The dSL Grammar

Created from Yacc specification *

January 13, 2003

1 Grammar

program	::=	uses_declaration data_type_declaration global_program_var_declarations global_statement_list "**PROGRAM**" TRM_IDENTIFIER program_var_declarations statement_list cycle_statements "**END_PROGRAM**"
cycle_statements	::=	["**CYCLE**" statement_list]
uses_declaration	::=	["**USES**" uses_list "**END_USES**"]
uses_list	::=	[TRM_IDENTIFIER "**;**"]
data_type_declaration	::=	["**TYPE**" type_declaration_list "**END_TYPE**"]
type_declaration_list	::=	{ type_declaration "**;**" }
rule_statement	::=	when_statement
		| global_globalisation_statement
		| typed_when_statement
		| typed_globalisation_statement
launch_directive	::=	["**LAUNCH**"]
async_directive	::=	["**ASYNC**"]
when_statement	::=	async_directive "**WHEN**" expression "**THEN**" local_variables statement_list "**END_WHEN**"
typed_when_statement	::=	async_directive "**WHEN**" "**IN**" simple_specification expression "**THEN**" local_variables statement_list "**END_WHEN**"
global_globalisation_statement	::=	"**CONTROL**" variable "**WITH**" expression
typed_globalisation_statement	::=	"**CONTROL**" "**IN**" simple_specification variable "**WITH**" expression
statement_list	::=	{ statement "**;**" } statement "**;**"

*Work supported by Bruxelles Capitales Region, grant no. RBC-BR 227/3298.

statement ::= [“**NIL**” | assignment_statement | method_call_statement | call_all_statement | iteration_statement | selection_statement | print_statement | error_statement | say_statement | log_statement | alias_statement | nature_statement | domain_statement | set_push_variable_statement | wait_statement]

print_statement ::= “**PRINT**” value_list

error_statement ::= “**ERROR**” value_list

say_statement ::= “**SAY**” value_list

log_statement ::= “**LOG**” value_list

iteration_statement ::= while_statement | repeat_statement | for_statement | exit_statement

selection_statement ::= if_statement | case_statement

method_call_statement ::= launch_directive variable “<−” TRM_IDENTIFIER “(” value_list “)”

assignment_statement ::= launch_directive variable “**:**=” expression

call_all_statement ::= “**ALL**” simple_specification “<−” TRM_IDENTIFIER “(” value_list “)”

alias_statement ::= “**ALIAS**” expression expression expression

nature_statement ::= “**SET_NATURE**” variable expression

domain_statement ::= “**SET_DOMAIN**” variable expression

set_push_variable_statement ::= “**SET_PUSH_VARIABLE**” variable expression expression expression

do_particle ::= “**DO**”

until_particle ::= “**UNTIL**”

exit_statement ::= “**EXIT**”

while_statement ::= “**WHILE**” expression do_particle statement_list “**END_WHILE**”

repeat_statement ::= “**REPEAT**” statement_list until_particle expression “**END_REPEAT**”

for_statement ::= “**FOR**” variable “**:**=” expression “**TO**” expression (do_particle statement_list “**END_FOR**” | “**BY**” expression do_particle statement_list “**END_FOR**”)

elsif_block ::= “**ELSIF**” expression “**THEN**” statement_list (“**ENDIF**” | “**ELSE**” statement_list “**ENDIF**” | elsif_block)

```
if_statement                ::= "IF" expression "THEN"
                                statement_list ( "ENDIF" | "ELSE"
                                statement_list "ENDIF" | elsif_block )
signed_integer_constant     ::= TRM_UNSIGNED_INTEGER
                              | TRM_INTEGER
                              | "-" TRM_UNSIGNED_INTEGER
                              | "+" TRM_UNSIGNED_INTEGER
case_list_element           ::= signed_integer_constant [ ":"
                                signed_integer_constant ]
case_list                   ::= { case_list_element "," }
                                case_list_element
case_element                ::= case_list ":" statement_list
list_of_case_elements       ::= { case_element } case_element
case_statement              ::= "CASE" expression "OF"
                                list_of_case_elements ( "END_CASE"
                                | "ELSE" statement_list
                                "END_CASE" )
wait_statement              ::= "WAIT" expression
value_list                  ::= [ expression [ "," value_list ] ]
int_rval                    ::= add_expression
expression                  ::= or_expression
equal_operator              ::= "="
                              | "<>"
comparison_operator         ::= "<"
                              | ">"
                              | "<="
                              | ">="
comparison                  ::= equ_expression { equal_operator
                                equ_expression }
equ_expression              ::= add_expression { comparison_operator
                                add_expression }
or_operator                 ::= "OR"
or_expression               ::= xor_expression { or_operator
                                xor_expression }
xor_operator                ::= "XOR"
xor_expression              ::= and_expression { xor_operator
                                and_expression }
and_operator                ::= "AND"
                              | "&"
and_expression              ::= comparison { and_operator comparison
                                }
add_expression              ::= { term ( "+" | "-" ) } term
term                        ::= power_expression
                              | term "*" unary_expression
                              | term "/" unary_expression
                              | term "MOD" unary_expression
power_expression            ::= unary_expression [ "POWER"
                                unary_expression ]
```

unary_expression	::=	primary_expression
	|	"−" primary_expression
	|	"+" unary_expression
	|	"**NOT**" unary_expression
	|	"**IS_UNKNOWN**" unary_expression
	|	" " unary_expression
	|	"**ADDRESS**" unary_expression
primary_expression	::=	TRM_INTEGER
	|	TRM_UNSIGNED_INTEGER
	|	TRM_LONG_INTEGER
	|	TRM_UNSIGNED_LONG_INTEGER
	|	TRM_FLOATING_POINT
	|	"**TRUE**"
	|	"**FALSE**"
	|	"**UNKNOWN_BOOL**"
	|	"**UNKNOWN_SINT**"
	|	"**UNKNOWN_INT**"
	|	"**UNKNOWN_DINT**"
	|	"**UNKNOWN_LINT**"
	|	"**UNKNOWN_USINT**"
	|	"**UNKNOWN_UINT**"
	|	"**UNKNOWN_UDINT**"
	|	"**UNKNOWN_ULINT**"
	|	"**UNKNOWN_REAL**"
	|	"**UNKNOWN_LREAL**"
	|	"**UNKNOWN_STRING**"
	|	"**UNKNOWN_TIME**"
	|	"**UNKNOWN_TIME_OF_DAY**"
	|	"**UNKNOWN_DATE_AND_TIME**"
	|	"**UNKNOWN_DATE**"
	|	"**NULL**"
	|	TRM_QUOTED_STRING
	|	TRM_TIME_CONSTANT
	|	TRM_DATE_CONSTANT
	|	TRM_DATE_AND_TIME_CONSTANT
	|	TRM_TIME_OF_DAY_CONSTANT
	|	variable
	|	"(" expression ")"
	|	function_call
function_call	::=	function_name "(" value_list ")"
function_name	::=	TRM_IDENTIFIER
variable	::=	variable "^"
	|	variable "**.**" TRM_IDENTIFIER
	|	variable "[" int_rval "]"
	|	TRM_IDENTIFIER
global_variable_declarations	::=	"**GLOBAL**" a_var_decl_list
		"**END_VAR**"

```
other_var_declarations          ::= var_declarations
program_var_declaration         ::= global_variable_declarations
                                  | other_var_declarations
program_var_declarations        ::= { program_var_declaration }
global_program_var_declarations ::= { global_variable_declarations }
a_var_decl_list                 ::= a_var_decl { ";" a_var_decl }
a_var_decl                      ::= [ TRM_IDENTIFIER ( ":"
                                    type_specification | "," a_var_decl ) ]
structure_specification         ::= structure_declaration
enum_specification              ::= enum_declaration
enum_declaration                ::= "(" enum_element_declaration_list ")"
enum_element_name               ::= TRM_IDENTIFIER
enum_element_declaration        ::= enum_element_name
enum_element_declaration_list   ::= enum_element_declaration { ","
                                    enum_element_declaration }
sub_range                       ::= signed_integer_constant ":"
                                    signed_integer_constant
sub_range_list                  ::= sub_range { "," sub_range }
type_specification              ::= array_specification
                                  | structure_specification
                                  | pointer_specification
                                  | list_specification
                                  | enum_specification
                                  | simple_specification
array_specification             ::= "ARRAY" "[" sub_range_list "]" "OF"
                                    type_specification
list_specification              ::= "LIST" "["
                                    TRM_UNSIGNED_INTEGER "]"
                                    "OF" type_specification
pointer_specification           ::= "POINTER" "TO" type_specification
structure_declaration           ::= "STRUCT"
                                    structure_element_declaration_list
                                    "END_STRUCT"
structure_element_name          ::= TRM_IDENTIFIER
structure_element_declaration   ::= structure_element_name ":"
                                    type_specification
structure_element_declaration_list ::= structure_element_declaration ";" [
                                    structure_element_declaration_list ]
a_type_name                     ::= TRM_IDENTIFIER
type_declaration                ::= a_type_name ":" type_specification
global_statement                ::= method_declaration
                                  | rule_statement
                                  | sequence
global_statement_list           ::= { global_statement }
local_variables                 ::= [ "VAR" a_var_decl_list "END_VAR" ]
var_declarations                ::= "VAR" a_var_decl_list "END_VAR"
```

method_declaration	::=	“**METHOD**” simple_specification “**::**” TRM_IDENTIFIER “(” a_var_decl_list “)” (local_variables statement_list “**END_METHOD**” | “**FORWARD**”)
sequence	::=	async_directive “**SEQUENCE**” TRM_IDENTIFIER local_variables statement_list “**END_SEQUENCE**”
simple_specification	::=	“**BOOL**”
	|	“**SINT**”
	|	“**INT**”
	|	“**DINT**”
	|	“**LINT**”
	|	“**USINT**”
	|	“**UINT**”
	|	“**UDINT**”
	|	“**ULINT**”
	|	“**REAL**”
	|	“**LREAL**”
	|	“**TIME**”
	|	“**TIME_OF_DAY**”
	|	“**DATE_AND_TIME**”
	|	“**DATE**”
	|	“**STRING**”
	|	TRM_IDENTIFIER

2 Remarks

[T] Means 1 at least one T.
{ T } Means T or ϵ
“**T**” is a literal.
TRM_T is a terminal.