EBNF Grammar for Objective Modula-2

Typography

- Production rules are shown in `fixed width`
- Alias rules are shown in *`italic fixed width`*
- Reserved words are shown in **`UPPERCASE BOLDFACE`**
- Other terminal symbols are shown in **`lowercase boldface`**
- Reserved symbols are shown in `'single quotes'` or `"double quotes"`

EBNF Notation

- Alternatives are separated by a vertical bar `|`
- Groups of entities are enclosed in parentheses `()`
- One or more occurrences are indicated by a trailing raised plus sign `+`
- Zero or more occurrences are indicated by a trailing raised asterisk `*`
- Zero or one occurrences are indicated by a trailing raised question mark `?`

Compilation Units

(1) compilationUnit
    programModule | definitionOfModule | implementationOfModule |
    protocolModule

(2) programModule
    MODULE *moduleId* ("[" *priority* "]")? ";"
    importList* block *moduleId* "."

(3) definitionOfModule
    DEFINITION MODULE *moduleId* ";"
    importList* definition*
    END *moduleId* "."

(4) implementationOfModule
    IMPLEMENTATION programModule

(5) protocolModule
    PROTOCOL *protocolId* ("(" *adoptedProtocols* ")")? ";"
    importList* (OPTIONAL? methodHeader)*
    END *protocolId* "."

(6) *moduleId* = **ident**

(7) *priority* = constExpression

(8) *protocolId* = **ident**

(9) *adoptedProtocols* = identList

EBNF Grammar for Objective Modula-2

Import Lists, Blocks, Declarations, Definitions

```
(10) importList
     ( FROM moduleId IMPORT ( identList | "*" ) | IMPORT identList ) ";"

(11) block
     declaration*
     ( BEGIN statementSequence )? END

(12) declaration
     CONST ( constantDeclaration ";" )* |
     TYPE ( typeDeclaration ";" )* |
     VAR ( variableDeclaration ";" )* |
     procedureDeclaration ";" |
     methodDeclaration ";"

(13) definition
     CONST ( constantDeclaration ";" )* |
     TYPE ( ident ( "=" ( type | OPAQUE ) | IS namedType ) ";" )* |
     VAR ( variableDeclaration ";" )* |
     procedureHeader ";" |
     methodHeader ";"
```

Constant Declarations

```
(14) constantDeclaration
     ident "=" ( constExpression | structuredValue )
```

Type Declarations

```
(15) typeDeclaration
     ident ( "=" type | IS namedType )

(16) type
     namedType | anonymousType | enumerationType | setType | classType

(17) namedType = qualident

(18) anonymousType
     arrayType | recordType | pointerType | procedureType

(19) enumerationType
     ENUM ( "(" baseType ")" )? identList? END |
     "(" identList ")"

(20) baseType = qualident
```

```
(21) arrayType
     ARRAY arrayIndex ( "," arrayIndex )*
     OF ( namedType | recordType | procedureType )

(22) arrayIndex = ordinalConstExpression

(23) ordinalConstExpression = constExpression

(24) recordType
     RECORD ( "(" baseType ")" )? fieldListSequence? END

(25) fieldListSequence
     fieldList ( ";" fieldList )*

(26) fieldList
     identList ":"
     ( namedType | arrayType | recordType | procedureType )

(27) classType
     "<*QUALIFIED*>"? CLASS "(" superClass ( "," adoptedProtocols )? ")"
     ( ( PUBLIC | MODULE | PROTECTED | PRIVATE )?
     fieldListSequence )*
     END

(28) superClass = qualident

(29) setType
     SET OF ( namedType | "(" identList ")" )

(30) pointerType
     POINTER TO IMMUTABLE? namedType

(31) procedureType
     PROCEDURE
     ( "(" formalTypeList ")" )?
     ( ":" returnedType )?

(32) formalTypeList
     attributedFormalType ( "," attributedFormalType )*

(33) attributedFormalType
     IMMUTABLE? VAR? formalType

(34) formalType
     ( ARRAY OF )? namedType

(35) returnedType = namedType
```


EBNF Grammar for Objective Modula-2

Variable Declarations

(36) variableDeclaration
     ident ("[" *machineAddress* "]" | "," identList)?
     ":" (*namedType* | anonymousType)

(37) *machineAddress* = constExpression

Procedure Declarations

(38) procedureDeclaration
     procedureHeader ";" block **ident**

(39) procedureHeader
     PROCEDURE
     ("(" **ident** ":" *receiverType* ")")?
     ident ("(" formalParamList ")")? (":" *returnedType*)?

(40) *receiverType* = **ident**

(41) formalParamList
     formalParams (";" (formalParams | variadicParams))*

(42) formalParams
     IMMUTABLE? **VAR**? identList ":" formalType

(43) variadicParams
     VARIADIC *handle* ("[" *indexParam* "]")? **OF**
     IMMUTABLE? **VAR**? (**ident** (("." **ident**)* | ("," **ident**)* ":" formalType)

(44) *handle* = **ident**

(45) *indexParam* = **ident**

Method Declarations

(46) methodDeclaration
     methodHeader ";" block **ident**

(47) methodHeader
     CLASS? **METHOD**
     "(" **ident** ":" (*receiverClass* | "*") ")"
     (**ident** | methodArg) methodArg*
     (":" *returnedType*)?

(48) *receiverClass* = qualident

(49) methodArg
     colonIdent "(" **IMMUTABLE**? **VAR**? **ident** ":" formalType ")"

Statements

(50) statement
     (assignmentOrProcedureCall | methodInvocation |
       ifStatement | caseStatement | whileStatement | repeatStatement |
       loopStatement | forStatement | tryStatement | criticalStatement |
       RETURN expression? | **EXIT**)?

(51) statementSequence
     statement (";" statement)*

(52) methodInvocation
     "[" receiver message "]"

(53) receiver
     ident | methodInvocation

(54) message
     ident (**colonIdent** expression)* |
     (**colonIdent** expression)+

(55) assignmentOrProcedureCall
     designator
     (":=" (expression | structuredValue) | "++" | "--" |
       actualParameters)?

(56) ifStatement
     IF expression **THEN** statementSequence
     (**ELSIF** expression **THEN** statementSequence)*
     (**ELSE** statementSequence)?
     END

(57) caseStatement
     CASE expression **OF** case ("|" case)*
     (**ELSE** statementSequence)?
     END

(58) case
     caseLabelList ":" statementSequence

(59) caseLabelList
     caseLabels ("," caseLabels)*

(60) caseLabels
     constExpression ("..." constExpression)?

(61) whileStatement
     WHILE expression **DO** statementSequence **END**

(62) repeatStatement
     REPEAT statementSequence **UNTIL** expression

(63) loopStatement
     LOOP statementSequence **END**

(64) forStatement
     FOR ident ":=" expression **TO** expression (**BY** constExpression)?
     DO statementSequence **END**

(65) tryStatement
     TRY statementSequence
     ON ident **DO** statementSequence
     CONTINUE statementSequence
     END

(66) criticalStatement
     CRITICAL "(" *classInstance* ")"
     statementSequence
     END

(67) *classInstance* = qualident

Expressions

(68) constExpression
     simpleConstExpr (relation simpleConstExpr | "::" namedType)?

(69) relation
     "=" | "#" | "<" | "<=" | ">" | ">=" | **IN** | **IS**

(70) simpleConstExpr
     ("+" | "-")? constTerm (addOperator constTerm)*

(71) addOperator
     "+" | "-" | **OR**

(72) constTerm
     constFactor (mulOperator constFactor)*

(73) mulOperator
     "*" | "/" | **DIV** | **MOD** | **AND** | "&"

```
(74) constFactor
     number | string | qualident | "(" constExpression ")" |
     ( NOT | "~" ) constFactor

(75) designator
     qualident ( designatorTail )?

(76) designatorTail
     ( ( "[" expressionList "]" | "^" ) ( "." ident )* )+

(77) expressionList
     expression ( "," expression )*

(78) expression
     simpleExpression ( relation simpleExpression | "::" namedType )?

(79) simpleExpression
     ( "+" | "-" )? term ( addOperator term )*

(80) term
     factor ( mulOperator factor )*

(81) factor
     number | string | designatorOrProcedureCall | methodInvocation
    "(" expression ")" | ( NOT | "~" ) factor |

(82) designatorOrProcedureCall
     qualident designatorTail? actualParameters?

(83) actualParameters
     "(" expressionList? ")"
```

Value Constructors

```
(84) structuredValue
     "{" ( valueComponent ( "," valueComponent )* )? "}"

(85) valueComponent
     constExpression ( ( BY | ".." ) constExpression  )? |
     structuredValue
```

Identifiers

```
(86) qualident
     ident ( "." ident )*
```


(87) identList
     ident (“,” **ident**)*

Pragmas

(88) pragma
     “<*” (
     (IF | ELSIF) constExpression | ELSE | ENDIF |
     (INFO | WARN | ERROR | FATAL) *compileTimeMessage* |
     INLINE | NOINLINE | FRAMEWORK | IBACTION | IBOUTLET | QUALIFIED |
     implementationDefinedPragma (“+” | “-” | “=” (**ident** | **number**))?
) “*>”

(89) *compileTimeMessage* = string

(90) *implementationDefinedPragma* = **ident**

Terminal Symbols

(91) **ident**
     (“_” | “$” | LETTER) (“_” | “$” | LETTER | DIGIT)*

(92) **colonIdent**
     ident “:”

(93) **number**
     DIGIT+ |
     BINARY-DIGIT+ “B” |
     DIGIT SEDECIMAL-DIGIT* (“C” | “H”) |
     DIGIT+ “.” DIGIT+ (“E” (“+” | “-”)? DIGIT+)?

(94) **string**
     “‘” (CHARACTER | ‘“’)* “‘” |
     ‘“’ (CHARACTER | “‘”)* ‘“’

(95) DIGIT
     “A” .. “Z” | “a” .. “z”

(96) DIGIT
     BINARY-DIGIT | “2” | “3” | “4” | “5” | “6” | “7” | “8” | “9”

(97) BINARY-DIGIT
     “0” | “1”

(98) SEDECIMAL-DIGIT
     DIGIT | “A” | “B” | “C” | “D” | “E” | “F”

```
(99) CHARACTER
     DIGIT | LETTER |
     " " | "!" | "#" | "$" | "%" | "&" | "(" | ")" | "*" | "+" |
     "," | "-" | "." | ":" | ";" | "<" | "=" | ">" | "?" | "@" |
     "[" | "]" | "^" | "_" | "`" | "{" | "|" | "}" | "~" |
     ESCAPE-SEQUENCE

(100)ESCAPE-SEQUENCE
     "\" ( "0" | "n" | "r" | "t" | "\" | "'" | '"' )
```

Ignore Symbols

```
(101)WHITESPACE
     " " | ASCII-TAB

(102)COMMENT
     NESTABLE-COMMENT | NON-NESTABLE-COMMENT | SINGLE-LINE-COMMENT

(103)NESTABLE-COMMENT
     "(*" ( ANY-CHAR | END-OF-LINE )* NESTABLE-COMMENT* "*)"

(104)NON-NESTABLE-COMMENT
     "/*" ( ANY-CHAR | END-OF-LINE )* "*/"

(105)SINGLE-LINE-COMMENT
     "//" ANY-CHAR* END-OF-LINE

(106)ANY-CHAR
     ASCII(8) | ASCII(32) .. ASCII(127) | ANY-UNICODE

(107)END-OF-LINE
     ASCII-LF ASCII-CR? | ASCII-CR ASCII-LF?
```


EBNF Grammar for Objective Modula-2

Reserved Words

```
AND ARRAY BEGIN BY BYCOPY BYREF CASE CLASS CONST CONTINUE CRITICAL DEFINITION
DIV DO ELSE ELSIF END ENUM EXIT FOR FROM IF IMMUTABLE IMPLEMENTATION IMPORT IN
INOUT IS LOOP METHOD MOD MODULE NOT OF ON OPAQUE OPTIONAL OR OUT POINTER PRIVATE
PROCEDURE PROTECTED PROTOCOL PUBLIC RECORD REPEAT RETURN SET SUPER THEN TO TRY
TYPE UNTIL VAR VARIADIC WHILE
```

Reserved Symbols

```
:= + - * / ++ -- & ~ = # < <= > >= ' " ( ) [ ] { } ^ | . , : ; .. :: <* *>
```

Pragma Identifiers

```
IF ELSIF ELSE ENDIF INFO WARN ERROR FATAL INLINE NOINLINE FRAMEWORK IBAction
IBOutlet QUALIFIED
```

Cross Reference

Symbol	Rule	Referenced from
actualParameters	80	52, 79
addOperator	68	67, 76
adoptedProtocols	9	5, 27
anonymousType	18	16, 18, 36
ANY-CHAR	103	100, 101, 102
arrayIndex	22	21
arrayType	21	18, 26
assignmentOrProcedureCall	52	47
attributedFormalType	33	32
baseType	20	19, 24
BINARY-DIGIT	94	90, 93
block	11	2, 38, 43
case	55	54
caseLabelList	56	55
caseLabels	57	56
caseStatement	54	47
CHARACTER	96	91
classInstance	64	63
classType	27	16
colonIdent	89	46, 51
COMMENT	99	-
compilationUnit	1	-
compileTimeMessage	86	85
constantDeclaration	14	12, 13
constExpression	65	7, 14, 23, 37, 57, 61, 71, 82, 85
constFactor	71	69, 71
constTerm	69	67

EBNF Grammar for Objective Modula-2

Symbol	Rule	Referenced from
criticalStatement	63	47
declaration	12	11
definition	13	3
definitionOfModule	3	1
designator	72	52
designatorOrProcedureCall	79	78
designatorTail	73	72, 79
DIGIT	93	88, 90, 95, 96
END-OF-LINE	104	100, 101, 102
enumerationType	19	16
ESCAPE-SEQUENCE	97	96
expression	75	47, 51, 52, 53, 54, 58, 59, 61, 74, 78
expressionList	74	73, 80
factor	78	77, 78
fieldList	26	25
fieldListSequence	25	24, 27
formalParamList	41	39
formalParams	42	41
formalType	34	33, 42, 46
formalTypeList	32	31
forStatement	61	47
handle	44	43
ident	88	6, 8, 13, 14, 15, 36, 38, 39, 40, 43, 44, 46, 50, 51, 61, 62, 73, 83, 84, 85, 87, 89
identList	84	9, 10, 19, 26, 29, 36, 42
ifStatement	53	47
implementationDefinedPragma	87	85
implementationOfModule	4	1

EBNF Grammar for Objective Modula-2

Symbol	Rule	Referenced from
importList	10	2, 3, 5
indexParam	45	43
LETTER	92	88, 96
loopStatement	60	47
machineAddress	37	36
message	51	49
methodArg	46	44
methodDeclaration	43	12
methodHeader	44	5, 13, 43
methodInvocation	49	47, 50, 78
moduleId	6	2, 3
mulOperator	70	69, 77
namedType	17	16, 21, 26, 29, 30, 34, 35
NESTABLE-COMMENT	100	99, 100
NON-NESTABLE-COMMENT	101	99
number	90	71, 78, 85
ordinalConstExpression	23	22
PointerType	30	18
pragma	85	-
priority	7	2
procedureDeclaration	38	12
procedureHeader	39	13, 38
procedureType	32	18, 21, 26
programModule	2	1, 4
protocolId	8	5
protocolModule	5	1
qualident	83	17, 20, 28, 45, 64, 71, 72, 79
receiver	50	49

EBNF Grammar for Objective Modula-2

Symbol	Rule	Referenced from
receiverClass	45	44
receiverType	40	39
recordType	24	18, 21, 26
relation	66	65, 75
repeatStatement	59	47
returnedType	35	31, 39, 44
SEDECIMAL-DIGIT	95	90
setType	29	16
simpleConstExpr	67	65
simpleExpression	76	75
SINGLE-LINE-COMMENT	102	99
statement	47	48
statementSequence	48	11, 53, 54, 55, 58, 59, 60, 61, 62, 63
string	91	71, 78, 86
structuredValue	81	14, 52, 82
superClass	28	27
term	77	76
tryStatement	62	47
type	16	13, 15
typeDeclaration	15	12
valueComponent	82	81
variableDeclaration	36	12, 13
variadicParams	43	41
whileStatement	58	47
WHITESPACE	98	-

Further Reading

http://objective.modula2.net

http://objective.modula2.net/grammar.shtml

