# SPSPSCL PROJECT

# Grammar of the *SPSPSCL* System Programming Language

**Technical Report CS-Sysplm-2-2020. Updated Dec 2021**

**Dr. José M. Garrido**
**Department of Computer Science**

**Jan, 2022**

**College of Computing and Software Engineering**
**Kennesaw State University**

# 1 The SPSCL Language

This report describes the formal grammar in BNF of the high-level system programming language *SPSCL*. This is part of larger ongoing project that has as its goal the investigation of newer approaches, methods, techniques, and tools for improving the software development of embedded systems.

The language syntax is defined at a higher level of abstraction than C, C++, and Java. Its compiler is implemented as a pre-processor that generates C code, that will subsequently be compiled and linked with standard tools. This report provides the BNF specification of the Sysplm language. This language was designed and its translator developed for implementing low-level programs, such as programs for micro-controllers.

It is hoped that with the SPSCL language and additional software support, tools, and approaches computer engineers would become more acquainted with sound software development processes. On the other hand, computer scientist would potentially gain more knowledge and interest in the hardware aspects of embedded systems.

The language syntax is defined at a higher level of abstraction than C, and includes language statements for improving program readability, debugging, maintenance, and correctness. The language design was influenced by Ada, Pascal, C, previous generation of system programming languages such as PL/M, SPL, etc.

# 2 The SPSCL Grammar

The following grammar specification is in a modified BNF notation. The terminal symbols are all in upper case.

```
prog    ::= imports symbols forward_refs specifications globals implement

imports    ::=
         | imports import_file

import_file    ::= IMPORT TSTRING
             | USE TSTRING

symbols    ::=
         | symbols symbol_def

symbol_def    ::= SYMBOL IDENTIFIER HCON

forward_refs    ::=
              | FORWARD forward_list

forward_list    ::= forwards
              | forward_list forwards
```

```
forwards    ::= INTERFACE IDENTIFIER
          | STRUCT IDENTIFIER

          | STRUCTYPE IDENTIFIER

          | check_ext func_main parameters

          | DEFINETYPE struct_enum IDENTIFIER IDENTIFIER

check_ext    ::=
           | MEXTERN

func_main    ::= FUNCTION IDENTIFIER oper_type
           | MAIN

oper_type    ::= RETURN chk_ptr chk_array ret_type

chk_ptr    ::=
         | POINTER OF

chk_array    ::=
           | ARRAY array_dim_list

array_dim_list    ::= LB array_index RB
                | array_dim_list LB array_index RB

array_index    ::= IDENTIFIER
             | ICON

ret_type     ::= TYPE type_name
           | STRUCT IDENTIFIER
           | STRUCTYPE IDENTIFIER

type_name          ::= MVOID
                 | COUNT
                 | INTEGER
                 | SHORT
                 | REAL
                 | FLOAT
                 | DOUBLE
                 | TBOOL
                 | CHAR
                 | TSTRING OF LENGTH ICON
                 | TBYTE

struct_enum    ::= STRUCT
             | ENUM

specifications     ::=
                 | SPECIFICATIONS spec_list
```

```
spec_list   ::= spec_def
          | spec_list spec_def

spec_def   ::= ENUM
         | STRUCT
         | DESCRIPTION

globals   ::=
        | GLOBAL  DECLARATIONS const_dec var_dec struct_dec

const_var_struct   ::= const_dec var_dec struct_dec

const_dec   ::=
          | CONSTANTS data_declarations

var_dec   ::= VARIABLES data_declarations

struct_dec   ::=
           | STRUCT data_declarations

data_declarations   ::= comp_declare
                  | data_declarations comp_declare

comp_declare   ::= DEFINE data_file

data_file   ::= sel_file
          | sel_dmode data_declaration

sel_dmode   ::=
          | PERSISTENT
          | SHARED
          | STATIC
          | MEXTERN

sel_file   ::=
         | MFILE

data_declaration   ::= IDENTIFIER opt_pointer parray_dec OF data_type

opt_pointer   ::=
            | POINTER

data_type   ::= TUNSIGNED
                        | CHAR
                        | INTEGER
                        | MVOID
                        | DOUBLE
                        | LONG
                        | SHORT
```

```
                        | FLOAT
                        | REAL
                        | TSTRING
                        | TBOOL
                        | TBYTE

parray_dec   ::=
          | ARRAY plist_const popt_array_val
          | LB
          | VALUE
          | EQUOP

plist_const   ::= LB iconst_ident RB
           | plist_const LB iconst_ident RB

iconst_ident   ::= ICON
            | IDENTIFIER

popt_array_val   ::=
              | value_eq array_val

value_eq   ::= VALUE
         | EQUOP

array_val   ::= simp_arr_val

simp_arr_val   ::= LB arg_list RB

arg_list   ::= expr
         | arg_list COMMA expr

implement   ::= IMPLEMENTATIONS main_head funct_list

main_head   ::=
          |  MAIN DESCRIPTION parameters

funct_list   ::= funct_body
           | funct_list funct_body

funct_body  ::= FUNCTION phead_fun pother_oper_def

phead_fun   ::=
          | PERSISTENT
          | STATIC

pother_oper_def   ::= pother_oper IS const_var_struct precond
                      BEGIN pactions  ENDFUN  IDENTIFIER

pother_oper   ::= acc_mut IDENTIFIER DESCRIPTION oper_type parameters
```

```
acc_mut    ::=
         | ACCESSOR
         | MUTATOR

precond    ::=
         | PRECONDITION pcondition

pcondition    ::= pcond1 OR pcond1
            | pcond1 AND pcond1
            | pcond1

pcond1    ::= NOT pcond2
        | pcond2

pcond2    ::= LP pcondition RP
        | expr RELOP expr
        | expr EQOP expr
        | expr eq_v expr
        | expr opt_not true_false
        | element

true_false    ::= MTRUE
            | MFALSE

eq_v    ::= EQUALS
      | GREATER THAN
      | LESS THAN
      | GREATER OR EQUAL
      | LESS OR EQUAL

opt_not    ::=
         | NOT

parameters    ::=
            | PARAMETERS  param_list

param_list    ::= param_def
            | param_list COMMA param_def

param_def    ::= param_mode data_declaration

param_mode    ::=
            | ALTERS
            | PRESERVES
            | PRODUCES
            | CONSUMES

expr    ::= term PLUS term
      | term MINUS term
      | term BAND term
```

```
      | term BOR term
      | term BXOR term

term   ::= punary
      | punary STAR punary
      | punary DIVOP punary
      | punary MOD punary
      | punary LSHIFT punary
      | punary RSHIFT punary

punary   ::= element
        | ADDRESS element
        | DEREF element
        | MINUS element
        | NEGATE element

element   ::= IDENTIFIER popt_ref
         | STRING
         | LETTER
         | ICON
         | HCON
         | FCON
         | MTRUE
         | MFALSE
         | LP expr RP

pactions   ::= action_def
          | pactions action_def

action_def   ::= ADD name_ref TO name_ref
            | SUBTRACT name_ref FROM name_ref
            | SET name_ref EQUOP expr
            | READ pvar_value_list
            | INPUT name_ref
            | DISPLAY pvar_value_list
            | DISPLAYN pvar_value_list
            | MCLOSE IDENTIFIER
            | MOPEN in_out
            | MFILE read_write
            | INCREMENT name_ref
            | DECREMENT name_ref
            | RETURN expr
            | CALL name_ref pusing_ref
            | IF pcondition THEN pactions ptest_elsif
                  opt_else ENDIF
            | FOR name_ref EQUOP expr downto expr
                  DO pactions ENDFOR
            | REPEAT pactions UNTIL pcondition ENDREPEAT
            | WHILE pcondition DO pactions ENDWHILE
            | CASE name_ref pcase_val pcase_def MENDCASE
```

```
            | MBREAK
            | MEXIT
            | ENDFUN name_ref
            | POTCONDITION pcondition

ptest_elsif   ::=
            | proc_elseif

proc_elseif   ::= ELSEIF pcondition THEN pactions
            | proc_elseif ELSEIF pcondition THEN pactions

downto   ::= TO
       | DOWNTO

pusing_ref   ::=
            | USING arg_list
            | parguments

parguments   ::= LP arg_list RP


pcase_val   ::= MWHEN expr COLON pactions
          | pcase_val MWHEN expr COLON pactions

pcase_def   ::=
          | DEFAULT COLON pactions

pvar_value_list   ::= expr
                | pvar_value_list COMMA expr

opt_else   ::=
         | ELSE pactions

in_out   ::= INPUT MFILE IDENTIFIER
       | OUTPUT MFILE IDENTIFIER

read_write   ::= READ pvar_value_list FROM IDENTIFIER
           | WRITE pvar_value_list TO IDENTIFIER

name_ref   ::= IDENTIFIER opt_ref pmember_opt  popt_dot

pmember_opt   ::=
            | pmember_of

pmember_of   ::= OF IDENTIFIER opt_ref
           | pmember_of OF IDENTIFIER opt_ref

opt_ref   ::= array_val

popt_ref   ::=
```

```
          | array_val
          | parguments

popt_dot    ::=
          | proc_dot

proc_dot    ::= DOT IDENTIFIER opt_ref
          | proc_dot DOT IDENTIFIER opt_ref
```