File created: 11-May-93 13:12:45 **{DSK}<usr>local>lde>loops>users>rules>RULESD.;1**

changes to: (VARS RULESDCOMS)

Read Table: XCL

Package: INTERLISP

Format: XCCS

```
(RPAQQ RULESDCOMS ((DECLARE\: DONTCOPY (PROP MAKEFILE-ENVIRONMENT RULESD))
                                                                             ; Copyright (c) 1982 by Xerox Corporation
                                                                             ; Written in August 1982 by Mark Stefik, Alan Bell, and Danny
                                                                             ; Bobrow
                                                                             ; Fns for getting declarations in LOOPS RuleSets.
                      (FNS * RULEDECLFNS)))

(DECLARE\: DONTCOPY)

;; Copyright (c) 1982 by Xerox Corporation

;; Written in August 1982 by Mark Stefik, Alan Bell, and Danny Bobrow

;; Fns for getting declarations in LOOPS RuleSets.

(RPAQQ RULEDECLFNS (|AssocAuditSpecification| |FlushComment?| |GetAuditClass| |GetCompilerOptions|
                          |GetControlType| |GetDebugVars| |GetMetaAssns| |GetOneShotFlg| |GetProgVars|
                          |GetRSAllDeclString| |GetRSDeclString| |GetRuleClass| |GetRuleMetaDecls|
                          |GetRuleSetArgs| |GetRuleSetDeclarations| |GetRuleSetTemplate| |GetTaskVars|
                          |GetTempVars| |GetWhileCondition| |GetWorkSpaceClass| |InterpretCompilerOptions|
                          |SpaceOrItem|))

(DEFINEQ

(|AssocAuditSpecification|
  (LAMBDA (|auditSpec|)                                                      (* |mjs:| "11-FEB-83 17:58")

         (* * |Converts| |an| |Audit| |specification| |into| |an| |association| |list| |of| |the| |form| -
         ((|varName1| . |valueExpr1|) (|varName2| . |valueExpr2|)) -
         |where| |each| |valueExpr| |is| |an| |expression| |to| |be| |compiled| |later| |by| |AuditRecordCodeGen.|)

    (PROG (|assocList| |varName| |varExpr|)

         (* * |Collect| |the| |association| |list.|)

         (|while| (AND |auditSpec| (NOT (EQ |rpar| (CAR |auditSpec|))))
            |do| (COND
                    ((AND (SETQ |varName| (|pop| |auditSpec|))
                          (LITATOM (CAR |auditSpec|)))               (* |Check| |the| |varName.|)
                     T)
                    (T (|FlushRule| "Strange variable name: " |varName| " in meta-Assignment Statement.")
                       (GO |Done|)))
                 (COND
                    ((EQ (CAR |auditSpec|)
                         |leftArrow|)                                (* |Check| |for| _)
                     (|pop| |auditSpec|))
                    (T (|FlushRule| "Missing" "_" "in Audit Specification")
                       (GO |Done|)))
                 (SETQ |varExpr| (|pop| |auditSpec|))                (* |Add| |to| |the| |Assoc| |list.|)
                 (SETQ |assocList| (CONS (CONS |varName| |varExpr|)
                                         |assocList|)))
      |Done|
         (RETURN |assocList|))))

(|FlushComment?|
  (LAMBDA (|noCheckFlg|)                                                     (* |mjs:| "16-FEB-83 15:52")

         (* * |Removes| |comments| |during| |RuleSet| |compilation.|)

    (PROG (|token| |doneFlg|)
         (|while| (OR |noCheckFlg| (AND (EQ (CAR |ruleSetTokens|)
                                            |lpar|)
                                        (EQ (CADR |ruleSetTokens|)
                                            |asterisk|)))
            |do|                                                     (* |pop| |first| |left| |paren.|)
                 (SETQ |noCheckFlg| NIL)
                 (|pop| |ruleSetTokens|)
                 (|repeatuntil| |doneFlg| |do| (SETQ |token| (|pop| |ruleSetTokens|))
                                               (COND
                                                  ((NULL |token|)
                                                   (SETQ |doneFlg| T))
                                                  ((EQ |token| |lpar|)
                                                                     (* |Recur| |for| |embedded| |parens.|)
                                                   (|FlushComment?| '|NoCheckFlg|))
                                                  ((EQ |token| |rpar|)
                                          (SETQ |doneFlg| T))))
                (SETQ |doneFlg| NIL))
          (RETURN NIL))))
```

(|GetAuditClass|

```
  (LAMBDA (|self|)                                                 ; Edited 10-May-88 18:39 by JAMES.PA

          (* * |Parses| |the| |Audit| |Class| |declaration| |at| |the| |beginning| |of| \a |RuleSet.|
          |Argument| |self| |is| |the| |RuleSet.|)

    (PROG (|auditClassName|)

          (* * |Flush| |any| |leading| |comments| |and| |verify| |statement| |type.|)

          (COND
             ((OR (NEQ (CAR |ruleSetTokens|)
                       '|Audit|)
                  (NEQ (CADR |ruleSetTokens|)
                       '|Class|))
              (|FlushRule| "Bad Audit Class Statement.")
              (GO |done|)))

          (* * |Pop| |the| |Audit,| |Class,| |and| |colon| |tokens.|)

          (|pop| |ruleSetTokens|)
          (|pop| |ruleSetTokens|)
          (|pop| |ruleSetTokens|)

          (* * |Get| |the| |Class.|)

          (SETQ |auditClassName| (COND
                                    ((NEQ (CAR |ruleSetTokens|)
                                          |semicolon|)
                                     (|pop| |ruleSetTokens|))))

          (* * |pop| |the| |semicolon.|)

          (|pop| |ruleSetTokens|)
      |done|
          (SETQ |rsAuditClass| (|GetClassRec| |auditClassName|))
          (COND
             ((AND |auditClassName| (NULL |rsAuditClass|))
              (|FlushRule| "Audit Class not recognized: " |auditClassName|)))
          (COND
             (|rsAuditClass| (SETQ |auditSpecification| (|GetClassValue| |rsAuditClass| '|metaAssns|)))))))
```

(|GetCompilerOptions|

```
  (LAMBDA NIL                                                      (* |dgb:| "17-Feb-84 14:02")

          (* * |Parses| |the| |Compiler| |Options| |declaration| |at| |the| |beginning| |of| \a |RuleSet.|
          |Argument| |self| |is| |the| |RuleSet.| |Skip| |if| |^userCompilerOptions| |is| |set.|)

    (PROG (|options| (|possibleOptions| (CONSTANT '(T B A S BT TT PR LC))))

          (* * |Verify| |statement| |type.|)

          (COND
             ((NEQ (CAR |ruleSetTokens|)
                   '|Compiler|)
              (|FlushRule| "No Compiler Options Statement.")
              (RETURN)))

          (* * |Pop| |the| |Compiler,| |Options,| |and| |colon| |tokens.|)

          (|pop| |ruleSetTokens|)
          (|pop| |ruleSetTokens|)
          (|pop| |ruleSetTokens|)

          (* * |Collect,| |uppercase,| |and| |check| |the| |options.|)

          (SETQ |options| (|while| (NEQ (CAR |ruleSetTokens|)
                                        |semicolon|)
                             |collect| (U-CASE (|pop| |ruleSetTokens|))))
          (COND
             ((SETQ |parseErrorFlg| (|for| |option| |in| |options| |thereis| (NOT (FMEMB |option| |possibleOptions|)))
               )
              (|FlushRule| "Unrecognized compiler option=" |parseErrorFlg|)))

          (* * |pop| |the| |semicolon.|)

          (|pop| |ruleSetTokens|)
          (|InterpretCompilerOptions| |options|))))
```

(|GetControlType|

```
  (LAMBDA NIL                                                            (* |dgb:| "17-Feb-84 18:03")

          (* * |Parses| |the| |Control| |Structure| |declaration| |at| |the| |beginning| |of| \a |RuleSet.|
          |Argument| |self| |is| |the| |RuleSet.|)

    (PROG (|type| (|controlTypes| (CONSTANT (LIST 'DO1 'DOALL 'WHILE1 'WHILEALL 'FOR1 'FORALL 'DONEXT
                                                  'WHILENEXT))))

          (* * |Flush| |any| |leading| |comments| |and| |verify| |statement| |type.|)

          (COND
             ((OR (NEQ (CAR |ruleSetTokens|)
                       '|Control|)
                  (NEQ (CADR |ruleSetTokens|)
                       '|Structure|))
              (|FlushRule| "No Control Structure Statement.  --- Assuming DOALL.")
              (SETQQ |type| DOALL)
              (GO |done|)))

          (* * |Pop| |the| |Control,| |Structure,| |and| |colon| |tokens.|)

          (|pop| |ruleSetTokens|)
          (|pop| |ruleSetTokens|)
          (|pop| |ruleSetTokens|)

          (* * |Get| |and| |Check| |the| |control| |type.|)

          (SETQ |type| (U-CASE (|pop| |ruleSetTokens|)))
          (COND
             ((NOT (FMEMB |type| |controlTypes|))
              (|FlushRule| "UnRecognized ControlType=" |type|)
              (SETQ |type| 'DOALL)))

          (* * |pop| |the| |semicolon.|)

          (|pop| |ruleSetTokens|)
      |done|
          (SETQ |controlType| |type|))))
```

(|GetDebugVars|

```
  (LAMBDA NIL                                                            (* |mjs:| "12-FEB-83 15:59")

          (* * |Parses| |the| |Debug| |Vars| |declaration| |at| |the| |beginning| |of| \a |RuleSet.|
          |Argument| |self| |is| |the| |RuleSet.|)

    (PROG (|vars|)                                                       (* |Flush| |any| |leading| |comments.|)
          (|FlushComment?|)
          (COND
             ((OR (NEQ (CAR |ruleSetTokens|)
                       '|Debug|)
                  (NEQ (CADR |ruleSetTokens|)
                       '|Vars|))
              (|FlushRule| "Strange Debug Vars Statement.")
              (GO |done|)))

          (* * |Pop| |the| |Debug,| |Vars,| |and| |colon| |tokens.|)

          (|pop| |ruleSetTokens|)
          (|pop| |ruleSetTokens|)
          (|pop| |ruleSetTokens|)

          (* * |Collect| |the| |Debug| |Vars.|)

          (SETQ |vars| (|while| (NEQ (CAR |ruleSetTokens|)
                                     |semicolon|)
                          |collect| (|pop| |ruleSetTokens|)))

          (* * |pop| |the| |semicolon.|)

          (|pop| |ruleSetTokens|)
      |done|
          (SETQ |debugVars| |vars|))))
```

(|GetMetaAssns|

```
  (LAMBDA NIL                                                            (* |mjs:| "12-FEB-83 17:18")

          (* * |Parses| |the| |Meta| |Assignments| |declaration| |at| |the| |beginning| |of| \a |RuleSet.|
          |Argument| |self| |is| |the| |RuleSet.|)

    (PROG (|auditTokens|)                                                (* |Flush| |any| |leading| |comments.|)
          (|FlushComment?|)
          (COND
             ((OR (NEQ (CAR |ruleSetTokens|)
                       '|Meta|)
                  (NEQ (CADR |ruleSetTokens|)
                    '|Assignments|))
              (|FlushRule| "Strange Meta Assignments Statement.")
              (GO |done|)))

          (* * |Pop| |the| |Meta,| |Assignments,| |and| |colon| |tokens.|)

          (|pop| |ruleSetTokens|)
          (|pop| |ruleSetTokens|)
          (|pop| |ruleSetTokens|)

          (* * |Collect| |the| |Audit| |Specification.|)

          (SETQ |auditTokens| (|while| (NEQ (CAR |ruleSetTokens|)
                                            |semicolon|)
                                 |collect| (|pop| |ruleSetTokens|)))

          (* * |pop| |the| |semicolon.|)

          (|pop| |ruleSetTokens|)                              (* |Set| |the| |global| |variable.|)
      |done|
          (SETQ |rsAuditSpecification| |auditTokens|))))

(|GetOneShotFlg|
  (LAMBDA NIL                                                  (* |dgb:| "21-Feb-84 11:55")

          (* |Name| |and| |assign| \a |TaskVar| |for| |one-shot| |rules| |in| |cyclic| |control| |structures.|
          |Sets| |global| |variable| |oneShotFlg| |to| |the| |name| |of| |new| |Task| |variable.|
          |Variable| |is| |used| |in| |CompileLHS| |and| |CompileRHS| |and| |added| |to| |either| |the| |task| |Vars| |or| |the| |rule| |Vars|
          |of| |the| |RuleSet.| |Subroutine| |of| |CompileRule.|)

    (PROG NIL
          (COND
             ((FMEMB |bang| |metaTokens|)                      (* |One| |shot| |Bang|)
              (SETQ |oneShotBangFlg| |oneBang|)
              (SETQ |metaTokens| (DREMOVE |bang| |metaTokens|)))
             (T                                                (* |One| |shot.|)
                (SETQ |oneShotBangFlg| NIL)))
          (SETQ |metaTokens| (DREMOVE 1 |metaTokens|))
          (COND
             ((AND (NULL |rsTaskFlg|)
                   (NOT (FMEMB |controlType| |cyclicControlStructures|)))
                                                               (* |Ignore| |1-shot| |indication| |if| |not| |cyclic| |and| |no| |tasking.|)
              (SETQ |oneShotFlg| NIL)
              (RETURN)))                                       (* |Compute| |the| |variable| |name| |for| |the| |rule.|)
          (SETQ |oneShotFlg| (PACK* '|^triedRule| |ruleNumber|))
          (COND
             (|rsTaskFlg|                                      (* I\f |Tasking,| |Add| |to| |taskVars.|)
                    (SETQ |rsInternalTaskVars| (NCONC1 |rsInternalTaskVars| |oneShotFlg|)))
             (T                                                (* |otherwise| |add| |to| |tempVars.|)
                (SETQ |rsInternalTempVars| (NCONC1 |rsInternalTempVars| |oneShotFlg|)))))))

(|GetProgVars|
  (LAMBDA NIL                                                  (* |dgb:| "21-Feb-84 15:26")

          (* * |Subroutine| |of| |CompileRuleList.| |Returns| |the| PROG |vars| |associated| |with| |the| |template| |for| \a |particular|
          |controlType.|)

    (PROG (|progVars| |allTemps|)
          (SETQ |allTemps| (APPEND |tempVars| |rsInternalTempVars|))
          (SETQ |progVars| (SELECTQ |controlType|
                               ((FOR1 FORALL DO1 DOALL)
                                    (CONS '|^value| |allTemps|))
                               (DONEXT (APPEND '(|^value| |ruleApplied|)
                                              |allTemps|))
                               (WHILE1 (COND
                                          (|rsRuleAppliedFlg| (APPEND '(|^value| |ruleApplied|)
                                                                     |allTemps|))
                                          (T (CONS '|^value| |allTemps|))))
                               (WHILEALL (COND
                                            (|rsRuleAppliedFlg| (APPEND '(|^value| |^prevValue| |ruleApplied|)
                                                                       |allTemps|))
                                            (T (CONS '|^value| |allTemps|))))
                               (WHILENEXT (APPEND '(|^value| |ruleApplied| |^firstRuleTried|)
                                                 |allTemps|))
                               NIL))
          (COND
             (|rsSomeRuleAuditFlg|                             (* |Add| \a |variable| |for| |the| |audit| |records| |if| |at| |least| |one|
                    |rule| |was| |audited.|)
                    (SETQ |progVars| (CONS '|^auditRecord| |progVars|))))
          (RETURN |progVars|))))

(|GetRSAllDeclString|
  (LAMBDA NIL                                                  (* |dgb:| "21-Feb-84 11:43")
            (* * |Returns| \a |string| |representation| |of| |the| |declarations| |for| |the| |RuleSet| |currently| |being| |edited,| |that| |is,| |the|
            |one| |for| |which| |the| |rule| |set| |global| |variables| |have| |been| |set.|
            |Should| |be| |called| AFTER |GetRuleSetDeclarations.|)

        (PROG (|str| (SCRLF (CONSTANT ";
                                  ")))
            (SETQ |str| (CONCAT "WorkSpace Class: " (|SpaceOrItem| |wsClass|)
                              SCRLF "Compiler Options: " (|SpaceOrItem| |rsCompilerOptions|)
                              SCRLF "Args: " (|SpaceOrItem| |rsArgs|)
                              SCRLF "Temporary Vars: " (|SpaceOrItem| |tempVars|)
                              SCRLF "Control Structure: " (|SpaceOrItem| |controlType|)
                              SCRLF "Iteration Condition: " (|SpaceOrItem| (|for| |term| |in| |rsWhileCondition|
                                                                          |collect| (|UnParseTerm| |term|)))
                              SCRLF "Audit Class: " (|SpaceOrItem| |rsAuditClass|)
                              SCRLF "Meta Assignments: " (|SpaceOrItem| |rsAuditSpecification|)
                              SCRLF "Rule Class: " (|SpaceOrItem| |rsRuleClass|)
                              SCRLF "Debug Vars: " (|SpaceOrItem| |debugVars|)
                              SCRLF "********"))
            (RETURN |str|))))
```

```
(|GetRSDeclString|
  (LAMBDA (|self|)                                                  (* |dgb:| "21-Feb-84 11:43")

            (* * |Returns| \a |string| |representation| |of| |the| |declarations| |for| |the| |RuleSet| |self.|)

        (PROG (|str| (SCRLF (CONSTANT ";
                                  ")))
            (SETQ |str| (CONCAT "WorkSpace Class: " (|SpaceOrItem| (@ |workSpace|))
                              SCRLF "Compiler Options: " (|SpaceOrItem| (@ |compilerOptions|))
                              SCRLF "Temporary Vars: " (|SpaceOrItem| (@ |tempVars|))
                              SCRLF "Control Structure: " (@ |controlStructure|)
                              SCRLF))
            (COND
               ((@ |whileCondition|)
                (SETQ |str| (CONCAT |str| "Iteration Condition: " (|SpaceOrItem| (@ |whileCondition|))
                                  SCRLF))))
            (COND
               ((@ |args|)
                (SETQ |str| (CONCAT |str| "Args: " (|SpaceOrItem| (@ |args|))
                                  SCRLF))))
            (COND
               ((AND (@ |auditClass|)
                     (NEQ (@ |auditClass|)
                          ($ |StandardAuditRecord|)))
                (SETQ |str| (CONCAT |str| "Audit Class: " (|SpaceOrItem| (@ |auditClass|))
                                  SCRLF))))
            (COND
               ((@ |metaAssignments|)
                (SETQ |str| (CONCAT |str| "Meta Assignments: " (|SpaceOrItem| (@ |metaAssignments|))
                                  SCRLF))))
            (COND
               ((AND (@ |ruleClass|)
                     (NEQ (@ |ruleClass|)
                          ($ |Rule|)))
                (SETQ |str| (CONCAT |str| "Rule Class: " (|SpaceOrItem| (@ |ruleClass|))
                                  SCRLF))))
            (COND
               ((@ |debugVars|)
                (SETQ |str| (CONCAT |str| "Debug Vars: " (|SpaceOrItem| (@ |debugVars|))
                                  SCRLF))))
            (RETURN |str|))))
```

```
(|GetRuleClass|
  (LAMBDA NIL                                                       (* |mjs:| "12-FEB-83 16:44")

            (* * |Parses| |the| |Rule| |Class| |declaration| |at| |the| |beginning| |of| \a |RuleSet.|
            |Argument| |self| |is| |the| |RuleSet.|)

        (PROG (|ruleClassName|)

            (* * |Flush| |any| |leading| |comments| |and| |verify| |statement| |type.|)

            (COND
               ((OR (NEQ (CAR |ruleSetTokens|)
                         '|Rule|)
                    (NEQ (CADR |ruleSetTokens|)
                         '|Class|))
                (|FlushRule| "Bad Rule Class Statement.")
                (GO |done|)))

            (* * |Pop| |the| |Rule,| |Class,| |and| |colon| |tokens.|)

            (|pop| |ruleSetTokens|)
            (|pop| |ruleSetTokens|)
            (|pop| |ruleSetTokens|)
            (* * |Get| |the| |Class.|)

            (SETQ |ruleClassName| (COND
                                     ((NEQ (CAR |ruleSetTokens|)
                                           |semicolon|)
                                      (|pop| |ruleSetTokens|))))

            (* * |pop| |the| |semicolon.|)

            (|pop| |ruleSetTokens|)
        |done|
            (SETQ |rsRuleClass| (|GetClassRec| |ruleClassName|))
            (COND
               ((AND |ruleClassName| (NULL |rsRuleClass|))
                (|FlushRule| "Rule class not found: " |ruleClassName|)))
            (COND
               (|rsRuleClass| (SETQ |ruleVars| (_ |rsRuleClass| |List!| 'IVS))))
            (RETURN))))
```

(|GetRuleMetaDecls|

```
  (LAMBDA NIL                                                     (* |dgb:| "17-Feb-84 16:06")

            (* * |Get| |the| |meta| |information| (|if| |any|) |associated| |with| \a |rule.|
            |Subroutine| |of| |CompileRule.|)

    (PROG (|metaTokens|)

            (* * |Set| |the| |defaults| |for| |the| |rule.|)

            (SETQ |ruleTraceFlg| |rsTraceFlg|)
            (SETQ |ruleBreakFlg| |rsBreakFlg|)
            (SETQ |ruleAuditFlg| |rsAuditFlg|)
            (SETQ |ruleMakeAuditRecordFlg| NIL)
            (SETQ |ruleAuditSpecification| NIL)
            (SETQ |ruleNeedsAuditFlg| NIL)
            (SETQ |oneShotFlg| NIL)
            (SETQ |oneShotBangFlg| NIL)
            (COND
               ((EQ |lbrace| (CAR |ruleSetTokens|))               (* |Here| |if| |{Meta-Information}| |provided| |for| |the| |rule.|)
                (SETQ |metaTokens| (|while| (AND (NEQ (CAR |ruleSetTokens|)
                                                      |semicolon|)
                                                 (NEQ (CAR |ruleSetTokens|)
                                                      |rbrace|))
                                      |collect| (|pop| |ruleSetTokens|)))
                (COND
                   ((EQ |rbrace| (CAR |ruleSetTokens|))           (* |pop| |the| |right| |brace.|)
                    (|pop| |ruleSetTokens|)))                     (* |Discard| |the| |left| |brace.|)
                (SETQ |metaTokens| (CDR |metaTokens|))
                (COND
                   ((FMEMB 1 |metaTokens|)                         (* |One| |Shots|)
                    (|GetOneShotFlg|)))
                (COND
                   ((FMEMB 'F |metaTokens|)                        (* |Trace| |Rule| |if| |satisfied|)
                    (SETQ |firstLastFlg| 'F)
                    (SETQ |metaTokens| (DREMOVE 'F |metaTokens|))))
                (COND
                   ((FMEMB 'L |metaTokens|)                        (* |Trace| |Rule| |if| |satisfied|)
                    (SETQ |firstLastFlg| 'L)
                    (SETQ |metaTokens| (DREMOVE 'L |metaTokens|))))
                (COND
                   ((FMEMB T |metaTokens|)                         (* |Trace| |Rule| |if| |satisfied|)
                    (SETQ |ruleTraceFlg| T)
                    (SETQ |metaTokens| (DREMOVE T |metaTokens|))))
                (COND
                   ((FMEMB 'TT |metaTokens|)                       (* |Trace| |Rule| |if| |tested|)
                    (SETQ |ruleTraceFlg| T)
                    (SETQ |metaTokens| (DREMOVE 'TT |metaTokens|))))
                (COND
                   ((FMEMB 'BT |metaTokens|)                       (* BT |Break| |Rule| |if| |tested.|)
                    (SETQ |ruleBreakFlg| 'BT)
                    (SETQ |metaTokens| (DREMOVE 'BT |metaTokens|)))
                   ((FMEMB 'B |metaTokens|)                        (* |Break| |Rule| |if| |satisfied.|)
                    (SETQ |ruleBreakFlg| 'B)
                    (SETQ |metaTokens| (DREMOVE 'B |metaTokens|))))
                (COND
                   ((FMEMB 'A |metaTokens|)                        (* |Audit| |Rule|)
                    (SETQ |ruleAuditFlg| T)
                    (SETQ |someRuleAuditFlg| T)
                    (SETQ |metaTokens| (DREMOVE 'A |metaTokens|))))
                (COND
                   (|metaTokens|                                   (* |Interpret| |the| |audit| |specs.|)
                          (SETQ |someRuleAuditFlg| T)
                          (SETQ |ruleAuditSpecification| (|AssocAuditSpecification| |metaTokens|)))))))))
```

```
(|GetRuleSetArgs|
  (LAMBDA NIL                                                    (* |mjs:| " 1-JUN-83 10:09")

              (* * |Parses| |the| |Args| |declaration| |at| |the| |beginning| |of| \a |RuleSet.|)

    (PROG (|args|)

              (* * |Pop| |the| |Args| |and| |colon| |tokens.|)

          (|pop| |ruleSetTokens|)
          (|pop| |ruleSetTokens|)

              (* * |Collect| |the| |Args|)

          (SETQ |args| (|while| (NEQ (CAR |ruleSetTokens|)
                                     |semicolon|)
                          |collect| (|pop| |ruleSetTokens|)))

              (* * |pop| |the| |semicolon.|)

          (|pop| |ruleSetTokens|)
      |done|
          (SETQ |rsArgs| |args|))))

(|GetRuleSetDeclarations|
  (LAMBDA (|sourceStr| |userCompilerOptions|)                    ; Edited 10-May-88 18:35 by JAMES.PA

              (* * |Parse| |the| |declarations| |at| |the| |beginning| |of| |the| |RuleSet.|
              |Subroutine| |of| |CompileRuleList.| |Argument| |self| |is| |the| |RuleSet.|)

    (PROG (|token| |newDecl| |oldDecl| |endDeclPos|)

              (* * |Find| |the| |declaration| |delimiters| |in| |the| |source| |string,| |parse| |the| |tokens,| |and| |process| |the| |declarations.|)

          (SETQ |endDeclPos| (STRPOS "********" |sourceStr|))
          (SETQ |declStr| (SUBSTRING |sourceStr| 1 |endDeclPos|))
          (|ParseTokens| |declStr|)

              (* * |Initialize| |for| |default| |declarations.|)

          (SETQ |controlType| 'DOALL)
          (SETQ |rsTraceFlg| NIL)
          (SETQ |rsBreakFlg| NIL)
          (SETQ |rsAuditFlg| NIL)
          (SETQ |rsTaskFlg| NIL)
          (COND
             ((NULL |userCompilerOptions|)
              (SETQ |rsCompilerOptions| NIL))
             (T (SETQ |rsCompilerOptions| (|InterpretCompilerOptions| |userCompilerOptions|))))
          (SETQ |wsClass| NIL)
          (SETQ |rsRuleClass| ($ |Rule|))
          (SETQ |rsAuditClass| ($ |StandardAuditRecord|))
          (SETQ |auditSpecification| (|GetClassValue| |rsAuditClass| '|metaAssns|))
          (SETQ |rsAuditSpecification| NIL)
          (SETQ |rsWhileCondition| NIL)
          (SETQ |ruleVars| NIL)
          (SETQ |taskVars| NIL)
          (SETQ |tempVars| NIL)
          (SETQ |debugVars| NIL)
          (SETQ |wsVars| NIL)
          (SETQ |rsArgs| NIL)
          (SETQ |rsSomeDeclChanged| NIL)

              (* * |Loop| |through| |the| |given| |RuleSet| |declarations.|)

      |rsDeclLoop|
          (|FlushComment?|)
          (SETQ |token| (CAR |ruleSetTokens|))
          (SELECTQ |token|
               (|WorkSpace| (|GetWorkSpaceClass|))
               (|Audit| (|GetAuditClass|))
               (|Rule| (|GetRuleClass|))
               (|Meta| (|GetMetaAssns|))
               (|Args| (|GetRuleSetArgs|))
               (|Control| (|GetControlType|))
               ((|While| |Iteration|)
                    (|GetWhileCondition|))
               (|Compiler| (|GetCompilerOptions|))
               (|Temporary| (|GetTempVars|))
               (|Task| (|GetTaskVars|))
               (|Debug| (|GetDebugVars|))
               (GO |NoMoreDecls|))
          (GO |rsDeclLoop|)
      |NoMoreDecls|
          (SETQ |rsSomeRuleAuditFlg| |rsAuditFlg|)
          (SETQ |auditSpecification| (|AssocAuditSpecification| (APPEND |auditSpecification| |rsAuditSpecification|))
                                                   ))
          (RETURN NIL))))
```

(**|GetRuleSetTemplate|**

```
  (LAMBDA NIL                                                         (* |dgb:| "17-Feb-84 17:50")

          (* * |Subroutine| |of| |CompileRuleList.| |Returns| \a |code| |template| |for| |the| |RuleSet| |that| |is| |specialized| |to| |the|
          |control| |Type.|)

    (COND
       ((AND |rsRuleAppliedFlg| (FMEMB |controlType| (CONSTANT (LIST 'WHILE1 'WHILEALL))))
                                                                      (* |Add| |extra| |bookkeeping| |if| |the| |While| |Condition|
                                                                      |reference| |^ruleApplied.|)
        (SELECTQ |controlType|
            (WHILE1 '(PROG |^progVars| (SETQ |ruleApplied| T)
                           |^firstRules|
                           |cycleLoop|
                               (COND
                                  ((NOT |^whileCondition|)            (* |Quit| |if| |while| |condition| |is| |not| |satisfied.|)
                                   (GO QUIT)))
                               (SETQ |ruleApplied| T)
                               (COND
                                  |^rules|)
                               (GO |cycleLoop|)
                           QUIT
                           |^lastRules|
                               (RETURN |^value|)))
            (WHILEALL '(PROG |^progVars| (SETQ |ruleApplied| T)
                             |^firstRules|
                             |cycleLoop|
                                 (COND
                                    ((NOT |^whileCondition|)
                                     (GO QUIT)))
                                 (SETQ |^value| |^noRuleApplied|)
                             |^rules|
                                 (COND
                                    ((EQ |^value| |^noRuleApplied|)  (* |Here| |if| |no| |rule| |applied.|)
                                     (SETQ |ruleApplied| NIL)
                                     (SETQ |^value| |^prevValue|))
                                    (T                                (* |Here| |if| |some| |rule| |applied.|)
                                       (SETQ |ruleApplied| T)
                                       (SETQ |^prevValue| |^value|)))
                                 (GO |cycleLoop|)
                             QUIT
                             |^lastRules|
                                 (RETURN |^value|)))
            NIL))
       (T (SELECTQ |controlType|
              (DO1 '(PROG |^progVars| |^firstRules|
                          (COND
                             |^rules|)
                      QUIT
                      |^lastRules|
                          (RETURN |^value|)))
              (DOALL '(PROG |^progVars| |^firstRules|
                        |^rules|
                        QUIT
                        |^lastRules|
                            (RETURN |^value|)))
              (DONEXT '(PROG |^progVars| |^firstRules|
                         |cycleLoop|
                             (SETQ |^prevValue| |^value|)
                             (SETQ |^value| |NotSetValue|)
                             (SELECTQ (@ |^task| |ruleNumber|)
                                  |^rules|
                                  NIL)
                         @   (|^task| |ruleNumber|)
                         _   (ADD1 (@ |^task| |ruleNumber|))
                             (COND
                                ((AND (EQ |^value| |NotSetValue|)
                                      (ILEQ (@ |^task| |ruleNumber|)
                                            (@ |^rs| |numRules|)))
                                                                      (* |Try| |again| |if| |no| |rule| |was| |satisfied| |and| |there| |are|
                                                                      |more| |rules| |to| |try.|)
                                 (GO |cycleLoop|)))
                         QUIT
                         |^lastRules|
                             (RETURN |^value|)))
              (WHILE1 '(PROG |^progVars| |^firstRules|
                         |cycleLoop|
                             (COND
                                ((NOT |^whileCondition|)              (* |Quit| |if| |while| |condition| |is| |not| |satisfied.|)
                                 (GO QUIT)))
                             (COND
                                |^rules|)
                             (GO |cycleLoop|)
                         QUIT
                         |^lastRules|
                             (RETURN |^value|)))
              ((FOR1 FORALL)
                   '(PROG |^progVars| |^firstRules|
                       |^forLoop|
                      QUIT
                      |^lastRules|
                          (RETURN |^value|)))
              (WHILEALL '(PROG |^progVars| |^firstRules|
                               |cycleLoop|
                                   (COND
                                      ((NOT |^whileCondition|)          (* |Quit| |if| |while| |condition| |is| |not| |satisfied.|)
                                       (GO QUIT)))
                               |^rules|
                                   (GO |cycleLoop|)
                               QUIT
                               |^lastRules|
                                   (RETURN |^value|)))
              (WHILENEXT '(PROG |^progVars| |^firstRules|
                                    (SETQ |^firstRuleTried| (@ |^task| |ruleNumber|))
                                    (COND
                                       ((NOT |^whileCondition|)
                                        (GO QUIT)))
                                |cycleLoop|
                                    (SELECTQ (@ |^task| |ruleNumber|)
                                         |^rules|
                                         NIL)
                                @   (|^task| |ruleNumber|)
                                _   (ADD1 (@ |^task| |ruleNumber|))
                                    (COND
                                       ((EQ |^value| |^noRuleApplied|)
                                                                        (* |Here| |if| |this| |rule| |not| |satisfied.|)
                                        (COND
                                           ((EQ (@ |^task| |ruleNumber|)
                                                |^firstRuleTried|)  (* |Quit| |if| |all| |the| |rules| |were| |tried| |but| |none| |were|
                                                                    |satisfied.|)
                                            (SETQ |^value| NIL)
                                            (GO QUIT))
                                           ((IGREATERP (@ |^task| |ruleNumber|)
                                                    (@ |^rs| |numRules|))
                                                                        (* |Try| |again| |starting| |at| |beginning.|)
                                            @
                                            (|^task| |ruleNumber|)
                                            |_1|
                                            (COND
                                               ((EQ |^firstRuleTried| 1)
                                                (GO QUIT)))))))
                                    (GO |cycleLoop|)
                                QUIT
                                |^lastRules|
                                    (RETURN |^value|)))
              (ERROR (CONCAT "Unrecognized Control Type=" |controlType|)))))))
```

(|**GetTaskVars**|

```
  (LAMBDA NIL                                                      (* |mjs:| "12-FEB-83 15:58")

          (* * |Parses| |the| |Task| |Vars| |declaration| |at| |the| |beginning| |of| \a |RuleSet.|
          |Argument| |self| |is| |the| |RuleSet.|)

    (PROG (|vars|)                                                 (* |Flush| |any| |leading| |comments.|)
          (|FlushComment?|)
          (COND
             ((OR (NEQ (CAR |ruleSetTokens|)
                       '|Task|)
                  (NEQ (CADR |ruleSetTokens|)
                       '|Vars|))
              (|FlushRule| "Bad Task Vars Statement.")
              (GO |done|)))

          (* * |Pop| |the| |Task,| |Vars,| |and| |colon| |tokens.|)

          (|pop| |ruleSetTokens|)
          (|pop| |ruleSetTokens|)
          (|pop| |ruleSetTokens|)

          (* * |Collect| |the| |Task| |Vars.|)

          (SETQ |vars| (|while| (NEQ (CAR |ruleSetTokens|)
                                         |semicolon|)
                          |collect| (|pop| |ruleSetTokens|)))

          (* * |pop| |the| |semicolon.|)

          (|pop| |ruleSetTokens|)
      |done|
          (SETQ |taskVars| |vars|))))
```

(**|GetTempVars|**

```
  (LAMBDA NIL                                                        (* |mjs:| "12-FEB-83 15:58")

          (* * |Parses| |the| |Temporary| |Vars| |declaration| |at| |the| |beginning| |of| \a |RuleSet.|
          |Argument| |self| |is| |the| |RuleSet.|)

    (PROG (|vars|)
          (COND
             ((OR (NEQ (CAR |ruleSetTokens|)
                       '|Temporary|)
                  (NEQ (CADR |ruleSetTokens|)
                       '|Vars|))
              (|FlushRule| "Bad Temporary Vars Statement.")
              (GO |done|)))

          (* * |Pop| |the| |Temporary,| |Type,| |and| |colon| |tokens.|)

          (|pop| |ruleSetTokens|)
          (|pop| |ruleSetTokens|)
          (|pop| |ruleSetTokens|)

          (* * |Collect| |the| |Temporary| |Vars.|)

          (SETQ |vars| (|while| (NEQ (CAR |ruleSetTokens|)
                                     |semicolon|)
                          |collect| (|pop| |ruleSetTokens|)))

          (* * |pop| |the| |semicolon.|)

          (|pop| |ruleSetTokens|)
      |done|
          (SETQ |tempVars| |vars|))))
```

(**|GetWhileCondition|**

```
  (LAMBDA NIL                                                        (* |dgb:| "21-Feb-84 11:42")

          (* * |Parses| |the| |While| |Condition| |declaration| |at| |the| |beginning| |of| \a |RuleSet.|
          |Argument| |self| |is| |the| |RuleSet.|)

    (PROG (|wcTokens|)                                               (* |Flush| |any| |leading| |comments.|)
          (|FlushComment?|)
          (COND
             ((OR (AND (NEQ (CAR |ruleSetTokens|)
                            '|While|)
                       (NEQ (CAR |ruleSetTokens|)
                            '|Iteration|))
                  (NEQ (CADR |ruleSetTokens|)
                       '|Condition|))
              (|FlushRule| "Strange While Condition Statement.")
              (GO |done|)))

          (* * |Pop| |the| |While,| |Condition,| |and| |colon| |tokens.|)

          (|pop| |ruleSetTokens|)
          (|pop| |ruleSetTokens|)
          (|pop| |ruleSetTokens|)

          (* * |Collect| |the| |While| |Condition.|)

          (SETQ |wcTokens| (|while| (NEQ (CAR |ruleSetTokens|)
                                         |semicolon|)
                              |collect| (|pop| |ruleSetTokens|)))

          (* * |pop| |the| |semicolon.|)

          (|pop| |ruleSetTokens|)

          (* * |Set| |rsRuleAppliedFlg| |if| |^ruleApplied| |is| |mentioned| |in| |the| |tokens.|)

          (SETQ |rsRuleAppliedFlg| (FMEMB '|ruleApplied| |wcTokens|))
                                                                     (* |Set| |the| |global| |taskVars.|)
      |done|
          (SETQ |rsWhileCondition| |wcTokens|))))
```

(**|GetWorkSpaceClass|**

```
  (LAMBDA NIL                                                        (* |mjs:| " 7-JUN-83 14:39")

          (* * |Parses| |the| |workSpace| |Class| |declaration| |at| |the| |beginning| |of| \a |RuleSet.|
          |Argument| |self| |is| |the| |RuleSet.|)

    (PROG (|wsClassName|)
            (* * |Flush| |any| |leading| |comments| |and| |verify| |statement| |type.|)

            (COND
               ((OR (NEQ (CAR |ruleSetTokens|)
                         '|WorkSpace|)
                    (NEQ (CADR |ruleSetTokens|)
                         '|Class|))
                (|FlushRule| "Bad WorkSpace Statement.")
                (GO |done|)))

            (* * |Pop| |the| |WorkSpace,| |Class,| |and| |colon| |tokens.|)

            (|pop| |ruleSetTokens|)
            (|pop| |ruleSetTokens|)
            (|pop| |ruleSetTokens|)

            (* * |Get| |the| |Class.|)

            (SETQ |wsClassName| (COND
                                   ((NEQ (CAR |ruleSetTokens|)
                                         |semicolon|)
                                    (|pop| |ruleSetTokens|))))

            (* * |pop| |the| |semicolon.|)

            (|pop| |ruleSetTokens|)
        |done|
            (SETQ |wsClass| (|GetClassRec| |wsClassName|))
            (COND
               (|wsClass| (SETQ |wsVars| (_ |wsClass| |List!| 'IVS))))
            (RETURN))))
```

(**|InterpretCompilerOptions|**

```
  (LAMBDA (|compilerOptions|)                                      (* |mjs:| " 8-FEB-83 18:27")

            (* * |Interprets| |compiler| |compilerOptions| |for| |the| |RuleSet| |compiler.|
            |Used| |in| |processing| |declarations,| |and| |when| |user| |specifies| |re-compilation| |without| |editing.|)

            (* * |Set| |global| |vars| |for| |the| |different| |options.|)

    (SETQ |rsAuditFlg| (COND
                          ((FMEMB 'A |compilerOptions|)
                           T)))
    (SETQ |rsTraceFlg| (COND
                          ((FMEMB 'TT |compilerOptions|)           (* |Trace| |if| |tested.|)
                           'TT)
                          ((FMEMB T |compilerOptions|)             (* |Trace| |if| |satisfied.|)
                           T)))
    (SETQ |rsBreakFlg| (COND
                          ((FMEMB 'BT |compilerOptions|)           (* |Break| |if| |Tested.|)
                           'BT)
                          ((FMEMB 'B |compilerOptions|)            (* |Break| |if| |Satisfied.|)
                           'B)))
    (SETQ |rsTaskFlg| (FMEMB 'S |compilerOptions|))
    (SETQ |rsLispCompileFlg| (FMEMB 'LC |compilerOptions|))
    (SETQ |rsPrintRuleFlg| (FMEMB 'PR |compilerOptions|))
    (SETQ |rsCompilerOptions| |compilerOptions|)))
```

(**|SpaceOrItem|**

```
  (LAMBDA (|item|)                                                 ; Edited 11-Jul-88 18:56 by jrb:

            (* |Returns| \a |space| |character| |if| |item| |is| NIL\, \a |string| |of| |separate| |items| |if| |item| |is| \a |list,| |the| |object| |name|
            |if| |name| |is| |an| |object,| |and| |item| |otherwise.|)

    (OR (COND
           ((LISTP |item|)
             ;; (PROG (str) (SETQ str "") (for item it on item do (SETQ it (CAR item)) (SETQ str (CONCAT str space it))) (RETURN str))
            (|for| \x |on| |item| |bind| \y (|str| _ "") |finally| (RETURN |str|)
               |do| (SETQ \y (CAR \x))
                    (COND
                       ((OR (EQ \y '\()
                            (EQ (CADR \x)
                                '\)))
                        (LIST \y)
                        (SETQ |str| (CONCAT |str| \y)))
                       (T (SETQ |str| (CONCAT |str| \y |space|))))))
           ((|Object?| |item|)
            (|ClassName| |item|)))
        |item| |space|)))

)

(PUTPROPS RULESD COPYRIGHT ("Venue" 1993))
```

FUNCTION INDEX

VARIABLE INDEX