Component Pascal Syntax

The lexical rules of Component Pascal are:

Ident = (Letter | “-“) {Letter | “-“ | Digit}.
Letter = “A” .. “Z” | “a” .. “Z” | “Å” .. “Œ” | “Ø” .. “œ” | “Ø” .. “ÿ”.
Digit = “0” | “1” | “2” | “3” | “4” | “5” | “6” | “7” | “8” | “9”.
Number = Integer | Real.
Integer = Digit {Digit} | Digit {HexDigit} (“H” | “L”).
Real = Digit {Digit} “.” {Digit} [ScaleFactor].
ScaleFactor = “E” [“+” | “-“] Digit {Digit}.
HexDigit = Digit | “A” | “B” | “C” | “D” | “E” | “F”.
Character = Digit {HexDigit} “X”.
String = “” {Char} “” | ‘’ {Char} ‘’.

The start symbol for a valid Component Pascal program is Module. The syntax rules of Component Pascal are:

Module = MODULE Ident “;” [ImportList] DeclSeq [BEGIN StatementSeq] [CLOSE StatementSeq] END Ident “;”.
ImportList = IMPORT [Ident “:=”] Ident {“,” [Ident “:=”] Ident} “;”.
DeclSeq = {CONST {ConstDecl “;”} | TYPE {TypeDecl “;”} | VAR {VarDecl “;”}}
          {ProcDecl “;” | ForwardDecl “;”}.
ConstDecl = IdentDef “=” ConstExpr.
TypeDecl = IdentDef “=” Type.
VarDecl = IdentList “:=” Type.
ProcDecl = PROCEDURE [Receiver] IdentDef [FormalPars]
          [“,” “NEW” [“,” (ABSTRACT | EMPTY | EXTENSIBLE)]
          [“;,” DeclSeq [BEGIN StatementSeq] END Ident].
ForwardDecl = PROCEDURE “^” [Receiver] IdentDef [FormalPars].
FormalPars = “(“ [FPSection “;” FPSection] “)“ [“;,” Type].
FPSection = [VAR | IN | OUT] Ident {“,” Ident} “:” Type.
Receiver = “(“ [VAR | IN] Ident “:” Ident “)“.
Type = Qualident
      | ARRAY [ConstExpr {“, “ ConstExpr}] OF Type
      | [ABSTRACT | EXTENSIBLE | LIMITED] RECORD [“(“ Qualident “)“]
      | FieldList “;” FieldList END
      | POINTER TO Type.
FieldList = [IdentList “;” Type].
StatementSeq = Statement {“,,” Statement}. 
Statement = [ Designator " := " Expr
| Designator [" ( " [ExprList] " )"]
| IF Expr THEN StatementSeq {ELSIF Expr THEN StatementSeq}
| [ELSE StatementSeq] END
| CASE Expr OF Case [" | " Case] [ELSE StatementSeq] END
| WHILE Expr DO StatementSeq END
| REPEAT StatementSeq UNTIL Expr
| FOR Ident " := " Expr TO Expr [BY ConstExpr] DO StatementSeq END
| LOOP StatementSeq END
| WITH Guard DO StatementSeq {" | " Guard DO StatementSeq}
| [ELSE StatementSeq] END
| EXIT
| RETURN [Expr]
].
Case = [CaseLabels [" , " CaseLabels] " : " StatementSeq].
CaseLabels = ConstExpr [" .. " ConstExpr].
Guard = Qualident [" : " Qualident].
ConstExpr = Expr.
Expr = SimpleExpr [Relation SimpleExpr].
SimpleExpr = [" + " | " - " ] Term {AddOp Term}.
Term = Factor {MulOp Factor}.
Factor = Designator
| Number
| Character
| String
| NIL
| Set
| " ( " Expr " ) "
| " ~ " Factor.
Set = [" ( " [Element [" , " Element] ] " ) "].
Element = Expr [" .. " Expr].
Relation = " = " | " # " | " < " | " <= " | " > " | " >= " | IN | IS.
AddOp = " + " | " - " | OR.
MulOp = " * " | " / " | DIV | MOD | " & ".
Designator = Qualident [" . " Ident | [" " ExprList " ] | " ^ " | " $ "
| [" ( " Qualident " ) " | [" ( " [ExprList] " ) "].
ExprList = Expr [" , " Expr].
IdentList = IdentDef [" , " IdentDef].
Qualident = [Ident [" . " ] Ident.
IdentDef = Ident [" * " | [" .. "].