NLoops – a brief summary

NLoops ref:0.4b

This document provides a brief summary of the forms for declaring classes, instances, etc and for calling methods. For a better introduction to NLoops please refer to the user guide.

initialisation

to #reset-objects

reset all NLoops structures. This removes all earlier declarations. It is a good idea to reset objects before doing any declarations

declarations

to #def-class [#name# #slots#]

declare a new class with specified slots

```
syntax:  #def-class  super-class.class-name  [...slots...]
eg:      #defclass “ant.red-ant” [“age” [“size” 5] “class.color”]
```

to #def-instance [#name# #slots#]

define a new instance with a given name

```
syntax:  #def-instance  class-name.instance-name    [...instance-slots...]
eg:      #def-instance  “red-ant.fred”   [[“age” 3]]
```

to #def-method [#method#]

the preferred way to define a method (unless you need to redirect the method to a NL procedure of a different name)

```
syntax:  #def-method  class-name.method-name
eg:      #def-method  “ant.move”
```

in practice #def-method calls map to #set-method (see below) so...

```
         #def-method "c.m" => #set-method "c.m" "c.m"
```

to #set-method [#method# #val#]

declare a method and link it to a named NL procedure

```
syntax:  #set-method  class-name.method-name  proc-name
eg:      #set-method  ant.go  proc-name
```

NLoops calling primitives – instance context

These primitives are all used in the context of the current NLoops instance (ie: typically while running a procedure for a turtle which has spawned an NLoops instance for itself or while running an NLoops method).

to # [#method#]

call the named method for the current instance (without providing arguments)
eg: `#"jump"`

to #do [#method# #args#]

call the named method for the current instance (providing arguments in a list)
eg: `#do"move" [3 4]`
Note that method arguments are always provided in a list but the NL procedure definition of the method treats arguments in the normal way. This applies to all passing of arguments to methods.

to-report #report [#method# #args#]

call the named reporter method for the current instance (providing arguments in a list)
eg: `#report"max" [3 4]`

to-report #<= [#slot#]

report the value of the named slot for the current instance
eg: `#<="x"`

to #=> [#slot# #val#]

set the value of the named slot for the current instance
eg: `#=>"x" 5`

duplicates (as requested)

to-report #get [#slot#]

as #<=

to #set [#slot# #val#]

as #=>

NLoops calling primitives – external context

These primitives are (roughly) equivalent the similar *internal context* forms except that they act on a specific, named instance. For the sake of the examples we assume there is an instance named "sue".

to #do-ob [#who# #method# #args#]

call the named method for the named instance (providing arguments in a list)
eg: `#do-ob "sue" "move" [3 4]`

to-report #report-ob [#who# #method# #args#]

call the named reporter method for the named instance (providing arguments in a list)
eg: `#report-ob "sue" "max" [3 4]`

to-report #get-ob [#who# #slot#]

report the value of the named slot for the named instance
eg: `#get-ob "sue" "x"`

to #set-ob [#who# #slot# #val#]

get the value of the named slot for the named instance
eg: `#set-ob "sue" "x" 5`

call by state

call by state primitives are described in a separate document

to #do-ob-modified [#who# #root# #slots# #args#]

to-report #report-ob-modified [#who# #root# #slots# #args#]

to #do-modified [#root# #slots# #args#]

to-report #report-modified [#root# #slots# #args#]

to #do-p-modified [#who# #root# #slots# #args#]

to-report #report-p-modified [#who# #root# #slots# #args#]

killers

to #die

ask the current object to "die", ie: remove all reference to it from the NLoops system
eg: `#die`

to #kill [#who#]

ask a named object to die
eg: `#kill "sue"`

other functions

to-report #class-of [#who#]

reports the NLoops class of the named object
eg: `#class-of "sue"`

to-report #my-class

reports the NLoops class of the current object
eg: `#my-class`

to-report #self

reports the identity of the current NLoops instance (generates an error if there is no instance in context)
eg: `#self`

to-report #ob-has-slot? [#who# #slot#]

reports true/false according to whether the named object has (or can inherit) the named slot
eg: `#ob-has-slot? "sue" "age"`

to-report #has-slot? [#slot#]

reports true/false according to whether the current object has (or can inherit) the named slot
eg: `#has-slot? "age"`

to-report #whoami

reports the identity of the current NLoops instance (or, if there is no instance in context, the "who" identity of the current context)
eg: `#whoami`

misc

to #show-objects
print the #objects table which contains information about all objects, classes & methods

to-report #valof [var-string]
take value of variable named in string, eg if X = 5, then (#valof "X") -> 5

#nil
variable with the value of an empty list

to-report #null? [#x#]
a test for an empty list

errors & debugging

#debug#
variable. When set to true debug information will be printed by the #debug procedure

to #debug [#str#]
prints #debug information when the #debug# variable is true

to #error [#str# #stuff#]
NL doesn't have an error primitive, this =p is used as a compromise to generate pseudo NLoops errors