Output:  textvalue=my name
writeln('is text:', bvname is text)     ! is text:true
writeln('is string:', bvname is string) ! is string:false
```

2.1.2 Accessing scalars, retrieving type and structural information

The subroutines of *xreflect* search for model entities by their name. With *mmreflect* the entities are first retrieved into a union via a call to `findident` and all subsequent operations are carried out directly on this union.

The following Mosel code example shows *mmreflect* correspondence for

- subroutines `xreflect~getscalarvalue`, `xreflect~setscalarvalue`, `xreflect~getvaluetype`, `xreflect~getstructtype`, and `xreflect~getindexsettypes`

xreflect

```
public declarations
  mystr: string
  myarr: array(range) of real
  myset: set of integer
  mylst: list of integer
  mydate: date
  mybool=true
  myrng=1..5
end-declarations


!*** Retrieving and setting scalar values
mystr := 'hello'
writeln('mystr=',getscalarvalue('mystr'))
                          ! Output:  myvar=hello
setscalarvalue('mystr',basicvalue('world'))
writeln('mystr=',mystr)    ! Output:  myvar=world

!*** Accessing structural information

writeln('myvar=', getvaluetype('mystr'))
        ! Output:  mystr=string
writeln('mystr struct is variable:',
  getstructtype('mystr')=XREFLECT_STR_REF)
        ! Output:  mystr struct is variable:true

writeln('myarr=', getvaluetype('myarr'))
        ! Output:  myarr=real
writeln('myarr struct is array:',
  getstructtype('myvar')=XREFLECT_STR_REF)
        ! Output:  mystr struct is variable:true

writeln('myset=', getvaluetype('myset'))
        ! Output:  myset=integer
writeln('myset struct is set:',
  getstructtype('myset')=XREFLECT_STR_SET)
        ! Output:  mystr struct is variable:true

writeln('myrng=', getvaluetype('myrng'))
        ! Output:  myrng=integer
writeln('myrng struct is set:',
  getstructtype('myrng')=XREFLECT_STR_SET)
        ! Output:  mystr struct is variable:true

writeln('mydate=', getvaluetype('mydate'))
        ! Output:  mydate=unsupported
writeln('mydate struct is variable:',
  getstructtype('mydate')=XREFLECT_STR_REF)
        ! Output:  mydate struct is variable:true

writeln('mybool=',getvaluetype('mybool'))
        ! Output:  mybool=boolean
writeln('mybool struct is const:',
  getstructtype('mybool')=XREFLECT_STR_CONST)
        ! Output:  mystr struct is variable:true

writeln('mylst=', getvaluetype('mylst'))
        ! Output:  mylst=integer
writeln('mylst struct is unsupported:',
  getstructtype('mylst')=XREFLECT_STR_UNSUPPORTED)


!*** Index set type information
public declarations
  myarr2: array(set of integer,set of string) of real
end-declarations
writeln('index set types=',
  getindexsettypes('myarr2'))
      ! Output:  index set types=[ `integer', `string']
```

mmreflect

```
public declarations
  mystr: string
  myarr: array(range) of real
  myset: set of integer
  mylst: list of integer
  mydate: date
  mybool=true
  myrng=1..5
  a: any
end-declarations

!*** Retrieving and assigning scalar values
mystr := 'hello'
asproc(findident('mystr', a))
writeln('mystr=',a)        ! Output:  mystr=hello
a := 'world'
writeln('mystr=',mystr)    ! Output:  mystr=world

!*** Accessing structural information
res:=findident('mystr', a)
writeln(testtype(res,STRUCT_REF+string.id))  ! Output:  true
writeln('mystr=string: ', a is string)
        ! Output:  mystr=string: true
writeln('mystr struct is variable:', a.struct=STRUCT_REF)
        ! Output:  mystr struct is variable:true

asproc(findident('myarr', a))
writeln('myarr=real: ', a.eltype=real.id)
        ! Output:  myarr=real: true
writeln('myarr struct is array:', a.struct=STRUCT_ARRAY)
        ! Output:  myarr struct is array:true

asproc(findident('myset', a))
writeln('myset=integer: ', a.eltype=integer.id)
        ! Output:  myset=integer: true
writeln('myset struct is set:', a.struct=STRUCT_SET)
        ! Output:  myset struct is set:true

asproc(findident('myrng', a))
writeln('myrng=integer: ', a.eltype=integer.id)
        ! Output:  myrng=integer: true
writeln('myrng struct is set:', a.struct=STRUCT_SET)
        ! Output:  myrng struct is set:true

asproc(findident('mydate', a))
writeln('mydate=date: ', a.typeid=date.id)
        ! Output:  mydate=date: true
writeln('mydate struct is variable:', a.struct=STRUCT_REF)
        ! Output:  mydate struct is variable:true

asproc(findident('mybool', a))
writeln('mybool=boolean: ', a.eltype=boolean.id)
        ! Output:  mybool=boolean: true
writeln('mybool struct is const:', a.struct=STRUCT_CONST)
        ! Output:  mybool struct is const:true

asproc(findident('mylst', a))
writeln('mylst=integer: ', a.eltype=integer.id)
        ! Output:  mylst=integer: true
writeln('mylst struct is list:', a.struct=STRUCT_LIST)
        ! Output:  mylst struct is list:true

!*** Index set type information
public declarations
  myarr2: array(set of integer,set of string) of real
  lst: list of integer
end-declarations
asproc(findident('myarr2', a))
forall(i in 1..a.array.nbdim) lst+=[a.array.index(i).eltype]
writeln('index set types=',lst)
        ! Output:  index set types=[1,3]
```

2.1.3 Accessing sets

Since the set access routines in *mmreflect* work on unions, the type of the set must be specified when accessing the corresponding set.

The following Mosel code example shows *mmreflect* correspondence for

- subroutines `xreflect~addsetelement`, `xreflect~finaliseset`, `xreflect~checksetcontains`, `xreflect~getsetelements`, and `xreflect~copysetelements`

xreflect

```
public declarations
  dlst: list of basicvalue
end-declarations

myset := {1,3}




addsetelement('myset', basicvalue(4))
writeln('myset=', myset)     ! Output:  myset={1,3,4}

finaliseset('myset')         ! No further changes
!addsetelement('myset',basicvalue(6))   ! Error
writeln('contains 6=', checksetcontains('myset',
  basicvalue(6)))            ! Output:  contains 6=false

writeln('myset=', getsetelements('myset'))
                             ! Output:  myset=[1,3,4]
dlst:=[]
copysetelements('myset', dlst)
writeln(dlst)                ! Output:  [1,3,4]
```

mmreflect

```
public declarations
  dlst: list of any
  mysettype=set of integer
end-declarations
myset := {1,3}
asproc(findident('myset', a))
if a is not mysettype then
! Alternatively:
!if findident('myset', a, mysettype.id)=0 then
  writeln("Unexpected type"); exit(1)
end-if
a.mysettype+={4}
writeln('myset=', myset)     ! Output:  myset={1,3,4}

finalise(a.set)              ! No further changes
!a+={6}                      ! Results in an error
writeln('contains 6=', 6 in a.mysettype)
                             ! Output:  contains 6=false

writeln('myset=', a)
                             ! Output:  myset={1,3,4}
dlst:=[]
dlst:=list(a.set)
writeln(dlst)                ! Output:  [1,3,4]
```

2.1.4 Accessing arrays

When accessing arrays via *mmreflect* functionality the type of array elements needs to be stated, with the exception of the routines `delcell`, `exists`, and `create` for which it is sufficient to state the generic type `any`.

The following Mosel code example shows *mmreflect* correspondence for

- Subroutines `xreflect~arrayentryexists`, `xreflect~getarrayentry`, `xreflect~setarrayentry`, `xreflect~getindexsetelements`, `xreflect~copyindexsetelements`, `xreflect~delarray`, and `xreflect~delarrayentry`

xreflect

```
public declarations
  myarr4: dynamic array(range,set of integer) of real
  dlst: list of basicvalue
end-declarations

myarr4(1,100) := 5.0
myarr4(1,101) := 5.1
myarr4(2,101) := 5.2


writeln('exists(2,100)=', arrayentryexists('myarr4',
  [basicvalue(2),basicvalue(100)]))
        ! Output:  exists(2,100)=false

!*** Accessing array elements
writeln('myarr4(2,101)=', getarrayentry('myarr4',
  [basicvalue(2),basicvalue(101)]))
        ! Output:  myarr4(2,101)=5.2
setarrayentry( 'myarr4', [basicvalue(1),
  basicvalue(101)], basicvalue(17.5) )
writeln('myarr4(1,101)=', myarr4(1,101))
        ! Output:  myarr4(1,101)=17.5




setarrayentry( 'myarr4', [basicvalue(3),
  basicvalue(100)], basicvalue(2.0) )
writeln('myarr4_i1=', getindexsetelements('myarr4',1))
        ! Output:  myarr4_i1=[1,2,3]

copyindexsetelements('myarr4',1,dlst)
writeln(dlst)          ! Output:  [1,2,3]

delarrayentry('myarr4',
  [basicvalue(1),basicvalue(101)])
writeln('exists(1,101)=', arrayentryexists('myarr4',
  [basicvalue(1),basicvalue(101)]))
        ! Output:  exists(1,101)=false

delarray('myarr4')
writeln('size=', myarr4.size)
        ! Output:  size=0
```

mmreflect

```
public declarations
  myarr4: dynamic array(range,set of integer) of real
  dlst: list of any
end-declarations

myarr4(1,100) := 5.0
myarr4(1,101) := 5.1
myarr4(2,101) := 5.2

stat:= findident('myarr4', a)
if not testtype(stat, STRUCT_ARRAY) then exit(1); end-
if
writeln('exists(2,100)=', exists(a.array(2,100).any))
        ! Output:  exists(2,100)=false

!*** Array element access using standard union handling
writeln('myarr4(2,101)=', a.array(2,101).real)
        ! Output:  myarr4(2,101)=5.2
a.array(1,101).real := 17.5
writeln('myarr4(1,101)=', myarr4(1,101))
        ! Output:  myarr4(1,101)=17.5
!*** Alternative forms using array element access routines
setarrval(a, 7.5, 1,101)
getarrval(myarr4, u, 1,101)
writeln('myarr4(1,101)=', u) ! Output:  myarr4(1,101)=7.5
setarrval(myarr4, 2.5, [1,101])
getarrval(a, u, [1,101])
writeln('myarr4(1,101)=', u) ! Output:  myarr4(1,101)=2.5

a.array(3,100).real := 2.0
writeln('myarr4_i1=', a.array.index(1))
        ! Output:  myarr4_i1=1..3

dlst:= list(a.array.index(1))
writeln(dlst)          ! Output:  [1,2,3]


delcell(a.array(1,101).any)
writeln('exists(1,101)=', exists(a.array(1,101).real))
        ! Output:  exists(1,101)=false


reset(a.array)     ! Same as:  delcell(a.array)
writeln('size=', myarr4.size)
        ! Output:  size=0
```

2.1.5 Array iterators

The module *mmreflect* defines the type `iterator` that can be used in a similar way as the type `arrayiterator` of *xreflect* for enumerating the entries defined for a Mosel array.

The following Mosel code example shows *mmreflect* correspondence for

- type `xreflect~arrayiterator`
- subroutines `xreflect~gethasvalue`, `xreflect~getindices`, `xreflect~getvalue`, `xreflect~iteratorinit`, and `xreflect~nextvalue`

xreflect

```
public declarations
  myarray:dynamic array(range,set of string) of real
end-declarations
! Populate our example data-set
myarray(100,'fred') := 100.1
myarray(100,'jim') := 101.5
myarray(101,'fred') := 215.7

! Iterate through array entries using the arrayiterator
declarations
  it:arrayiterator
end-declarations

iteratorinit(it, 'myarray')

writeln(it.hasvalue)    ! Output:  false (before start)
while (nextvalue(it)) do
  write('myarray(', it.indices)
        ! Index tuple: list of basicvalue
  writeln(') = ', it.value)


end-do
  ! Output:
  !  myarray([100,fred]) = 100.1
  !  myarray([100,jim]) = 101.5
  !  myarray([101,fred]) = 215.7
writeln(it.hasvalue)   ! Output:  false (after end)
```

mmreflect

```
public declarations
  myarray:dynamic array(range,set of string) of real
end-declarations
! Populate our example data-set
myarray(100,'fred') := 100.1
myarray(100,'jim') := 101.5
myarray(101,'fred') := 215.7

! Iterate through array entries using the iterator
declarations
  it:iterator
  u:any
end-declarations
inititer(it, myarray)
writeln(it.status)              ! Output:  0 (before start)
writeln(exists(myarray(it)))    ! Output:  false
while (nextcell(it)) do
  write('myarray(', it.indices)  ! Index tuple: list of any
  writeln(') = ', myarray(it))
!*** Alternative form: using getarrval with iterator
!  getarrval(myarray,u,it.indices)
!  write(') = ',u)
end-do
  ! Output:
  !  myarray([100,fred]) = 100.1
  !  myarray([100,jim]) = 101.5
  !  myarray([101,fred]) = 215.7
writeln(it.status)              ! Output:  2 (after end)
writeln(exists(myarray(it)))    ! Output:  false
```

2.1.6 Accessing subroutines

The module *mmreflect* generalizes the handling of subroutines to include subroutine arguments and it also adds functionality for handling functions.

The following Mosel code example shows *mmreflect* correspondence for

- subroutines `xreflect~findproc` and `xreflect~callproc`

xreflect

```
!*** Calling a procedure
public procedure myproc
  writeln('hello world');
end-procedure
callproc('myproc')      ! Output:  hello world

public procedure myproc1
  writeln('hello');
end-procedure
public procedure myproc2(xyz:string)
  writeln('world: ', xyz);
end-procedure
! Subroutines with arguments are not supported


!*** Checking for a subroutine




writeln('findproc(myproc1)=', findproc('myproc1'))
        ! Output:   findproc(myproc1)=true
! Calling the subroutine
callproc('myproc')      ! Output:  hello world

writeln('findproc(myproc2)=', findproc('myproc2'))
! myproc2 will not be found as it takes an argument
        ! Output:   findproc(myproc2)=false




!*** Functions are not supported
public function myfunc: text
  returned:=text('-')*10
end-function
writeln('findproc(myfunc)=', findproc('myfunc'))
        ! Output:   findproc(myfunc)=false
```

mmreflect

```
!*** Calling a procedure
public procedure myproc
  writeln('hello world');
end-procedure
callproc('myproc')       ! Output:  hello world

public procedure myproc1
  writeln('hello');
end-procedure
public procedure myproc2(xyz:string)
  writeln('world: ', xyz);
end-procedure
! Calling a subroutine with an argument
callproc('myproc2', 'my text')  ! Output:  world: my text

!*** Checking for a subroutine
declarations
  srtype=procedure   ! Type definition for 'findident'
  srtype2=procedure(string)
end-declarations
writeln('findproc(myproc1)=',
  getstruct(findident('myproc1',a,srtype.id))=STRUCT_ROUTINE)
       ! Output:   findproc(myproc1)=true
! Calling the subroutine that has been retrieved
callproc(a)              ! Output:  hello world
writeln('findproc(myproc2)=',
  getstruct(findident('myproc2',a,srtype.id))=STRUCT_ROUTINE)
       ! Output:   findproc(myproc2)=false
writeln('findproc(myproc2)=',
  getstruct(findident('myproc2',a,srtype2.id))=STRUCT_ROUTINE)
       ! Output:   findproc(myproc2)=true

!*** Support for functions
public function myfunc: text
  returned:='-'*10
end-function
writeln('findproc(myfunc)=',
  testtype(findident('myfunc',a), STRUCT_ROUTINE+text.id))
       ! Output:   findproc(myfunc)=true
declarations
  res: any
end-declarations
callfunc(a,res)
writeln("Result=", res)  ! Output:  Result=----------
```

CHAPTER 3

basicvalue Type

The *xreflect* module exports a type `xreflect~basicvalue` which provides a way to manage values regardless of types. A `basicvalue` contains a single boolean, integer, real or string value. The type of value can be queried directly `gettype` function and you can access the value directly using `getboolvalue`, `getintvalue`, `getrealvalue` or `getstringvalue`.

A new `basicvalue` can be created from a boolean, integer, real, string or text using the constructor, e.g.:

```
bvtrue := basicvalue(true)
bvzero := basicvalue(0)
bvpi := basicvalue(3.14)
bvname := basicvalue('my name')
```

An uninitialized *basicvalue* will contain the boolean value `false`.

`basicvalue` should only be used when passing values to/from functions of the *xreflect* module. In other cases, the standard Mosel `any` type should always be used instead.

3.1 Subroutines

getboolvalue

Purpose

This subroutine is deprecated and will be removed in a future release. Use the `any` type instead of `basicvalue`.

Get boolean contained within `basicvalue`

Synopsis

```
function xreflect~getboolvalue( bv:xreflect~basicvalue ):boolean
```

Argument

`bv` The basicvalue to query

Return value

The boolean value from the basicvalue

Example

The following:

```
myval := basicvalue(true)
writeln('boolvalue=',myval.boolvalue)
```

produces this output:

```
boolvalue=true
```

Further information

If the referenced *basicvalue* does not contain a boolean, a reasonable type conversion will be applied.

Related topics

`gettype`, `getintvalue`, `getrealvalue`, `getstringvalue`

getintvalue

Purpose

This subroutine is deprecated and will be removed in a future release. Use the `any` *type instead of* `basicvalue`*.*

Get integer contained within `basicvalue`

Synopsis

```
function xreflect~getintvalue( bv:xreflect~basicvalue ):integer
```

Argument

`bv` The basicvalue to query

Return value

The integer value from the basicvalue

Example

The following:

```
myval := basicvalue(17)
writeln('intvalue=',myval.intvalue)
```

produces this output:

```
intvalue=17
```

Further information

If the referenced *basicvalue* does not contain an integer, a reasonable type conversion will be applied.

Related topics

`gettype`, `getboolvalue`, `getrealvalue`, `getstringvalue`

getrealvalue

Purpose

This subroutine is deprecated and will be removed in a future release. Use the `any` type instead of `basicvalue`.

Get real contained within `basicvalue`

Synopsis

```
function xreflect~getrealvalue( bv:xreflect~basicvalue ):real
```

Argument

`bv` The basicvalue to query

Return value

The floating point value from the basicvalue

Example

The following:

```
myval := basicvalue(17.5)
writeln('realvalue=',myval.realvalue)
```

produces this output:

```
realvalue=17.5
```

Further information

If the referenced *basicvalue* does not contain a real, a reasonable type conversion will be applied.

Related topics

`gettype`, `getboolvalue`, `getintvalue`, `getstringvalue`

getstringvalue

Purpose

This subroutine is deprecated and will be removed in a future release. Use the `any` type instead of `basicvalue`.

Get string contained within `basicvalue`

Synopsis

```
function xreflect~getstringvalue( bv:xreflect~basicvalue ):string
```

Argument

`bv` The basicvalue to query

Return value

The string value from the basicvalue

Example

The following:

```
myval := basicvalue('hello')
writeln('stringvalue=',myval.stringvalue)
```

produces this output:

```
stringvalue=hello
```

Further information

If the referenced *basicvalue* does not contain a string, a reasonable type conversion will be applied.

Related topics

`gettype`, `getboolvalue`, `getintvalue`, `getrealvalue`, `gettextvalue`

gettextvalue

Purpose

This subroutine is deprecated and will be removed in a future release. Use the `any` type instead of `basicvalue`.

Get string contained within `basicvalue`, as a text

Synopsis

```
function xreflect~gettextvalue( bv:xreflect~basicvalue ):text
```

Argument

`bv` The basicvalue to query

Return value

The string value from the basicvalue

Example

The following:

```
myval := basicvalue('hello')
writeln('textvalue=',myval.textvalue)
```

produces this output:

```
textvalue=hello
```

Further information

If the referenced *basicvalue* does not contain a string, a reasonable type conversion will be applied.

Related topics

`gettype`, `getboolvalue`, `getintvalue`, `getrealvalue`, `getstringvalue`

gettype

Purpose

This subroutine is deprecated and will be removed in a future release. Use the `any` *type instead of* `basicvalue`*.*

Get type of the value contained within a `basicvalue`

Synopsis

```
function xreflect~gettype( bv:xreflect~basicvalue ):string
```

Argument

`bv` The *basicvalue* to query

Return value

The type of the *basicvalue*, equal to one of the following constants:

`XREFLECT_TYP_BOOL`	boolean value
`XREFLECT_TYP_INT`	integer value
`XREFLECT_TYP_REAL`	real value
`XREFLECT_TYP_STRING`	string value

Example

The following:

```
myval := basicvalue('hello')
writeln('type of myval=',myval.type)
```

produces this output:

```
type of myval=string
```

Chapter 4

Accessing Entities

The *xreflect* module offers various subroutines for reading and writing the values of named entities.

4.1 Querying Entity Types

4.1.1 Subroutines

getvaluetype

Purpose

This subroutine is deprecated and will be removed in a future release. Use subroutine `findident` *of module mmreflect instead.*
Query the type of the value(s) that can be stored in an entity

Synopsis

```
function xreflect~getvaluetype( entityname:string ):string
```

Argument

`entityname` The name of the public entity to query

Return value

The value type, equal to one of the following constants:

`XREFLECT_TYP_BOOL`	boolean value
`XREFLECT_TYP_INT`	integer value
`XREFLECT_TYP_REAL`	real value
`XREFLECT_TYP_STRING`	string value
`XREFLECT_TYP_UNSUPPORTED`	a value type not supported by *xreflect*
`XREFLECT_NOT_FOUND`	*entityname* was not found in the dictionary

Example

The following:

```
public declarations
  myvar: string,
  myarr: array(range) of real
end-declarations
writeln('myvar=',getvaluetype('myvar'))
writeln('myarr=',getvaluetype('myarr'))
```

produces this output:

```
myvar=string
myarr=real
```

Further information

1. The *value type* of an entity is the type of the values it can contain. For example, the value type of an `array of integer` is `XREFLECT_TYP_INT`.
2. If the specified entity name cannot be found, the function will return `XREFLECT_NOT_FOUND`.

Related topics

`getstructtype`, `getindexsettypes`

getstructtype

Purpose

This subroutine is deprecated and will be removed in a future release. Use subroutine `findident` *of module mmreflect instead.*
Query the structural type of an entity

Synopsis

```
function xreflect~getstructtype( entityname:string ):string
```

Argument

`entityname` The name of the public entity to query

Return value

The structure type, equal to one of the following constants:

`XREFLECT_STR_ARR`	array
`XREFLECT_STR_CONST`	scalar constant
`XREFLECT_STR_REF`	scalar variable
`XREFLECT_STR_SET`	set
`XREFLECT_STR_UNSUPPORTED`	a structural type not supported by *xreflect*
`XREFLECT_NOT_FOUND`	*entityname* was not found in the dictionary

Example

The following:

```
public declarations
  myvar: string,
  myarr: array(range) of real
end-declarations
writeln('myvar=',getstructtype('myvar'))
writeln('myarr=',getstructtype('myarr'))
```

produces this output:

```
myvar=variable
myarr=array
```

Further information

1. The *structural type* of an entity defines how it structure(s) the values it stores. For example, the structural type of an `array of integer` is `XREFLECT_STR_ARR`.
2. If the specified entity name cannot be found, the function will return `XREFLECT_NOT_FOUND`.

Related topics

`getvaluetype`, `getindexsettypes`

getindexsettypes

Purpose

This subroutine is deprecated and will be removed in a future release. Use subroutine `geteltype` *of module mmreflect instead on the index set instead.*
Query the type of each index set of a named array entity

Synopsis

```
function xreflect~getindexsettypes( entityname:string ):list of string
```

Argument

`entityname` The name of the public entity to query

Return value

List of the value types; one of the following constants for each index set of the array:

`XREFLECT_TYP_BOOL`	boolean set
`XREFLECT_TYP_INT`	integer set
`XREFLECT_TYP_REAL`	real set
`XREFLECT_TYP_STRING`	string set
`XREFLECT_TYP_UNSUPPORTED`	a set type not supported by *xreflect*

Example

The following:

```
public declarations
  myarr: array(set of integer,set of string) of real
end-declarations
writeln('index set types=',getindexsettypes('myarr'))
```

produces this output:

```
index set types=[ `integer', `string']
```

Further information

1. If the specified entity name cannot be found, the model will terminate with a runtime error.
2. If the specified entity is not an array, the model will terminate with a runtime error.

Related topics

`getstructtype`, `getvaluetype`

4.2 Scalar Entities

4.2.1 Subroutines

getscalarvalue

Purpose

This subroutine is deprecated and will be removed in a future release. Use subroutine `findident` *of module mmreflect instead.*
Query the value of a scalar variable

Synopsis

```
function xreflect~getscalarvalue( entityname:string ):xreflect~basicvalue
```

Argument

`entityname` The name of the public entity to query

Return value

A *basicvalue* containing the value stored in the entity.

Example

The following:

```
public declarations
  myvar: string,
end-declarations
myvar := 'hello'
writeln('myvar=',getscalarvalue('myvar'))
```

produces this output:

```
myvar=hello
```

Further information

1. If the specified entity name cannot be found, the model will terminate with a runtime error.
2. If the specified entity is not of a supported type, the model will terminate with a runtime error.
3. If the specified entity is not a scalar variable or constant, the model will terminate with a runtime error.

Related topics

`gettype`, `setscalarvalue`

setscalarvalue

Purpose

This subroutine is deprecated and will be removed in a future release. Use subroutine `findident` *of module mmreflect instead.*
Update the value of a scalar variable

Synopsis

```
procedure xreflect~setscalarvalue( entityname:string,
      newvalue:xreflect~basicvalue )
```

Arguments

`entityname`	The name of the public entity to set
`newvalue`	The value to assign to the entity

Example

The following:

```
public declarations
  myvar: string,
end-declarations
setscalarvalue('myvar',basicvalue('hello'))
writeln('myvar=',myvar)
```

produces this output:

```
myvar=hello
```

Further information

1. If the specified entity name cannot be found, the model will terminate with a runtime error.
2. If the specified entity is not of a supported type, the model will terminate with a runtime error.
3. If the specified entity is not a scalar variable or constant, the model will terminate with a runtime error.
4. If the provided *basicvalue* contains a value of a different type from the specified entity, the model will terminate with a runtime error.
5. If the specified entity is a constant, the model will terminate with a runtime error unless the provided *basicvalue* contains the same value as the constant entity.

Related topics

`gettype`, `getscalarvalue`

4.3 Set Entities

4.3.1 Subroutines

addsetelement

Purpose

This subroutine is deprecated and will be removed in a future release. Use subroutine `findident` of module mmreflect to find set, then += operator to add elements.
Add a new element to a named set

Synopsis

```
procedure xreflect~addsetelement( entityname:string,
      newvalue:xreflect~basicvalue )
```

Arguments

`entityname`	The name of the public entity to add to
`newvalue`	The value to add to the entity

Example

The following:

```
public declarations
  myvar: set of integer,
end-declarations
myvar := {1,2,3}
addsetelement('myvar',basicvalue(4))
writeln('myvar=',myvar)
```

produces this output:

```
myvar={1,2,3,4}
```

Further information

1. If the specified set already contains the new value, it will be unchanged.
2. If the specified entity name cannot be found, the model will terminate with a runtime error.
3. If the specified entity is not a set of a supported type, the model will terminate with a runtime error.
4. If the supplied *basicvalue* does not contain a value of the same type as the specified set, the model will terminate with a runtime error.
5. If the specified entity is a constant set, the model will terminate with a runtime error unless the given value is already in the set.

Related topics

`checksetcontains`, `getsetelements`

checksetcontains

Purpose

This subroutine is deprecated and will be removed in a future release. Use subroutine `findident` *of module mmreflect to find set, then* `in` *operator to check contents.*
Check whether a named set contains a given value

Synopsis

```
function xreflect~checksetcontains( entityname:string,
      value:xreflect~basicvalue ):boolean
```

Arguments

`entityname`	The name of the public entity to access
`value`	*basicvalue* containing value to look for

Example

The following:

```
public declarations
  myvar=1..5,
end-declarations
writeln('contains 6=',checksetcontains('myvar',basicvalue(6)))
```

produces this output:

```
copntains 6=false
```

Further information

1. If the specified entity name cannot be found, the model will terminate with a runtime error.
2. If the specified entity is not a set of a supported type, the model will terminate with a runtime error.
3. If the supplied *basicvalue* does not contain a value of the same type as the specified set, the model will terminate with a runtime error.

Related topics

`getsetelements`, `addsetelement`

finaliseset

Purpose

This subroutine is deprecated and will be removed in a future release.
Finalise a named set

Synopsis

```
procedure xreflect~finaliseset( entityname:string )
```

Argument

`entityname` The name of the public entity to access

Example

The following:

```
public declarations
  myvar: set of integer,
end-declarations
myvar := {1,2,3}
finaliseset('myvar')
```

will prevent any further changes to `myvar`

Further information

1. If the specified entity name cannot be found, the model will terminate with a runtime error.
2. If the specified entity is not a set, the model will terminate with a runtime error.
3. If the specified set is already finalised, this function will do nothing

getindexsetelements

Purpose

This subroutine is deprecated and will be removed in a future release. Use subroutine `findident` of module mmreflect to find array.

Copy the elements from an index set of a specified array into a list of basicvalue

Synopsis

```
function xreflect~getindexsetelements( entityname:string, setnum:integer
      ):list of xreflect~basicvalue
procedure xreflect~copyindexsetelements( entityname:string, setnum:integer,
      destlist:list of xreflect~basicvalue )
```

Arguments

`entityname`	The name of the public array entity to access
`setnum`	Which index set to access (1 being the first)
`destlist`	The list into which to copy the entities

Return value

A newly-created list containing each value from the specified index set, in *basicvalue* types

Example

The following:

```
public declarations
  myvar: array(1..3,4..5) of string,
end-declarations
writeln('myvar=',getindexsetelements('myvar',1))
```

produces this output:

```
myvar=[1,2,3]
```

Further information

1. `copyindexsetelements` is similar to `getindexsetelements`, but copies the values into an existing list rather than returning a new one.
2. If the specified entity name cannot be found, the model will terminate with a runtime error.
3. If the specified entity is not an array, the model will terminate with a runtime error.
4. If the requested index set is not of a supported type, the model will terminate with a runtime error.

Related topics

`getsetelements`

getsetelements

Purpose

This subroutine is deprecated and will be removed in a future release. Use subroutine `findident` of module mmreflect to find set.

Copy the elements from a specified set into a list of basicvalue

Synopsis

```
function xreflect~getsetelements( entityname:string ):list of
      xreflect~basicvalue
procedure xreflect~copysetelements( entityname:string, destlist:list of
      xreflect~basicvalue )
```

Arguments

`entityname`	The name of the public entity to accessfu
`destlist`	The list into which to copy the entities

Return value

A newly-created list containing each value from the specified entity, in *basicvalue* types

Example

The following:

```
public declarations
  myvar: set of integer,
end-declarations
myvar := {1,2,3}
writeln('myvar=',getsetelements('myvar'))
```

produces this output:

```
myvar=[1,2,3]
```

Further information

1. `copysetelements` is similar to `getsetelements`, but copies the values into an existing list rather than returning a new one.
2. If the specified entity name cannot be found, the model will terminate with a runtime error.
3. If the specified entity is not a set of a supported type, the model will terminate with a runtime error.

Related topics

`checksetcontains`, `getindexsetelements`

4.4 Array Entities

4.4.1 Subroutines

arrayentryexists

Purpose

This subroutine is deprecated and will be removed in a future release. Use subroutine `findident` of module mmreflect to find array.

Check if a named array contains a given entry

Synopsis

```
function xreflect~arrayentryexists( entityname:string, indices:list of
      xreflect~basicvalue ):boolean
```

Arguments

`entityname`	The name of the public entity to access
`indices`	A list containing one value for each index set of the given array

Return value

true if the specified array contains an entry with the given index values, false otherwise

Example

The following:

```
public declarations
  myvar: dynamic array(set of integer,set of integer) of real
end-declarations
myvar(1,100) := 5.0
myvar(1,101) := 5.1
myvar(2,101) := 5.2
writeln('exists(2,100)=',arrayentryexists('myvar',
    [basicvalue(2),basicvalue(101)]))
```

produces this output:

```
exists(2,100)=false
```

Further information

1. If the specified entity name cannot be found, the model will terminate with a runtime error.
2. If the specified entity is not an array, the model will terminate with a runtime error.
3. If the supplied indices list does not contain one basicvalue of the correct type for each index set of the array, the model will terminate with a runtime error.
4. The array index set contents will not be modified by this function

Related topics

`getarrayentry`, `setarrayentry`

delarray

Purpose

This subroutine is deprecated and will be removed in a future release. Use subroutine `findident` *of module mmreflect to find array.*
Delete all entries from the a named array

Synopsis

```
procedure xreflect~delarray( entityname:string )
```

Argument

`entityname` The name of the public entity to access

Further information

1. If the specified entity name cannot be found, the model will terminate with a runtime error.
2. If the specified entity is not an array, the model will terminate with a runtime error.

Related topics

`delarrayentry`

delarrayentry

Purpose

This subroutine is deprecated and will be removed in a future release.
Delete a given entry from a named array

Synopsis

```
procedure xreflect~delarrayentry( entityname:string, indices:list of
      xreflect~basicvalue )
```

Arguments

`entityname`	The name of the public entity to access
`indices`	A list containing one value for each index set of the given array

Further information

1. If the array does not contain an entry for the given indices, this subroutine will do nothing.
2. If the specified entity name cannot be found, the model will terminate with a runtime error.
3. If the specified entity is not an array, the model will terminate with a runtime error.
4. If the supplied indices list does not contain one basicvalue of the correct type for each index set of the array, the model will terminate with a runtime error.
5. The array index set contents will not be modified by this function

Related topics

`delarray`

getarrayentry

Purpose

This subroutine is deprecated and will be removed in a future release. Use subroutine `findident` of module mmreflect to find array.

Get a single value from a named array

Synopsis

```
function xreflect~getarrayentry( entityname:string, indices:list of
      xreflect~basicvalue ):xreflect~basicvalue
```

Arguments

`entityname`	The name of the public entity to access
`indices`	A list containing one value for each index set of the given array

Return value

A basicvalue containing the array entry for the given index values

Example

The following:

```
public declarations
  myvar: dynamic array(set of integer,set of integer) of real
end-declarations
myvar(1,100) := 5.0
myvar(1,101) := 5.1
myvar(2,101) := 5.2
writeln('myvar(2,101)=',getarrayentry('myvar',[basicvalue(2),basicvalue(101)]))
```

produces this output:

```
myvar(2,101)=5.2
```

Further information

1. If the array does not contain an entry for the given indices, the function will return the default value for the array type (e.g. false, 0 or "")
2. If the specified entity name cannot be found, the model will terminate with a runtime error.
3. If the specified entity is not an array of a supported type, the model will terminate with a runtime error.
4. If the supplied indices list does not contain one basicvalue of the correct type for each index set of the array, the model will terminate with a runtime error.
5. The array index set contents will not be modified by this function

Related topics

`arrayentryexists`, `setarrayentry`

setarrayentry

Purpose

This subroutine is deprecated and will be removed in a future release. Use subroutine `findident` of module mmreflect to find array.
Set a given entry of a named array

Synopsis

```
procedure xreflect~setarrayentry( entityname:string, indices:list of
      xreflect~basicvalue, newval:xreflect~basicvalue )
```

Arguments

`entityname`	The name of the public entity to access
`indices`	A list containing one value for each index set of the given array
`newval`	The value to insert into the array

Return value

A basicvalue containing the array entry for the given index values

Example

The following:

```
public declarations
  myvar: dynamic array(set of integer,set of integer) of real,
end-declarations
myvar(1,100) := 5.0
myvar(1,101) := 5.1
myvar(2,101) := 5.2
setarrayentry( 'myvar', [basicvalue(1),basicvalue(101)], basicvalue(17.5) )
writeln('myvar(1,101)=',myvar(1,101))
```

produces this output:

```
myvar(1,101)=17.5
```

Further information

1. If the specified entity name cannot be found, the model will terminate with a runtime error.
2. If the specified entity is not an array of a supported type, the model will terminate with a runtime error.
3. If the specified entity does not contain values of the same type passed in *newval*, the model will terminate with a runtime error.
4. If the supplied indices list does not contain one basicvalue of the correct type for each index set of the array, the model will terminate with a runtime error.
5. The array index set contents may be modified by this function

Related topics

`getarrayentry`, `arrayentryexists`

CHAPTER 5

arrayiterator Type

The *xreflect* module provides a new type *xreflect arrayiterator* which will allow you to programatically iterate through the entries of a named array. Each `arrayiterator` variable represents an iteration through the array, from start to end.

You initially associate the *arrayiterator* with an array by calling the `iteratorinit` procedure. The iterator has a current position in the array; this will initially be before the first entry. Then every time you call `nextvalue`, the iterator will advance to the next entry in the array - initially the first, then the second, etc. `nextvalue` returns true until the iterator reaches the end of the array, when it returns false.

While iterating through the array, you can obtain information about the current entry using the `getindices` and `getvalue` functions.

5.1 Example

The following example demonstrates iterating through the array and outputting each element, including index values

```
! Populate our example data-set
public declarations
  myarray:dynamic array(set of integer,set of string) of real
end-declarations
myarray(100,'fred') := 100.1
myarray(100,'jim') := 101.5
myarray(101,'fred') := 215.7

! Iterate through the array entries using the arrayiterator
declarations
  it:arrayiterator
end-declarations
iteratorinit(it, 'myarray')
while (nextvalue(it)) do
  write('myarray(')
  forall (v in it.indices) do
    write(v,',')
  end-do
  writeln(') = ',it.value)
end-do
```

will output:

```
myarray(100,fred,) = 100.1
myarray(100,jim,) = 101.5
myarray(101,fred,) = 215.7
```

5.2 Subroutines

gethasvalue

Purpose

This subroutine is deprecated and will be removed in a future release. Use type `iterator` *of module mmreflect.*

Queries whether the array iterator is positioned on an array entry

Synopsis

```
function xreflect~gethasvalue( it:xreflect~arrayiterator ):boolean
```

Argument

`it` The array iterator

Return value

true if the iterator is positioned on an array entry, false if it before the first entry or after the last

Related topics

`iteratorinit`, `nextvalue`

getindices

Purpose

This subroutine is deprecated and will be removed in a future release. Use type `iterator` *of module mmreflect.*

Obtain the index values for the current entry of the array iterator

Synopsis

```
function xreflect~getindices( it:xreflect~arrayiterator ):list of
      xreflect~basicvalue
```

Argument

`it` The array iterator

Return value

A list of one basicvalue for each index set of the array

Further information

If the iterator is positioned before the first or after the last array entry, returns an empty list

Related topics

`getvalue`

getvalue

Purpose

This subroutine is deprecated and will be removed in a future release. Use type `iterator` *of module mmreflect.*

Obtain the value for the current entry of the array iterator

Synopsis

```
function xreflect~getvalue( it:xreflect~arrayiterator ):xreflect~basicvalue
```

Argument

`it` The array iterator

Return value

The value of the current entry of the array, as a *basicvalue*

Further information

If the iterator is positioned before the first or after the last array entry, returns a boolean *basicvalue* containing `false`

Related topics

`getindices`

iteratorinit

Purpose

This subroutine is deprecated and will be removed in a future release. Use type `iterator` *of module mmreflect.*

Position an array iterator before the first entry of a named array

Synopsis

```
procedure xreflect~iteratorinit( it:xreflect~arrayiterator,
      entityname:string )
```

Arguments

`it`	The array iterator
`entityname`	Name of a public array entity

Further information

1. If the specified entity name cannot be found, the model will terminate with a runtime error.
2. If the specified entity is not an array of a supported type, the model will terminate with a runtime error.
3. If the specified array has an index set of an unsupported type, the model will terminate with a runtime error.

Related topics

`nextvalue`, `getindices`, `getvalue`

nextvalue

Purpose

This subroutine is deprecated and will be removed in a future release. Use type `iterator` *of module mmreflect.*

Advance the array iterator to the next entry of the array

Synopsis

```
function xreflect~nextvalue(it:xreflect~arrayiterator):boolean
```

Argument

`it` The array iterator

Return value

true if the array iterator is now positioned on an array entry, false if it is now positioned after the last entry of the array

Related topics

`iteratorinit`

CHAPTER 6

Calling Subroutines

The *xreflect* module allows you to call a zero-argument public procedure declared in the current model or an imported package.

6.1 Subroutines

callproc

Purpose

This subroutine is deprecated and will be removed in a future release. Use subroutine `callproc` of module mmreflect instead.
Call a zero-argument public procedure with the given name

Synopsis

```
procedure xreflect~callproc( procname:string )
```

Argument

`procname` The name of the procedure to call

Example

The following:

```
public procedure myproc
 writeln('hello world');
end-procedure
callproc('myproc')
```

produces this output:

```
hello world
```

Further information

1. If procname is not the name of a 0-argument procedure, the model will terminate with a runtime error.
2. If procname is overloaded, callproc will call the overload that has 0-arguments.

Related topics

`findproc`

findproc

Purpose

This subroutine is deprecated and will be removed in a future release. Use subroutine `findident` of module mmreflect instead.
Check if a zero-argument public procedure with the given name exists

Synopsis

```
function xreflect~findproc( procname:string ):boolean
```

Argument

`procname` The name of the procedure to find

Return value

If a zero-argument public procedure with the given name is found, true. Otherwise, false.

Example

In the following:

```
public procedure myproc1
  writeln('hello');
 end-procedure
 public procedure myproc2(xyz:string)
  writeln('world');
 end-procedure
 writeln('findproc(myproc1)=',findproc('myproc1'))
 writeln('findproc(myproc2)=',findproc('myproc2'))
```

myproc1 will be found but myproc2 will not be found as it requires an argument. Therefore the example produces this output:

```
findproc(myproc1)=true
findproc(myproc2)=false
```

Further information

1. If procname is the name of a function, findproc will return false.
2. If procname takes arguments, findproc will return false.
3. If procname is overloaded, findproc will return true if one of the overloads takes no arguments.

Related topics

`callproc`

Appendix A

Contacting FICO

FICO provides clients with support and services for all our products.

FICO Customer Support

FICO Customer Support offers technical support and services ranging from self-help tools to direct assistance with a FICO technical support engineer. Support is available to all clients who have an active maintenance contract.

The FICO Customer Self-Service Portal (support.fico.com) is a secure web portal that allows users to open, review, and update their support cases; manage their organization's portal users; find solutions to common problems in the FICO Knowledge Base; and view the availability of their cloud applications 24 hours a day, 7 days a week.

You can find support contact information and a link to the FICO Customer Self-Service Portal (online support) on the Product Support home page (www.fico.com/en/product-support).

Please include *'Xpress'* in the subject line of your support queries.

Documentation

FICO continually looks for new ways to improve and enhance the value of the products and services we provide.

If you have comments or suggestions regarding how we can improve this documentation, let us know by sending your suggestions to techpubs@fico.com. Please include your contact information (name, company, email address, and optionally, your phone number) so we may reach you if we have questions.

FICO Learning

FICO Learning is the principal provider of product training for our clients and partners. FICO Learning offers instructor-led classroom courses, web-based training, seminars, and training tools for both new user enablement and ongoing performance support.

For additional information, visit the FICO Learning home page at www.fico.com/en/product-training or email producteducation@fico.com.

Sales and maintenance

If you need information on other Xpress Optimization products, or you need to discuss maintenance contracts or other sales-related items, contact FICO by:

- Phone: +1 (408) 535-1500 or +44 207 940 8718
- Web: www.fico.com/optimization and use the available contact forms

About FICO

FICO (NYSE:FICO) is a leading analytics software company, helping businesses in 90+ countries make better decisions that drive higher levels of growth, profitability, and customer satisfaction. Learn more at www.fico.com or contact us at www.fico.com/en/contact-us.

Index

A
addsetelement, 25
arrayentryexists, 31

C
callproc, 43
checksetcontains, 26

D
delarray, 32
delarrayentry, 33

F
finaliseset, 27
findproc, 44

G
getarrayentry, 34
getboolvalue, 11
gethasvalue, 37
getindexsetelements, 28
getindexsettypes, 20
getindices, 38
getintvalue, 12
getrealvalue, 13
getscalarvalue, 22
getsetelements, 29
getstringvalue, 14
getstructtype, 19
gettextvalue, 15
gettype, 16
getvalue, 39
getvaluetype, 18

I
iteratorinit, 40

N
nextvalue, 41

S
setarrayentry, 35
setscalarvalue, 23