```
grammar SparkS_v2;

options
{
    language = 'CSharp';
}

//#########################################################################################################################
//                                                                                                                       #
//                                                PARSER RULES                                                           #
//                                                                                                                       #
//#########################################################################################################################

//=======================================================================================================================
// Grammar starting rule                                                                                                |
//=======================================================================================================================
script
    : NEWLINE*
     TESTPROCESS ID
     NEWLINE+

      header
      body

     NEWLINE*
     END TESTPROCESS
      NEWLINE* EOF
    ;

//=======================================================================================================================
// the header of a script consists of several declarations                                                              |
//=======================================================================================================================
header
    :
    (include
    | require
    | bind
    | order
    | filter
    | testPoint
    | testGroup
    | defineStmt
    )*
    ;
```

```
//=============================================================================================================================
// the body of a script consists of several statements.                                                                       |
// it includes the actions in a test process.                                                                                 |
//=============================================================================================================================
body
    :
      block
      shutDown?
    ;

//=============================================================================================================================
// this rule defines the 'shutdown' statement                                                                                 |
// all the statements/commands that must be executed before finishing                                                         |
// the test process go after 'shutdown:' label. it is a predefined label.                                                     |
//=============================================================================================================================
shutDown
    : SHUTDOWN COLON NEWLINE+
        block
    ;

//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
//                                                                                                                           +
//                                                  HEADER RULES                                                             +
//                                                                                                                           +
//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++


//=============================================================================================================================
// rules that defines libraries to be included into the script.                                                               |
// example : Math, Dialog, ...                                                                                                |
//=============================================================================================================================
include
    : INCLUDE LIBRARY ID NEWLINE+
    ;

//=============================================================================================================================
// rule that defines the equipment to be used in the test.                                                                    |
// it may be ETE or UUT devices                                                                                               |
// example : PSA, PSG                                                                                                         |
//=============================================================================================================================
require
    : REQUIRE ID AS LINKERTYPE STRINGLITERAL TESTTYPE STRINGLITERAL (INTERFACENAME STRINGLITERAL)? NEWLINE+
    ;
```

```
//=============================================================================================================================
// this rule defines the bind statement. it is used to define some properties of the test script.                            |

// example: Test Name                                                                                                        |
//=============================================================================================================================
bind
    : BIND ID TO STRINGLITERAL NEWLINE+
    ;

//=============================================================================================================================
// this rule defines the order statement.                                                                                    |
// this is used in the test point loop to change the order of loop                                                           |
//=============================================================================================================================
order
    : ORDER ID NEWLINE+
        (orderItem NEWLINE+)+
      END ORDER NEWLINE+
    ;

orderItem
   : (ASCENDING | DESCENDING) ID
   ;


//=============================================================================================================================
// this rule defines the filter statement.                                                                                   |
// this is used in test point loops to filter test points.                                                                   |
// it filters test points based on the specified param values.                                                               |
// it may be an exact match, below than, higher than, or between two values.                                                 |
//=============================================================================================================================
filter
    : FILTER ID NEWLINE+
        (filterItem NEWLINE+)+
      END FILTER NEWLINE+
    ;

filterItem
   : paramID op=(LTEQ | GTEQ | LT | GT| EQ | NEQ) STRINGLITERAL   #filterItemSingleLimit
    | paramID O_BRAC STRINGLITERAL COMMA STRINGLITERAL C_BRAC     #filterItemDoubleLimit
   ;

//=============================================================================================================================
// this rule defines the test point type statement.                                                                          |
// the STRINGLITERAL in measure part is for UofM                                                                             |
//                                                                                                                           |
```

```
// possible alternatives for provide list:                                                                    |
// param optional                                                                                             |
// param optional [default]                                                                                   |
// param optional [,] [default]                                                                               |
//                                                                                                            |
// param                                                                                                      |
// param [,]                                                                                                  |
//                                                                                                            |
// if optional keyword comes, default value must be specified                                                 |
//============================================================================================================
testPoint
    : TESTPOINT  tpId NEWLINE+
        PROVIDE NEWLINE*
           testPointParam+ NEWLINE*
        MEASURE NEWLINE* ID IN STRINGLITERAL  NEWLINE+
       END  TESTPOINT NEWLINE+
    ;

testPointParam
    : paramID (enumValues | (OPTIONAL ((enumValues)? defaultValue)))? NEWLINE+
    ;

tpId
    : ID
    ;

paramID
   : ID
   ;

enumValues
   : O_BRAC STRINGLITERAL (COMMA STRINGLITERAL)* C_BRAC
   ;

defaultValue
   : O_BRAC DEFAULT STRINGLITERAL C_BRAC
   ;

//============================================================================================================
// this rule defines the test group statement and how it is bound to test point type.                        |
//============================================================================================================
testGroup
    :TESTGROUP ID TESTPOINT tpId NEWLINE+
    ;
```

```
//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
//                                                                                                                              +
//                                                  BODY RULES                                                                  +
//                                                                                                                              +
//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++


//==============================================================================================================================
// the possible statements that can be used in the body of the script                                                           |
//==============================================================================================================================
stmt
    : forBlock
    | expandStmt
    | ifThenElseStmt
    | tryBlock
    | set_stmt
    | constDeclaration
    | functionCall
    | sleepCommand
    ;

block
    : (stmt (NEWLINE+  stmt)* NEWLINE*)?
    ;

//==============================================================================================================================
// this rule defines the function call statement.                                                                               |
//==============================================================================================================================
functionCall
    : ID DOT ID (LPAREN (functionArgument (COMMA functionArgument)*)? RPAREN)?
    ;

functionArgument
    : STRINGLITERAL
    //add other types -> arith
    ;

//==============================================================================================================================
// this rule defines how a constant can be defined                                                                              |
//==============================================================================================================================
constDeclaration
    : CONST ID EQ DOUBLELITERAL NEWLINE+
    ;
```

```
//=========================================================================================================================
// this rule defines the try/catch statement                                                                             |
//=========================================================================================================================
tryBlock
    : TRY NEWLINE*
        block
      (tryBlockOnStat)+
      END TRY
    ;

tryBlockOnStat
    : ON ID DO NEWLINE* block END DO NEWLINE+
    ;

//=========================================================================================================================
// this rule specifies the 'define' statement.                                                                           |
// it has two types:                                                                                                     |
// - one is used to define how a test point type is derived from another test point type.                              |
// - the other is used to define macros to be used as functions in different parts of the script.                      |
//=========================================================================================================================
defineStmt
   : DEFINE ID AS NEWLINE+
      defineSubStmt
     END DEFINE NEWLINE+
   ;

defineSubStmt
    : fillStmt
    | macro
    ;

fillStmt
   : FILL ID USING ID NEWLINE+
      //(fillSetStmt)+
      block
     END FILL NEWLINE+
   ;

macro
   : block
   ;
```

```
//=============================================================================================================================
// this rule defines the 'expand' statement.                                                                                  |
// it is used in two cases:                                                                                                   |
// 1) when we want to derive a new test point from an already available test point. For that reason it has to                 |
// go inside a test point loop construct. Using it in other places would make it not to work.                                 |
// 2) for macro call                                                                                                          |
//=============================================================================================================================
expandStmt
   : EXPAND ID (REPLACE expandReplaceStmt (NEWLINE* COMMA NEWLINE* expandReplaceStmt)*)?
   ;

expandReplaceStmt
   : ID WITH ID
   ;

//=============================================================================================================================
// this rule defines the if-then-else statement                                                                               |
//=============================================================================================================================
ifThenElseStmt :
    IF conditionBlock (ELSEIF conditionBlock)* (ELSE NEWLINE+ block)? END_IF
    ;

conditionBlock :
    boolExprMain THEN NEWLINE+ block
    ;

//=============================================================================================================================
// the rules that handle the bool expression                                                                                  |
//=============================================================================================================================
boolExprMain
    : boolExpr                                                          #boolExprRule
   | left=boolOperand op=(EQ | NEQ) right=boolOperand                   #boolExprEq
   ;
boolOperand
    : boolExpr                                                          #boolOperandexpr
   | ID                                                                 #boolOperandID
   ;
boolExpr
    : NOT boolExprSub                                                   #boolNotExpr
    | left=boolExprSub op=(AND|OR|XOR) right=boolExprSub                #boolBinaryExpr
    | boolExprValue                                                     #boolExprVal
    ;
boolExprSub
    : LPAREN boolExpr RPAREN                                            #boolExprParen
    | boolExprValue                                                     #boolExprSubVal
    ;
```

```
boolExprValue
    : left=arithExpr op=(LTEQ | GTEQ | LT | GT| EQ | NEQ) right=arithExpr           #BoolArithmeticExpr
    | (TRUE |  FALSE)                                                               #boolAtom
    ;


//=============================================================================================================================
// rules that handle arithmetic expressions                                                                                   |
//=============================================================================================================================
arithExpr
    : left=arithExprSub op=('*' | '/') right=arithExprSub          #binaryExpr
    | left=arithExprSub op=('+' | '-') right=arithExprSub          #binaryExpr
    | arithExprValue                                               #arithExprVal
    ;

arithExprSub
   : '(' arithExpr ')'                                             #arithExprSubParen
   | arithExprValue                                                #arithExprSubVal
   ;

arithExprValue
    : DOUBLELITERAL                                                #arithExprNumber
    //STRINGLINTERAL needs to be added! -> if ModulationSource = "External" then
    | ID                                                           #arithExprId            //this is for variables
    | functionCall                                                 #arithExprFunctionCall
    //| ID DOT ID                                        #arithExprIdAttr        //this is for property of an object
    ;

//=============================================================================================================================
// for-loop variations                                                                                                        |
// includes two variations: one for test point loops, another that loops through a range of values                            |
// [lowerLimit, upperLimit]                                                                                                   |
//=============================================================================================================================
forBlock
    : FOR  EACH loopRange DO NEWLINE+
        loopBody
      END  FOR
    ;
loopRange
    : ID  IN  ID (FILTER filterID)? (ORDER orderID)?                    #testPointRange
    | ID  IN  O_BRAC intRangeDataType COMMA intRangeDataType C_BRAC     #intRange
    ;
intRangeDataType
    : DOUBLELITERAL                                                     #intRangeInteger
    | ID                                                                #intRangeID
    ;
```

```
loopBody
    : block
    ;

filterID
   : ID
    ;

orderID
   : ID
   ;

//=============================================================================================================
// rules that define set statements                                                                           |
//=============================================================================================================
set_stmt
   : SET  ID TO  setExpr                  #setVar
   | SET  ID DOT ID  TO  setExpr          #setAttr
   //| SET UNC TO setExpr                 #setUNC     //handle this in the interpreter using ID
   ;

setExpr
   : arithExpr                            #setExprArith
   | STRINGLITERAL                        #setExprString
   //| ID DOT ID                          #setExprAttr
   | functionCall                         #setExprFunctionCall
   ;

//=============================================================================================================
// sleep rule                                                                                                 |
// this is for when you want to pause the test process for a while                                            |
//=============================================================================================================
sleepCommand
    : SLEEP FOR DOUBLELITERAL ID NEWLINE+
    ;




//=============================================================================================================
//                                                                                                            |
//                                          LEXER TOKENS                                                      |
//                                                                                                            |
//=============================================================================================================
```

```
// Whitespace
NEWLINE     : '\r'? '\n';
WS          : [ \t] -> skip;

INCLUDE     :'include';
LIBRARY     :'library';
REQUIRE     :'require';
LINKERTYPE  :'linkerType';
TESTTYPE    :'testType';
INTERFACENAME :'interfaceName';
BIND        :'bind';
ORDER       :'order';
FILTER      :'filter';
TRY         :'try';
ON          :'on';
ASCENDING   :'ascending';
DESCENDING  :'descending';
TESTPROCESS :'testProcess';
TESTPOINT   :'testPoint';
FILL        :'fill';
USING       :'using';
EXPAND      :'expand';
REPLACE     :'replace';
WITH        :'with';
FROM        :'from';
OPTIONAL    :'optional';
TESTGROUP   :'testGroup';
DEFINE      :'define';
AS          :'as';
PROMPT      :'prompt';
DO          :'do';
END         :'end';
IF          :'if';
ELSE        :'else';
FOR         :'for';
EACH        :'each';
IN          :'in';
SET         :'set';
TO          :'to';
THEN        :'then';
COMMA       :',';
DOT         :'.';
COLON       :':';
IS          :'IS';
NULL        :'NULL';
```

```
END_IF      :'end if';
ELSEIF      :'else if';
PROVIDE     :'provide';
MEASURE     :'measure';
DEFAULT     :'default';
CONST       :'constant';
SLEEP       :'sleep';
SHUTDOWN    :'SHUTDOWN';

TRUE: 'TRUE';
FALSE: 'FALSE';

//-----------------------
// Control Chars        |
//-----------------------
LPAREN : '(';
RPAREN : ')';
O_BRAC : '[';
C_BRAC : ']';

//------------------------------
// Binary Arithmetic Operators |
//------------------------------
MULT : '*';
DIV   : '/';
PLUS     : '+';
MINUS    : '-';

//-----------------------
// Comparison Operators |
//-----------------------
EQ      : '=';
NEQ     : '/=';
GTEQ    : '>=';
LTEQ    : '<=';
GT      : '>';
LT      : '<';

//-----------------------
// Logical Operators    |
//-----------------------
NOT : 'not';
AND : 'and';
OR  : 'or';
XOR : 'xor';
```

```
//------------------------------
// Literals and Identifiers    |
//------------------------------

fragment DIGIT : [0-9];
fragment LETTER : [A-Za-z];
fragment CHARACTER : DIGIT | LETTER | '_';

//INT: (PLUS|MINUS)? DIGIT+ ( ('e' | 'E') INT)?;
DOUBLELITERAL : (PLUS|MINUS)? DIGIT+ ('.' DIGIT+)? ( ('e' | 'E') (PLUS|MINUS)? ('0'..'9')+)?;
STRINGLITERAL : '"' (~["\r\n])* '"';

ID: LETTER CHARACTER*;

//-----------------------
//Comments              |
//-----------------------
COMMENT
    :   '/*' .*? '*/'  -> skip // match anything between /* and */
    ;
LINE_COMMENT
    : //'//' ~[\r\n]* '\r'? '\n' -> skip
      '//' ~[\r\n]* -> skip
    ;
```