# metap

*Useful meta-predicates protocol.*

author:
    `Paulo Moura`
version:
    `5.0`
date:
    `2011/1/18`

compilation:
    `static, context_switching_calls`

*(no dependencies on other files)*

## Public interface

### include/3

*Returns a list of all list elements that satisfy a predicate.*

compilation:
    `static`

template:
    `include(Closure,List,Included)`

meta-predicate template:
    `include(1,*,*)`

mode - number of solutions:
    `include(+callable,+list,-list) - one`

### exclude/3

*Returns a list of all list elements that fail to satisfy a predicate.*

compilation:
    `static`

template:
    `exclude(Closure,List,Excluded)`

meta-predicate template:
    `exclude(1,*,*)`

mode - number of solutions:
    `exclude(+callable,+list,-list) - one`

### findall_member/4

*Finds all members of a list that satisfy a given test.*

compilation:
    `static`

template:
    `findall_member(Member,List,Test,Result)`

meta-predicate template:
    `findall_member(*,*,0,*)`

mode - number of solutions:
    `findall_member(@term,+list,@callable,-list) - one`

## findall_member/5

*Finds all members of a list that satisfy a given test appending the given tail to the result.*

compilation:
```
static
```
template:
```
findall_member(Member,List,Test,Result,Tail)
```
meta-predicate template:
```
findall_member(*,*,0,*,*)
```
mode - number of solutions:
```
findall_member(@term,+list,@callable,-list,+list) - one
```

## partition/4

*Partition a list of elements in two lists using a predicate.*

compilation:
```
static
```
template:
```
partition(Closure,List,Included,Excluded)
```
meta-predicate template:
```
partition(1,*,*,*)
```
mode - number of solutions:
```
partition(+callable,+list,-list,-list) - one
```

## partition/6

*Partitions a list in lists with values less, equal, and greater than a given value using a comparison predicate with the same argument order as compare/3.*

compilation:
```
static
```
template:
```
partition(Closure,List,Value,Less,Equal,Greater)
```
meta-predicate template:
```
partition(3,*,*,*,*,*)
```
mode - number of solutions:
```
partition(+callable,+list,@term,-list,-list,-list) - one
```

## fold_left/4

*List folding (left associative).*

compilation:
```
static
```
template:
```
fold_left(Closure,Accumulator,List,Result)
```
meta-predicate template:
```
fold_left(3,*,*,*)
```
mode - number of solutions:
```
fold_left(+callable,?term,+list,?term) - zero_or_more
```

## scan_left/4

*List scanning; similar to folding but returns the intermediate and final results (left associative).*

compilation:
    `static`

template:
    `scan_left(Closure,Accumulator,List,Results)`

meta-predicate template:
    `scan_left(3,*,*,*)`

mode - number of solutions:
    `scan_left(+callable,?term,+list,?list) - zero_or_more`

## fold_right/4

*List folding (right associative).*

compilation:
    `static`

template:
    `fold_right(Closure,Accumulator,List,Result)`

meta-predicate template:
    `fold_right(3,*,*,*)`

mode - number of solutions:
    `fold_right(+callable,?term,+list,?term) - zero_or_more`

## scan_right/4

*List scanning; similar to folding but returns the intermediate and final results (right associative).*

compilation:
    `static`

template:
    `scan_right(Closure,Accumulator,List,Results)`

meta-predicate template:
    `scan_right(3,*,*,*)`

mode - number of solutions:
    `scan_right(+callable,?term,+list,?list) - zero_or_more`

## map/2

*True if the predicate succeeds for each list element.*

compilation:
    `static`

template:
    `map(Closure,List)`

meta-predicate template:
    `map(1,*)`

mode - number of solutions:
    `map(+callable,?list) - zero_or_more`

## map/3

*List mapping predicate taken arguments from two lists of elements.*

compilation:
    `static`

template:
    `map(Closure,List1,List2)`

meta-predicate template:
`map(2,*,*)`

mode - number of solutions:
`map(+callable,?list,?list) - zero_or_more`

### map/4

*List mapping predicate taken arguments from three lists of elements.*

compilation:
`static`

template:
`map(Closure,List1,List2,List3)`

meta-predicate template:
`map(3,*,*,*)`

mode - number of solutions:
`map(+callable,?list,?list,?list) - zero_or_more`

### map/5

*List mapping predicate taken arguments from four lists of elements.*

compilation:
`static`

template:
`map(Closure,List1,List2,List3,List4)`

meta-predicate template:
`map(4,*,*,*,*)`

mode - number of solutions:
`map(+callable,?list,?list,?list,?list) - zero_or_more`

### map/6

*List mapping predicate taken arguments from five lists of elements.*

compilation:
`static`

template:
`map(Closure,List1,List2,List3,List4,List5)`

meta-predicate template:
`map(5,*,*,*,*,*)`

mode - number of solutions:
`map(+callable,?list,?list,?list,?list,?list) - zero_or_more`

### map/7

*List mapping predicate taken arguments from six lists of elements.*

compilation:
`static`

template:
`map(Closure,List1,List2,List3,List4,List5,List6)`

meta-predicate template:
`map(6,*,*,*,*,*,*)`

mode - number of solutions:
`map(+callable,?list,?list,?list,?list,?list,?list) - zero_or_more`

### map/8

*List mapping predicate taken arguments from seven lists of elements.*

compilation:
```
static
```
template:
```
map(Closure,List1,List2,List3,List4,List5,List6,List7)
```
meta-predicate template:
```
map(7,*,*,*,*,*,*,*)
```
mode - number of solutions:
```
map(+callable,?list,?list,?list,?list,?list,?list,?list) - zero_or_more
```

### map_reduce/5

compilation:
```
static
```
meta-predicate template:
```
map_reduce(2,3,*,*,*)
```
mode - number of solutions:
```
map_reduce(+callable,+callable,+term,?list,?term) - zero_or_more
```

## Protected interface

*(none)*

## Private predicates

*(none)*