loopp

Loop control constructs protocol.

author:
`Paulo Moura`
version:
`1.21`
date:
`2009/5/19`

compilation:
`static, context_switching_calls`

(no dependencies on other files)

Public interface

whiledo/2

While Condition is true do Action.

compilation:
`static`

template:
`whiledo(Condition,Action)`

meta-predicate template:
`whiledo(0,0)`

mode - number of solutions:
`whiledo(+callable,@callable) - zero_or_one`

dowhile/2

Do Action while Condition is true.

compilation:
`static`

template:
`dowhile(Action,Condition)`

meta-predicate template:
`dowhile(0,0)`

mode - number of solutions:
`dowhile(@callable,+callable) - zero_or_one`

foreach/3

For each element Element in List call Goal.

compilation:
`static`

template:
`foreach(Element,List,Goal)`

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

mode - number of solutions:
`foreach(@var,+list(term),@callable) - zero_or_one`

forto/3

Call Goal counting up from First to Last. Increment is 1. For convenience and clarity, First and Last can be arithmetic expressions. This predicate fails iff the Goal fails.

compilation:
`static`

template:
`forto(First,Last,Goal)`

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

mode - number of solutions:
`forto(+number,+number,@callable) - zero_or_one`

forto/4

Call Goal counting up from First to Last and instantiating Count to each successive value. Increment is 1. For convenience and clarity, First and Last can be arithmetic expressions. This predicate fails iff the Goal fails.

compilation:
`static`

template:
`forto(Count,First,Last,Goal)`

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

mode - number of solutions:
`forto(@var,+number,+number,@callable) - zero_or_one`

forto/5

Call Goal counting up from First to Last and instantiating Count to each successive value. For convenience and clarity, First, Last, and Increment can be arithmetic expressions (uses Increment absolute value). This predicate fails iff the Goal fails.

compilation:
`static`

template:
`forto(Count,First,Last,Increment,Goal)`

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

mode - number of solutions:
`forto(@var,+number,+number,+number,@callable) - zero_or_one`

fordownto/3

Call Goal counting down from First to Last. Decrement is 1. For convenience and clarity, First and Last can be arithmetic expressions. This predicate fails iff the Goal fails.

compilation:
`static`

template:
`fordownto(First,Last,Goal)`

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

mode - number of solutions:
`fordownto(+number,+number,@callable) - zero_or_one`

fordownto/4

Call Goal counting down from First to Last and instantiating Count to each successive value. Decrement is 1. For convenience and clarity, First and Last can be arithmetic expressions. This predicate fails iff the Goal fails.

compilation:
```
static
```

template:
```
fordownto(Count,First,Last,Goal)
```

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

mode - number of solutions:
```
fordownto(@var,+number,+number,@callable) - zero_or_one
```

fordownto/5

Call Goal counting down from First to Last and instantiating Count to each successive value. For convenience and clarity, First, Last, and Decrement can be arithmetic expressions (uses Decrement absolute value). This predicate fails iff the Goal fails.

compilation:
```
static
```

template:
```
fordownto(Count,First,Last,Decrement,Goal)
```

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

mode - number of solutions:
```
fordownto(@var,+number,+number,+number,@callable) - zero_or_one
```

Protected interface

(none)

Private predicates

(none)