

File created: 28-Mar-89 11:18:05 {NB:PARC:XEROX}<NOTECARDS>1.3M>LIBRARY>NCPATHPARSE.;1

previous date: 5-Nov-86 16:38:38 {QV}<NOTECARDS>1.3L>LIBRARY>NCPATH>NCPATHPARSE.;1

Read Table: INTERLISP

Package: INTERLISP

Format: XCCS

::
:: Copyright (c) 1986, 1989 by Xerox Corporation. All rights reserved.

(RPAQQ NCPATHPARSECOMS

```
[(* * This file is intended to be the genesis/testbed for ideas about parsing the user language for a
path-specification facility into an NCPATHFSM for the functions of NCPATH to use. The first list of
functions are the current ones in use. Right now, the system only parses linear specifications.)
(FNS NCPATHParse NCPATHParse.Path NCPATHParse.PathStep NCPATHParse.PathStepDescriptor
NCPATHParse.LiteralPathDescription NCPATHParse.FunctionalPathDescription NCPATHParse.CreateFSMNode
NCPATHParse.CreatePredicateForm NCPATHParse.RepeaterExpression NCPATHParse.RepeaterToken
NCPATHParse.LimitedRepeaterToken NCPATHParse.LoopDecision NCPATHParse.CreateLoop
NCPATHParse.PathStepOperation)
(FNS NCPATHParse.FunctionP NCPATHParse.CheckAndComputeFlag NCPATHParse.CombinePredicates)
(FNS NCPATHParse.CombinationExpressions)
(FNS NCPATHParse.CoalesceStates NCPATHParse.CoalesceRepeaterStates)
(* * The following functions are utilities from other sources)
(FNS NAND LOGICAL.EQUAL)
(DECLARE%: DONTEVAL@LOAD DOEVAL@COMPILE DONTCOPY COMPILEVAR (ADDVARS (NLAMA NAND)
(NLAML)
(LAMA LOGICAL.EQUAL))
```

(* * This file is intended to be the genesis/testbed for ideas about parsing the user language for a path-specification facility into an NCPATHFSM for the functions of NCPATH to use. The first list of functions are the current ones in use. Right now, the system only parses linear specifications.)

(DEFINEQ

(NCPATHParse

[LAMBDA (Expression NoteFile DepthLimit) (* Newman " 4-Nov-86 09:49")

(* * This function is intended to be the top-level function that parses an expression into an FSM that NCPATH.FSM.PathCollect can deal with.)

```
(if (NCP.OpenNoteFileP NoteFile)
then [LET [(FirstState (NCPATHParse.CoalesceStates (NCPATHParse.Path Expression NoteFile)
(if FirstState
then (create NCPATHFSM
InitialState _ FirstState
CurrentState _ FirstState
AbsoluteDepthLimit _ (OR DepthLimit 0)
else (NCP.ReportError NoteFile " is not an open notefile."))
```

(NCPATHParse.Path

[LAMBDA (Expression NoteFile) (* Newman " 5-Nov-86 16:21")

(* * This function parses an individual path as a list of steps or repeater expressions.)

```
(if (NULL Expression)
then NIL
else (OR (NCPATHParse.PathStep Expression NoteFile)
[if (LISTP Expression)
then (for Item in Expression collect (OR (NCPATHParse.Path Item NoteFile)
(RETURN NIL)
(if (EQUAL 1 (LENGTH Expression))
then (NCPATHParse.Path (CAR Expression)
NoteFile)
(if (EQUAL 2 (LENGTH Expression))
then (NCPATHParse.RepeaterExpression (CADR Expression)
(MKLIST (NCPATHParse.Path (CAR Expression)
NoteFile))
```

(NCPATHParse.PathStep

[LAMBDA (Expression NoteFile) (* Newman " 4-Nov-86 10:19")

(* * This function parses an individual step of a path. A Step is either a path step descriptor or some combination of descriptors.)

```
(if (NULL Expression)
then NIL
else (OR (NCPATHParse.PathStepDescriptor Expression NoteFile)
[if (MEMBER (CAR Expression)
' (OR AND NOT))
then (NCPATHParse.PathStepOperation Expression NoteFile)]
```

(NCPPathParse.PathStepDescriptor

[LAMBDA (Expression NoteFile)

(* Newman " 5-Nov-86 16:21")

(* * Parses a path step descriptor. A descriptor is a literal descriptor, a functional descriptor, a "don't care" expression, or a variable pointing to a descriptor of one of the other types. The variable is checked for immediate circularity, but not for indirect circularity; either of these conditions could cause an infinite loop.)

```
(if (NULL Expression)
  then NIL
  else (OR (NCPPathParse.LiteralPathDescription Expression NoteFile)
           (NCPPathParse.FunctionalPathDescription Expression)
           (if (EQUAL Expression 'ANY)
               then (NCPPathParse.CreateFSMNode (FUNCTION TRUE)
                                                    T T T)
               (if [AND (BOUNDP Expression)
                       (NOT (EQUAL Expression (EVAL Expression))
                           then (NCPPathParse.PathStepDescriptor (EVAL Expression)
                                                                    NoteFile]))
```

(NCPPathParse.LiteralPathDescription

[LAMBDA (Expression NoteFile)

(* Newman " 4-Nov-86 13:11")

(* * This function parses literal path descriptions. These are path descriptions that are valid link labels or valid card types with appropriate prefixes as described in the grammar.)

```
(OR (if (NCP.ValidLinkLabel Expression NoteFile)
        then (NCPPathParse.CreateFSMNode Expression T T))
    (if (AND (EQUAL (SUBATOM Expression 1 1)
                    '@')
          (NCP.ValidCardType (SUBATOM Expression 2 -1)))
        then (NCPPathParse.CreateFSMNode (SUBATOM Expression 2 -1)
                                           NIL T))
    (if (AND (EQUAL (SUBATOM Expression 1 1)
                    '_')
          (NCP.ValidLinkLabel (SUBATOM Expression 2 -1)
                               NoteFile))
        then (NCPPathParse.CreateFSMNode (SUBATOM Expression 2 -1)
                                           T NIL))
    (if (AND (MEMBER (SUBATOM Expression 1 2)
                    '(_@
                      @_))
          (NCP.ValidCardType (SUBATOM Expression 3 -1)))
        then (NCPPathParse.CreateFSMNode (SUBATOM Expression 3 -1)
                                           T NIL))
```

(NCPPathParse.FunctionalPathDescription

[LAMBDA (Expression)

(* Newman " 5-Nov-86 16:21")

(* * This function parses functional path descriptions. These are path descriptions which are the names of Lisp predicates with the appropriate prefixes.)

```
(OR (if (AND (EQUAL (SUBATOM Expression 1 1)
                    '%#')
          (NCPPathParse.FunctionP (SUBATOM Expression 2 -1)))
        then (NCPPathParse.CreateFSMNode (SUBATOM Expression 2 -1)
                                           T T T))
    (if (AND (MEMBER (SUBATOM Expression 1 2)
                    '(@# %#@)')
          (NCPPathParse.FunctionP (SUBATOM Expression 3 -1)))
        then (NCPPathParse.CreateFSMNode (SUBATOM Expression 3 -1)
                                           NIL T T))
    (if (AND (MEMBER (SUBATOM Expression 1 2)
                    '(_#
                      %#_))
          (NCPPathParse.FunctionP (SUBATOM Expression 3 -1)))
        then (NCPPathParse.CreateFSMNode (SUBATOM Expression 3 -1)
                                           NIL T T))
    (if (AND (MEMBER (SUBATOM Expression 1 3)
                    '(_#@
                      _@#
                      @_#
                      @#_ %#@_ %#_@))
          (NCPPathParse.FunctionP (SUBATOM Expression 4 -1)))
        then (NCPPathParse.CreateFSMNode (SUBATOM Expression 4 -1)
                                           NIL T T))
```

(NCPPathParse.CreateFSMNode

[LAMBDA (Item Link/CardFlag DirectionFlag FunctionFlag)

(* Newman " 4-Nov-86 11:37")

(* * This function creates an instance of the NCPPathFSMNode data type according to the arguments passed in. Its chief purpose is to call NCPPathParse.CreatePredicateForm)

```
(create NCPPathFSMNode
  Predicate _ (if FunctionFlag
              then Item
              else (NCPPathParse.CreatePredicateForm Item Link/CardFlag))
  Card/Link _ Link/CardFlag
  Direction _ DirectionFlag]
```

(NCPPathParse.CreatePredicateForm

[LAMBDA (Type Link/CardFLAG) (* Newman " 4-Nov-86 08:52")

(* * This function creates a LAMBDA expression that will serve as a predicate. See PARSE.COMBINE.PREDICATES.)

```
`(LAMBDA (Item)
  (EQUAL (% (COND
            (Link/CardFLAG (FUNCTION NCP.LinkType))
            (T 'NCP.CardType))
         Item)
   (QUOTE %, Type])
```

(NCPPathParse.RepeaterExpression

[LAMBDA (RepeaterExpression LoopSteps) (* Newman "18-Mar-86 09:21")

(* * This function parses repeater expressions. Unfortunately, I have not yet determined how to deal with limited repeater expressions.)

```
(OR (NCPPathParse.RepeaterToken RepeaterExpression LoopSteps)
    (NCPPathParse.LimitedRepeaterToken RepeaterExpression LoopSteps)
    (NCP.ReportError " Repeater Expression mucked up " RepeaterExpression)
    (BREAK1 NIL T])
```

(NCPPathParse.RepeaterToken

[LAMBDA (RepeaterExpression LoopSteps) (* Newman "19-Mar-86 15:41")

(* * creates a repeater loop with no limit, or with infinite limit)

```
(if (EQUAL RepeaterExpression '*')
    then (NCPPathParse.CreateLoop LoopSteps 0 0)
    elseif (EQUAL RepeaterExpression '+')
    then (NCPPathParse.CreateLoop LoopSteps 1 0])
```

(NCPPathParse.LimitedRepeaterToken

[LAMBDA (RepeaterExpression LoopSteps) (* Newman "18-Mar-86 09:20")

(* * This function parses repeater tokens that have an integral limit.)

```
(OR (NCPPathParse.LoopDecision RepeaterExpression LoopSteps 1 '+')
    (NCPPathParse.LoopDecision RepeaterExpression LoopSteps 0 '*')
    (NCP.ReportError " Repeater Expression mucked up " RepeaterExpression)
    (BREAK1 NIL T])
```

(NCPPathParse.LoopDecision

[LAMBDA (Expression LoopSteps MinimumRepeat Symbol) (* Newman "18-Mar-86 09:18")

(* * This function decides what kind of loop is to be created and then creates it.)

```
(if (EQUAL Symbol (SUBATOM Expression 1 1))
    then (NCPPathParse.CreateLoop LoopSteps (OR MinimumRepeat 0)
        (OR (NUMBERP (SUBATOM Expression 2 -1))
            (NCP.ReportError " Repeater Expression mucked up " Expression)))
    elseif (NUMBERP Expression)
    then (NCPPathParse.CreateLoop LoopSteps 1 Expression])
```

(NCPPathParse.CreateLoop

[LAMBDA (LoopSteps MinTimes MaxTimes) (* Newman "15-Mar-86 12:17")

(* * Here we create a loop expression that PARSE.CoalesceStates understands.)

```
(LIST 'PARSE.DO.REPEAT MinTimes MaxTimes LoopSteps])
```

(NCPPathParse.PathStepOperation

[LAMBDA (Expression NoteFile) (* Newman " 4-Nov-86 14:42")

(* * Here we parse combinations of pathsteps.)

```
(NCPPathParse.CombinationExpressions (for Step in (CDR Expression) collect (OR (NCPPathParse.PathStep Step
                                                                              NoteFile)
                                                                              (RETURN NIL)))
  (SELECTQ (CAR Expression)
    (AND 'AND)
```

```

(OR 'OR)
(NOT 'NAND)
(NCP.ReportError " Operator not AND, OR, or NOT in NCPPathParse.PathStepOperation. ")
)

```

(DEFINEQ

(NCPPathParse.FunctionP

[LAMBDA (Function) (* Newman " 4-Nov-86 11:39")

(* * This function determines whether or not a function passed in is an appropriate function for use by NCPPath functions. The criterial is rather limited at the moment, including only that the function must be defined, must accept only one argument, and must not be an NLAMBDA function.)

```

(AND (GETD Function)
(EQUAL 1 (NARGS Function))
(NOT (NLAMBDAFNP Function]))

```

(NCPPathParse.CheckAndComputeFlag

[LAMBDA (StepList FlagName) (* Newman " 5-Nov-86 10:22")

(* * This function computes the Direction and Card/Link flags when parsing a combination FSMNode.)

```

(if [APPLY 'LOGICAL.EQUAL (for Step in StepList collect (EVAL `(fetch (NCPPathFSMNode %, FlagName) of Step]
then [EVAL `(fetch (NCPPathFSMNode %, FlagName) of (CAR StepList]
else (NCP.ReportError " Flags don't all match " StepList)
'ERROR])

```

(NCPPathParse.CombinePredicates

[LAMBDA (StepList Operation) (* Newman "14-Mar-86 16:07")

(* * This function builds the LAMBDA expression that will be the predicate in a combination FSMNode. I wish we could compile the LAMBDA expression for speed. Perhaps we could use a GENSYM, and compile the function that way?)

```

` (LAMBDA (Item)
, (CONS Operation (for I in StepList collect (LIST (fetch (NCPPathFSMNode Predicate) of I)
'Item]))
)

```

(DEFINEQ

(NCPPathParse.CombinationExpressions

[LAMBDA (StepList Operation) (* Newman "24-Mar-86 15:19")

(* * This function combines pathstep specifications using AND, OR, or NOT.)

```

(if (NULL StepList)
then NIL
else (LET [(Card/LinkFlag (NCPPathParse.CheckAndComputeFlag StepList 'Card/Link))
(DirectionFlag (NCPPathParse.CheckAndComputeFlag StepList 'Direction))
(if (OR (EQUAL Card/LinkFlag 'ERROR)
(EQUAL DirectionFlag 'ERROR))
then NIL
else (create NCPPathFSMNode
Predicate _ (NCPPathParse.CombinePredicates StepList Operation)
Card/Link _ Card/LinkFlag
Direction _ DirectionFlag])
)
)

```

(DEFINEQ

(NCPPathParse.CoalesceStates

[LAMBDA (NodeList) (* Newman "18-Mar-86 09:28")

(* * This function takes a lisp-style list of NCPPathFSMNodes and turns them into a linked list. The linked list will include circularities where repeater expressions exist.)

```

(if (NULL NodeList)
then NIL
elseif (EQUAL (TYPENAME NodeList)
'NCPPathFSMNode)
then NodeList
elseif (EQUAL (TYPENAME (CAR NodeList))
'NCPPathFSMNode)
then (replace (NCPPathFSMNode NextNodes) of (CAR NodeList) with (NCPPathParse.CoalesceStates (CDR NodeList)))
(CAR NodeList)
elseif (AND (LISTP NodeList)
(EQUAL (CAR NodeList)
'PARSE.DO.REPEAT))
then (NCPPathParse.CoalesceRepeaterStates (MKLIST (CADDR NodeList))
NIL
)
)

```

FUNCTION INDEX

LOGICAL.EQUAL	5	NCPATHPARSE.FunctionalPathDescription	2
NAND	5	NCPATHPARSE.FunctionP	4
NCPATHPARSE	1	NCPATHPARSE.LimitedRepeaterToken	3
NCPATHPARSE.CheckAndComputeFlag	4	NCPATHPARSE.LiteralPathDescription	2
NCPATHPARSE.CoalesceRepeaterStates	5	NCPATHPARSE.LoopDecision	3
NCPATHPARSE.CoalesceStates	4	NCPATHPARSE.Path	1
NCPATHPARSE.CombinationExpressions	4	NCPATHPARSE.PathStep	1
NCPATHPARSE.CombinePredicates	4	NCPATHPARSE.PathStepDescriptor	2
NCPATHPARSE.CreateFSMNode	2	NCPATHPARSE.PathStepOperation	3
NCPATHPARSE.CreateLoop	3	NCPATHPARSE.RepeaterExpression	3
NCPATHPARSE.CreatePredicateForm	3	NCPATHPARSE.RepeaterToken	3

VARIABLE INDEX

NCPATHPARSECOMS	1
-----------------------	---
