

EBNF Grammar for Objective Modula-2 (Syntax Analysis)

Typographic Conventions

- Production rules are shown in fixed width font
- Reserved words are shown in **UPPERCASE BOLDFACE**
- Other terminal symbols are shown in **lowercase boldface**
- Separator symbols are shown in "double quotes"

Compilation Units

- (1) compilation-unit =
 program | definition-module | implementation-module | protocol-module
- (2) program =
 MODULE ident library-implementation-body **ident** "."
- (3) definition-module =
 DEFINITION MODULE ident
 (library-declaration-body |
 class-declaration-body | category-declaration-body)
 ident "."
- (4) implementation-module =
 IMPLEMENTATION MODULE ident
 { library-implementation-body |
 class-implementation-body | category-implementation-body }
 ident "."
- (5) protocol-module =
 PROTOCOL MODULE ident ";"
 { any-import } { const-declaration | type-declaration }
 { [**OPTIONAL** | **REQUIRED**] method-declaration } **END ident** "."
- (6) library-declaration-body =
 ";" { library-or-class-import }
 { declaration-permitted-in-library-interface } **END**
- (7) class-declaration-body =
 ":" **ident** ";" { any-import }
 { declaration-permitted-in-class-interface } **END**
- (8) category-declaration-body =
 REFINES ident ";" { any-import }
 { declaration-permitted-in-category-interface } **END**
- (9) library-implementation-body =
 ["[" const-expression "]"] ";" { library-or-class-import } block
- (10) class-implementation-body =
 ":" **ident** ";" { any-import }
 { declaration-permitted-in-class } **END**
- (11) category-implementation-body =
 REFINES ident ";" { any-import }
 { declaration-permitted-in-class } **END**

EBNF Grammar for Objective Modula-2 (Syntax Analysis)

Import Lists

- (12) any-import =
 (**IMPORT** [**CLASS** | **PROTOCOL**] ident-list) | unqualified-import ";"
- (13) library-or-class-import =
 (**IMPORT** [**CLASS**] ident-list) | unqualified-import ";"
- (14) unqualified-import =
 FROM ident IMPORT ident-list ";"

Declarations

- (15) declaration-permitted-in-library-interface =
 const-declaration | type-declaration |
 library-variable-declaration | procedure-declaration
- (16) declaration-permitted-in-class-interface =
 const-declaration | type-declaration |
 instance-variable-declaration | method-declaration
- (17) declaration-permitted-in-category-interface =
 const-declaration | type-declaration | method-declaration
- (18) declaration-permitted-in-block =
 const-declaration | type-declaration |
 variable-declaration | procedure-declaration-or-implementation
- (19) declaration-permitted-in-class =
 declarations-permitted-in-block | method-declaration-or-implementation
- (20) const-declaration =
 CONST { **ident** "=" const-expression ";" }
- (21) type-declaration =
 TYPE { **ident** ["=" type-designator] ";" ["< * FORWARD * >"] }
- (22) variable-declaration =
 VAR { ident-list ":" type-designator ";" }
- (23) library-variable-declaration =
 VAR { ident-list ":" type-designator ";" } ["< * IMMUTABLE * >"]
- (24) instance-variable-declaration =
 [**PUBLIC** | **PROTECTED** | **PRIVATE**] **VAR**
 { ident-list ":" type-designator ";" }
- (25) procedure-declaration =
 procedure-declaration-header ";" ["< * FORWARD * >"]
- (26) procedure-declaration-header =
 PROCEDURE ident
 "(" [formal-param-section { ";" formal-param-section }] ")"
 [":" qualified-ident]
- (27) formal-param-section =
 [**VAR**] ident-list ":" formal-type

EBNF Grammar for Objective Modula-2 (Syntax Analysis)

- (28) method-declaration =
 method-declaration-header ";" [<* FORWARD *>]
- (29) method-declaration-header =
 (**CLASS** | **INSTANCE**)
 METHOD (**ident** | method-arg) { method-arg } ":" **ident**

Data Types

- (30) type-designator =
 array-type | non-array-type-designator
- (31) non-array-type-designator =
 qualified-ident | enumeration-type | record-type | set-type |
 pointer-type | procedure-type
- (32) enumeration-type =
 "(" ident-list ")"
- (33) array-type =
 ARRAY array-index
 ({ **OF** "ARRAY array-index" } | ({ "," array-index })
 OF non-array-type-designator
- (34) array-index =
 enumeration-type | ("[" ["0" ".."] const-expression "]")
- (35) record-type =
 RECORD ident-list ":" type-designator
 { ";" ident-list ":" type-designator } **END**
- (36) set-type =
 SET OF (qualified-ident | enumeration-type)
- (37) pointer-type =
 POINTER TO type-designator
- (38) procedure-type =
 PROCEDURE
 ["(" [[**VAR**] formal-type { "," [**VAR**] formal-type }] ")"]
 [":" qualified-ident]
- (39) formal-type =
 [**ARRAY OF**] qualified-ident

Variable and Value Designators

- (40) variable-designator =
 qualified-ident { "." **ident** | "[" expression-list "]" | "^" }
- (41) value-designator =
 qualified-ident { "." **ident** | "[" expression-list "]" | "^" }
 [actual-parameters]
- (42) actual-parameters =
 "(" [expression-list] ")"

EBNF Grammar for Objective Modula-2 (Syntax Analysis)

(43) qualified-ident =
 ident { "." **ident** }

(44) ident-list =
 ident { "," **ident** }

Blocks

(45) procedure-declaration-or-implementation =
 procedure-declaration-header (block | ";" [<* FORWARD *>])

(46) method-declaration-or-implementation =
 method-declaration-header (block | ";" [<* FORWARD *>])

(47) method-arg =
 labeled-ident "(" **ident** ":" **ident** ")"

(48) block =
 { declaration-permitted-in-block }
 [**BEGIN** statement-sequence] **END**

Statements

(49) statement-sequence =
 statement { ";" statement }

(50) statement =
 [assignment | incr-or-decr-statement | procedure-call | message |
 if-statement | case-statement | while-statement | repeat-statement |
 loop-statement | for-statement | return-statement | try-statement |
 critical-statement | **EXIT**]

(51) assignment =
 variable-designator ":"= expression

(52) incr-or-decr-statement =
 variable-designator
 (**postfix-increment-operator** | **postfix-decrement-operator**)

(53) procedure-call =
 variable-designator [actual-parameters]

(54) message =
 "[" **ident** [**ident**]
 (**labeled-ident** expression) { **labeled-ident** expression } "]"

(55) if-statement =
 IF expression **THEN** statement-sequence
 { **ELSIF** expression **THEN** statement-sequence }
 [**ELSE** statement-sequence] **END**

(56) case-statement =
 CASE expression **OF** [case-label-list ":" statement-sequence]
 { "|" case-label-list ":" statement-sequence }
 [**ELSE** statement-sequence] **END**

EBNF Grammar for Objective Modula-2 (Syntax Analysis)

```
(57) case-label-list =  
    const-range { "," const-range }  
  
(58) while-statement =  
    WHILE expression DO statement-sequence END  
  
(59) repeat-statement =  
    REPEAT statement-sequence UNTIL expression  
  
(60) loop-statement =  
    LOOP statement-sequence END  
  
(61) for-statement =  
    FOR ident := expression TO expression [ BY const-expression ]  
    DO statement-sequence END  
  
(62) return-statement =  
    RETURN [ expression ]  
  
(63) try-statement =  
    TRY statement-sequence  
    ON expression DO statement-sequence  
    CONTINUE statement-sequence END  
  
(64) critical-statement =  
    CRITICAL "(" ident ")"  
    statement-sequence END
```

Expressions

```
(65) expression-list =  
    expression { "," expression }  
  
(66) expression =  
    simple-expression [ relational-operator simple-expression ]  
  
(67) simple-expression =  
    [ sign ] term { term-operator term }  
  
(68) term =  
    factor { factor-operator factor }  
  
(69) factor =  
    literal | value-designator | message | unary-expression |  
    "(" expression ")"  
  
(70) unary-expression =  
    unary-operator factor  
  
(71) const-expression =  
    simple-const-expression [ relational-operator simple-const-expression ]  
  
(72) simple-const-expression =  
    [ sign ] const-term { term-operator const-term }  
  
(73) const-term =  
    const-factor { factor-operator const-factor }
```

EBNF Grammar for Objective Modula-2 (Syntax Analysis)

- (74) `const-factor =`
 `literal | qualified-ident | const-set | const-unary-expression |`
 `"(" const-expression ")"`
- (75) `const-set =`
 `[qualified-ident] "{" [const-range { "," const-range }] "}"`
- (76) `const-range =`
 `const-expression [".." const-expression]`
- (77) `const-unary-expression =`
 `unary-operator const-factor`

Operators

- (78) `sign =`
 `"+" | "-"`
- (79) `unary-operator =`
 `NOT | logical-not-operator`
- (80) `relational-operator =`
 `equal-operator | not-equal-operator |`
 `less-than-operator | less-or-equal-operator |`
 `greater-than-operator | greater-or-equal-operator | IN`
- (81) `term-operator =`
 `plus-operator | minus-operator | OR`
- (82) `factor-operator =`
 `multiply-operator | division-operator |`
 `DIV | MOD | AND | logical-and-operator`

Literals

- (83) `literal =`
 `number-literal | character-code-literal | string-literal`
- (84) `number-literal =`
 `octal-whole-number-literal | decimal-whole-number-literal |`
 `sedecimal-whole-number-literal | real-number-literal`
- (85) `character-code-literal =`
 `7-bit-ascii-character-code-literal | unicode-character-code-literal`

Reserved Words

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

EBNF Grammar for Objective Modula-2 (Syntax Analysis)

Terminals

ident, labeled-ident, octal-whole-number-literal, decimal-whole-number-literal, sedecimal-whole-number-literal, 7-bit-ascii-character-code-literal, unicode-character-code-literal, real-number-literal, string-literal, assign-operator, logical-and-operator, logical-not-operator, equal-operator, not-equal-operator, greater-than-operator, greater-or-equal-operator, less-than-operator, less-or-equal-operator, plus-operator, postfix-increment-operator, minus-operator, postfix-decrement-operator, multiply-operator, divide-operator, pointer-dereference-operator, message-prefix

Separators

"(", ")", "[", "]", "{", "}", ".", "..", ",", ":", ";", "|"

Pragmas

<* FORWARD *>, <* FRAMEWORK *>, <* IBAction *>, <* IBOutlet *>, <* IMMUTABLE *>

Cross Reference

Non-terminal symbol	Rule	Referenced from
actual-parameters	42	41, 53
any-import	12	5, 7, 8, 10, 11
array-index	34	33
array-type	33	30
assignment	51	50
block	48	9, 45, 46
case-label-list	57	56
case-statement	56	50
category-declaration-body	8	3
category-implementation-body	11	4
character-code-literal	85	83
class-declaration-body	7	3
class-implementation-body	10	4
compilation-unit	1	–
const-declaration	20	5, 15, 16, 17, 18

EBNF Grammar for Objective Modula-2 (Syntax Analysis)

Non-terminal symbol	Rule	Referenced from
const-expression	71	9, 20, 34, 61, 74, 76
const-factor	74	73, 77
const-range	76	57, 75
const-set	75	74
const-term	73	72
const-unary-expression	77	74
critical-statement	64	50
declaration-permitted-in-block	18	48
declaration-permitted-in-category-interface	17	8
declaration-permitted-in-class	19	10, 11
declaration-permitted-in-class-interface	16	7
declaration-permitted-in-library-interface	15	6
definition-module	3	1
enumeration-type	32	31, 34, 36
expression	66	51, 54, 55, 56, 58, 59, 61, 62
expression-list	65	40, 41, 42
factor	69	68, 70
factor-operator	82	68, 73
for-statement	61	50
formal-param-section	27	26
formal-type	39	27, 38
ident-list	44	12, 13, 14, 22, 23, 24, 27, 32, 35
if-statement	55	50
implementation-module	4	1
incr-or-decr-statement	52	50
instance-variable-declaration	24	16

EBNF Grammar for Objective Modula-2 (Syntax Analysis)

Non-terminal symbol	Rule	Referenced from
library-declaration-body	6	1
library-implementation-body	9	2, 4
library-or-class-import	13	6, 9
library-variable-declaration	23	15
literal	83	69, 74
loop-statement	60	50
message	54	50, 69
method-arg	47	29
method-declaration	28	5, 16, 17, 19
method-declaration-header	29	28, 46
method-declaration-or-implementation	46	19
non-array-type-designator	31	30, 33
number-literal	84	83
pointer-type	37	31
procedure-call	53	50
procedure-declaration	25	15
procedure-declaration-header	26	25, 45
procedure-declaration-or-implementation	45	18
procedure-type	38	31
program	2	1
protocol-module	5	1
qualified-ident	43	26, 31, 36, 38, 39, 40, 41, 74, 75
record-type	35	31
relational-operator	80	66, 71
repeat-statement	59	50
return-statement	62	50

EBNF Grammar for Objective Modula-2 (Syntax Analysis)

Non-terminal symbol	Rule	Referenced from
set-type	36	31
sign	78	67, 72
simple-const-expression	72	71
simple-expression	67	66
statement	50	49
statement-sequence	49	48, 55, 56, 58, 59, 60, 61, 63, 64
term	68	67
term-operator	81	67, 72
try-statement	63	50
type-declaration	21	5, 15, 16, 17, 18
type-designator	30	21, 22, 23, 24, 35, 37
unary-expression	70	69
unary-operator	79	70, 77
unqualified-import	14	12, 13
value-designator	41	69
variable-declaration	22	18
variable-designator	40	51, 52, 53
while-statement	58	50