ECMA

EUROPEAN COMPUTER MANUFACTURERS ASSOCIATION

FORMAL DEFINITION
of the
SYNTAX OF COBOL

September 1970

Free copies of this document are available from ECMA,
European Computer Manufacturers Association,
114 Rue du Rhône — 1204 Geneva (Switzerland)

ECMA

EUROPEAN COMPUTER MANUFACTURERS ASSOCIATION

FORMAL DEFINITION of the SYNTAX OF COBOL

September 1970

FORMAL DEFINITION
of the
SYNTAX OF COBOL

CONTENTS

PREFACE

PREFACE

This formal definition of the syntax of COBOL was prepared by the ECMA Technical Committee on COBOL (TC6).

The work was initially undertaken at the request of the CODASYL COBOL Publication Subcommittee. It resulted in the publication in 1967 of a Preliminary Edition based on COBOL Edition 65. This new edition is based on the ISO Draft Recommendation 1989 on COBOL.

The document comprises four distinct parts and an appendix. The first part briefly describes the notation used, the second part is the formal definition of the COBOL syntax, the third part is an index showing where each meta-variable is defined and where it is used, the fourth part contains explanatory notes for those definitions marked with an asterisk, and the appendix is a complete and rigorous description of the metalanguage. The second part is divided into three sections: syntactic definitions of general nature, level 1 syntax defining the COBOL text and level 2 syntax defining the COBOL program. The level 1 syntax describes the basic structure of the COBOL language. It defines a set of strings, called COBOL texts, in terms of generalized words (including COBOL words, literals, arithmetic and relational operators, etc.) and word separators. The level 2 syntax describes the detailed structure of the COBOL language. It defines a set of strings, called COBOL programs, in terms of specific sequences of generalized words and word separators. Although a COBOL text and a COBOL program have each been defined as a string of characters, an attempt has been made to show the relationship between such a string and the Reference Format.

The metalanguage used is an extension of the metalanguage used in the ALGOL 60 Report, known as the Backus normal form. It is introduced in the first part: "Introduction to the notation used" and described in detail in the appendix under the title "Formalism for syntactical definition". Most extensions have been introduced to reduce the number and complexity of production rules constituting the formal definition of the COBOL syntax. For example certain extensions greatly simplify the description of the nested structure of records. Whenever these extensions are used, the usual Backus notation, based on Chomsky context-free grammars (type 2), could have been used. However, the convention adopted to show relationship between declaration of data-names and the subsequent use of those data-names is different in that this relationship could not be expressed in Backus notation. This is a well known context-dependent aspect of programming languages. English text has been used where needed to adequately supplement the metalanguage.

It has been difficult to decide whether some COBOL rules should be included in the syntax and somewhat arbitrary decisions had to be made. The level of detail expressed in the production rules is also somewhat arbitrary. It is often founded on an attempt to facilitate the use of this formal definition by the human reader, in conjunction with the existing descriptions of COBOL. For the same reason, the names of metavariables have been chosen to reflect their meaning, and the names defined in the draft ISO Recommendation on COBOL have been used wherever feasible.

The application of the production rules given in level 2 syntax will generate all valid COBOL programs. However, invalid programs will also be generated. For example the following is not reflected:

- uniqueness of names
- relationship between qualifiers and the corresponding data herarchy.
- relationships between subscripts or indices and the corresponding table declarations
- some relationships between clauses and/or statements
- possible indentation of data description entries.

With the exceptions mentioned above, this formal definition is believed to be in agreement with the ISO Recommendation on COBOL.

However, the modular structure of the ISO Recommendation is not reflected; the syntax shown applies to the combination of the upper levels of all modules.

INTRODUCTION TO THE NOTATION

Introduction to the NOTATION

1. General

In the following it is assumed that the reader is familiar with the standard COBOL specification. This informal explanation is intended to further the understanding of the notation used in this Formal Syntactic Definition by way of examples where the new symbols are described in the order of appearance. It is followed by a summary of all the symbols concerned (*).

Observe that the standard COBOL specification already takes advantage of the existence of a kind of formal syntactic definition as shown by the Formats. The present Formal Syntactic Definition is intended to be merely more rigorous, leaving less or (hopefully) no place for ambiguous interpretation of the syntax.

2. Informal Approach through Examples

2.1 As a first example consider entry D124 (page 82):

```
D124    <block-clause>::=
              BLOCK # [CONTAINS #]
             [<positive-integer> # TO #]
              <positive-integer>
             ↓[# CHARACTERS] ! # RECORDS ↑
```

and the corresponding format of the COBOL specification (ref. 2 SEQ, chapter 7, 2.3.2) :

BLOCK CONTAINS [integer-1 TO] integer-2 { CHARACTERS / RECORDS }

Now, compare these two descriptions element by element :

Formal Syntactic Definition	COBOL Specification Format
BLOCK #	BLOCK
[CONTAINS #]	CONTAINS
[<positive-integer># TO #]	[integer-1 TO]
<positive-integer>	integer-2
↓[# CHARACTERS] !# RECORDS↑	{ CHARACTERS / RECORDS }

It will be seen that the formal notation has been designed to be as closely related as possible to the COBOL specification formats. Thus

a) the appearance of upper-case letters means, in both descriptions, the actual occurrence of these letters in program text;

b) in both descriptions, square brackets [] mean that the contents may or may not appear in the program text at the user's option;
Note, however, some differences :

c) underlined upper-case words of the COBOL specification are not underlined in the formal notation, whereas

---

* A more formal and complete description of the notation is to be found in the Appendix.

d) noise words, shown with non-underlined upper-case words in the COBOL formats, are described in the formal notation by upper-case words enclosed in square brackets (comprising only the noise word and possibly the symbol #, which is explained below),

e.g. [CONTAINS #] instead of CONTAINS
[# CHARACTERS] instead of CHARACTERS

e) the symbol ! ('or' represented by exclamation mark) has been introduced to express that an alternative has to be selected from two or more possibilities; thus

[# CHARACTERS] ! # RECORDS

means that either # CHARACTERS or # RECORDS exclusively, may appear in the program text. If the former, it may be omitted as shown by the surrounding square brackets.

It will be noticed that the right hand part of D 124 shows the exact punctuation allowed or required in the COBOL text :

f) # means the actual occurrence of one or more spaces. Note also the introduction of the symbols

< > { }

These are various types of delimiters.

g) Syntactic units representing parts of the program text to be filled in at the user's option from a given set of possible words, strings of words or other entities are represented by lower-case words (or hyphenated words) enclosed in Backus brackets < >. In the above example

<positive-integer>

is such an instance. In fact such a lower-case word enclosed in Backus brackets can be considered as a variable, that is, a notation to be replaced by a variable content; "variables" of this kind have been given the name meta-variables.

As will be seen a meta-variable can be used to represent not only an element (like positive-integer), but any specified portion of a COBOL program. For instance, the whole compound

```
BLOCK # [CONTAINS #]
[<positive-integer> # TO #]
<positive-integer>
{[# CHARACTERS] ! # RECORDS }
```

is represented in the example by the single meta-variable

<block-clause>

This is possible because at entry D 124 <block-clause> has been precisely defined as being equivalent to the above compound.

h) This equivalence is specified by the symbol ::= separating <block-clause> from its definition which is given by the compound.
The symbol ::= means "is defined by" and the whole entry is called a meta-definition.

i) Notice in addition that the alternatives

[# CHARACTERS] ! # RECORDS

appear enclosed in the special braces ⦃ ⦄ as follows :

⦃[# CHARACTERS] ! # RECORDS ⦄

The aim of the special braces

⦃ ⦄

is to delimit a specific portion in the formal notation. In this particular case the function of the braces is to determine the scope of the alternatives defined by the ! operator. Later, other uses of delimitation by braces will be described (in particular in connection with the ellipsis).

2.2 As a second example, consider the D 16 entry (page 57):

D 16 <fd-clauses> ::=

⦃[<;><block-clause>] ⁝
[<;><record-contains-clause>] ⁝
<;><label-records-clause> ⁝
[<;><value-of-clause>] ⁝
⦃[<;><data-records-clause>] !
<;><report-clause>⦄⦄

Here is a whole structure named (i.e. defining) <fd-clauses>, of which the above mentioned <block-clause> is just one constituent part, and moreover an optional one, since it is enclosed in square brackets.

This second example introduces some new symbols.

j) There is the symbol <;> representing the occurrence of one or more spaces optionally preceded by a semicolon.

k) Then there are the permutation brackets

⦃ ⦄

and the permutation separators

⁝

meaning that all syntactic units contained between the permutation brackets and delimited by the permutation separators may appear in any order in the program text, at the user's option.

It is now possible to interpret the meta-definition D 16. Remember that the ! designates (and separates) two possible alternatives. Thus, D 16 means that

<fd-clauses>

represents a compound of clauses optionally preceded by a semicolon, where the compound comprises one mandatory clause, namely

<label-records-clause>

three optional clauses, namely

<block-clause>
<record-contains-clause>
<value-of-clause>

and either the <data-records-clause> which is optional or the <report-clause> and, further, that these clauses may appear in any order by virtue of the permutation brackets.

In the same way as <block-clause> was defined in the first example as D124, the other meta-variables

<record-contains-clause>
<label-records-clause>
<value-of-clause>
<data-records-clause>
<report-clause>

are defined elsewhere in the book, that is at their proper entries.
In D 16 only the meta-variable <fd-clauses> is defined.

2.3 Going one more step backwards consider now the entry D 15 (page57):

D 15 <ms-file-description> ::=

←FD # ↑
⟨<sequential-ms-file-name-declaration>!
<random-ms-file-name-declaration>⟩
⟨<;><copy-clause>!
<fd-clauses>⟩<.>

In this third example new symbols are introduced, namely <.>, ← and ↑

l) The symbol <.> represents the actual occurrence of a period followed by one or more spaces.

m) The symbol ← (horizontal arrow) means that the next character must be in area A of a new line of the reference format.

n) The symbol ↑ (vertical arrow) means that the next character must be in area B of either the same line or the following one of the reference format.

Thus, this third example reads

<ms-file-description> is defined by the letter F in area A of a new line, followed by the letter D, followed by one or more spaces, then, beginning in area B,

either a <sequential-file-name-declaration>,
or a <random-file-name-declaration>,

then

either one or more spaces optionally preceded by a semi-colon, and followed by a <copy-clause>
or the <fd-clauses>

in each case, followed by a period and one or more spaces.

2.4 Looking further backwards it will be seen that <ms-file-description> appears in the definition of <file-specification> :

D 5 <file-specification> ::=

<non-ms-file-description>
[<non-ms-record-description>]...!
<ms-file-description>
[<ms-record-description>]...

This meta-definition demonstrates the usage of the last symbol to be described in this introduction, namely the repetition symbol, called ellipsis, and represented by three dots ...

o) The ellipsis specifies that the immediately preceding syntactic unit may be repeated any number of times at the user's option, exactly as stated in the familiar COBOL specification. The delimitation of the "immediately preceding syntactic unit" may be found by searching for the immediately preceding closing brace, bracket or Backus bracket and finding the logically matching opening brace, bracket or Backus bracket. Between these two lies the syntactic unit concerned.

Examining D 5, it will easily be seen that <file-specification> is defined as one of two alternatives. For instance, the second alternative (after the ! symbol) consists of

> the appearance of <ms-file-description> followed by no, one, or more occurrences of <ms-record-description>

Notice also that since the latter is a meta-variable these successive occurrences will in general represent successive different representations of <ms-record-description>.

2.5 Following the track still further backwards, it will be seen that <file-specification> appears in the definition of <file-section-body> (D4), which, in turn, appears in <file-section> (D3), and so on through <data-division-body> (D2), back, eventually, to <data-division> (D1).

So, the Data Division is defined by means of a chaining of successive meta-definitions, forming a completely determined tree and showing the exact layout of the Data Division of every syntactically correct COBOL program.

Having acquainted himself with the notation by working through the above examples, the reader will then find no difficulty going through the following summary thereby recapitulating and refining the knowledge gained.

3. Summary of the NOTATION used in the Formal Syntactic Definition

3.1 The Meta-Definition

A meta-definition is a syntax rule expressed in a formal notation. Each meta-definition defines a new syntactical entity as a specific arrangement of COBOL characters and other syntactical entities. The name of the syntactical entity to be defined appears on the left-hand side of the definition symbol which is followed by the definition.

3.2 The Metalanguage

3.2.1 Definition Symbol

The following symbol, ::=, is used as definition symbol.

3.2.2 Meta-Variables (Syntactical Entities)

Lower-case words and other symbols enclosed in Backus brackets represent parts of the COBOL text, whose permissible structures are defined outside the containing meta-definition.

3.2.3 Meta-Constants

Upper-case words, numbers and special characters not enclosed in Backus brackets represent the actual occurrence of these upper-case words, numbers and special characters in the COBOL text.

3.2.4 Meta-Operators and Meta-Delimiters

3.2.4a Alternatives

The OR Symbol, !, indicates and separates alternatives.

3.2.4b Braces

Braces, { }, enclosing a portion of a meta-definition, are used for two different functions:

i) to indicate that a selection of one of the options listed between the braces and separated by the OR symbol, !, must be made;

ii) to delimit a portion of the meta-definition, for instance in connection with repetition (see 3.2.4c below).

3.2.4c Repitition

An ellipsis, ..., indicates possible repitition of the preceding element. The preceding element may be a COBOL character, a meta-variable or a group of such elements enclosed in brackets or braces.

3.2.4d Permutation

When the order of elements is immaterial these elements are separated by the permutation-separator, ‡ , and grouped within permutation-brackets { }.
Unless otherwise specified the sequence of elements shown is compulsory.

3.3 Other Conventions

The following symbols are used :

b	to represent the COBOL character "space"
#	to represent one or more b
←	to show that the following element must start in area A of a new line of the reference format
↑	to show that the following element must start in area B of the reference format.

FORMAL DEFINITION OF COBOL SYNTAX

G. SYNTACTIC DEFINITIONS OF GENERAL NATURE

EMPTY

G1 <empty> ::=

COBOL GRAPHICS

G2 <non-zero-digit> ::=
1 ! 2 ! 3 ! 4 ! 5 ! 6 ! 7 ! 8 ! 9

G3 <zero-digit> ::=
0

G4 <digit> ::=
<zero-digit> ! <non-zero-digit>

G5 <letter> ::=
A ! B ! C ! D ! E ! F ! G !
H ! I ! J ! K ! L ! M ! N !
O ! P ! Q ! R ! S ! T ! U !
V ! W ! X ! Y ! Z

* G6 <space> ::=
$\not{b}$

G7 <arithmetic-expression-character> ::=
+ ! - ! * ! /

* G8 <relation-character> ::=
> ! < ! =

* G9 <currency-sign> ::=
$\not{\$}$

G10 <terminating-character> ::=
. ! , ! ;

G11 <quotation-mark> ::=
"

G12 <parenthesis> ::=
(!)

G13 <proper-punctuation-character> ::=
<terminating-character> !
<quotation-mark> !
<parenthesis>

G14 <special-character> ::=
<arithmetic-expression-character> !
<relation-character> !
<currency-sign> !
<proper-punctuation-character>

G15 <cobol-character> ::=
<digit> !
<letter> !
<space> !
<special-character>

G16 <computer-character> ::=
<cobol-character> !
<additional-data-character>

COBOL CONTROLS

* G17 <strophe-mark> :=
←

* G18 <skip-into-area-b> ::=
↑

SOME FREQUENTLY USED SEPARATORS

G19 <spaces> ::=
<space>...

For readability purposes, the abbreviation # will be used for <spaces> .

G20 <.> ::=
. #

G21 <,> ::=
[,] #

G22 <;> ::=
[;] #

WORD

G23 <word-element> ::=
<digit> ! <letter> ! -

G24 <word-terminator> ::=
<digit> ! <letter>

G25 <word> ::=
<word-terminator>
[[<word-element>...]
<word-terminator>]
The maximum number of characters is 30.

G26 <alpha-word> ::=
<word> containing <letter>

G27 <non-reserved-word> ::=
<word> diff <reserved-word>

G28 <non-reserved-alpha-word> ::=
<alpha-word> diff <reserved-word>

PROPER NONNUMERIC LITERAL

G29 <literal-string> ::=
<computer-character>...
not containing "
The maximum number of characters is 120.

G30 <proper-nonnumeric-literal> ::=
" <literal-string> "

PROPER NUMERIC LITERAL

G31 <integer> ::=
<digit>...
The maximum number of digits is 18.

G32 <decimal-fraction> ::=
<decimal-point> <integer>

G33 <unsigned-proper-numeric-literal> ::=
<integer> !
<decimal-fraction> !
<integer> <decimal-fraction>
The maximum number of digits is 18.

G34 <sign> ::=
+ ! -

G35 <proper-numeric-literal> ::=
[<sign>] <unsigned-proper-numeric-literal>

G36 <positive-integer> ::=
<integer>
containing <non-zero-digit>

FIGURATIVE CONSTANT

G37 <simple-figurative-constant> ::=
ZERO ! ZEROS ! ZEROES !
SPACE ! SPACES !
HIGH-VALUE ! HIGH-VALUES !
LOW-VALUE ! LOW-VALUES !
QUOTE ! QUOTES

G38 <compound-figurative-constant> ::=
ALL # <simple-figurative-constant> !
ALL # <proper-nonnumeric-literal>

G39 <figurative-constant> ::=
<simple-figurative-constant> !
<compound-figurative-constant>

G40 <zero-figurative-constant> ::=
ZERO ! ZEROS ! ZEROES

LITERAL

G41 <nonnumeric-literal> ::=
<proper-nonnumeric-literal> !
<figurative-constant>

G42 <numeric-literal> ::=
<proper-numeric-literal> !
<zero-figurative-constant>

G43 <literal> ::=
<nonnumeric-literal> !
<numeric-literal>

ARITHMETIC OPERATOR

G44 <arithmetic-operator> ::=
+ ! - ! * ! / ! **

PROPER RELATIONAL OPERATOR

* G45 <proper-relational-operator> ::=
> ! < ! =

PICTURE CHARACTER STRING

G46 <picture-character-string> ::=
<picture>

COMMENT STRING

G47 <comment-string> ::=
<computer-character>...
not containing <.>

OTHER LANGUAGE STRING

G48 <other-language-string> ::=
<computer-character>...

T. COBOL TEXT

SEPARATORS

T1 <separator> ::=
! <.> ! <,> ! <;>

T2 <other-language-string-terminator> ::=
<empty>

T3 <generalized-separator> ::=
<separator> !
<other-language-string-terminator>

GENERALIZED CHARACTER-STRING

T4 <generalized-character-string-type-one> ::=
<word> !
<proper-nonnumeric-literal> !
<proper-numeric-literal> !
<arithmetic-operator> !
<proper-relational-operator>

T5 <generalized-character-string-type-two> ::=
<picture-character-string> !
<comment-string> !
<other-language-string>

T6 <generalized-character-string> ::=
<generalized-character-string-type-one> !
<generalized-character-string-type-two>

STRUCTURE OF COBOL TEXT

T7 <structure> ::=
<generalized-character-string> !
<structure>
<generalized-separator> [<skip-into-area-b>]
<structure> !
(<structure>)

T8 <strophe> ::=
<strophe-mark>
<structure>
<separator>

T9 <cobol-text> ::=
<strophe>...

N. NAMES DEFINED BY THE IMPLEMENTOR

HARDWARE NAMES

N1 <computer-name> ::=
<word>
specified by the implementor

N2 <implementor-name-for-type-of-io-unit> ::=
<word>
specified by the implementor

N3 <implementor-name-for-individual-io-unit> ::=
<word>
specified by the implementor

N4 <implementor-name-for-rerun-medium> ::=
<word>
specified by the implementor

N5 <implementor-name-for-individual-switch> ::=
<word>
specified by the implementor

OTHER NAMES

N6 <additional-data-character> ::=

This proper-meta-variable is specified by the implementor.

N7 <implementor-name-for-paper-advance> ::=
<word>
specified by the implementor

N8 <implementor-name-for-code-for-report-groups> ::=
<word>
specified by the implementor

N9 <other-language-name> ::=
<word>
specified by the implementor

C. COBOL PROGRAM

COBOL PROGRAM STRUCTURE

C1 <cobol-program> ::=
<identification-division>
<environment-division>
<data-division>
<procedure-division>

I. IDENTIFICATION DIVISION

IDENTIFICATION DIVISION STRUCTURE

I1 <identification-division> ::=
←IDENTIFICATION # DIVISION <.>
<identification-division-body>

I2 <identification-division-body> ::=
<program-id-paragraph>
[<author-paragraph>]
[<installation-paragraph>]
[<date-written-paragraph>]
[<date-compiled-paragraph>]
[<security-paragraph>]
[<remarks-paragraph>]

PROGRAM-ID PARAGRAPH

I3 <program-id-paragraph> ::=
←PROGRAM-ID <.>
<program-id-paragraph-body>

I4 <program-id-paragraph-body> ::=
<program-id-entry>

I5 <program-id-entry> ::=
<program-name-declaration> <.>

I6 <program-name-declaration> ::=
<non-reserved-word>

DATE-COMPILED PARAGRAPH

I7 <date-compiled-paragraph> ::=
←DATE-COMPILED <.>
[<comment-paragraph_body>]

OTHER PARAGRAPHS

I8 <author-paragraph> ::=
←AUTHOR <.>
[<comment-paragraph-body>]

I9 <installation-paragraph> ::=
←INSTALLATION <.>
[<comment-paragraph-body>]

I10 <date-written-paragraph> ::=
←DATE-WRITTEN <.>
[<comment-paragraph-body>]

I11 <security-paragraph> ::=
←SECURITY <.>
[<comment-paragraph-body>]

I12 <remarks-paragraph> ::=
←REMARKS <.>
[<comment-paragraph-body>]

COMMENT PARAGRAPH BODY

I13 <comment-paragraph-body> ::=
<comment-entry>...

I14 <comment-entry> ::=
<comment-string> <.>

E. ENVIRONMENT DIVISION

ENVIRONMENT DIVISION STRUCTURE

E1 <environment-division> ::=
←ENVIRONMENT # DIVISION <.>
<environment-division-body>

E2 <environment-division-body> ::=
<configuration-section>
[<input-output-section>]

CONFIGURATION SECTION STRUCTURE

E3 <configuration-section> ::=
←CONFIGURATION # SECTION <.>
<configuration-section-body>

E4 <configuration-section-body> :=
<source-computer-paragraph>
<object-computer-paragraph>
[<special-names-paragraph>]

INPUT-OUTPUT SECTION STRUCTURE

E5 <input-output-section> ::=
←INPUT-OUTPUT # SECTION <.>
<input-output-section-body>

E6 <input-output-section-body> ::=
<file-control-paragraph>
[<i-o-control-paragraph>]

SOURCE-COMPUTER PARAGRAPH

E7 <source-computer-paragraph> ::=
←SOURCE-COMPUTER <.>
{<copy-entry> !
<source-computer-paragraph-body>}

E8 <source-computer-paragraph-body> ::=
<source-computer-entry>

E9 <source-computer-entry> ::=
<computer-name> <.>

OBJECT-COMPUTER PARAGRAPH

E10 <object-computer-paragraph> ::=
←OBJECT-COMPUTER <.>
{<copy-entry> !
<object-computer-paragraph-body>}

E11 <object-computer-paragraph-body> ::=
<object-computer-entry>

E12 <object-computer-entry> ::=
<computer-name> [<,> <memory-size-clause>]
[<,> <segment-limit-clause>] <.>

E13 <memory-size-clause> ::=
MEMORY # [SIZE #]
<integer> # {WORDS ! CHARACTERS ! MODULES}

SEGMENT-LIMIT CLAUSE

E14 <segment-limit-clause> ::=
SEGMENT-LIMIT # IS #
<priority-number-limit>

E15 <priority-number-limit> ::=
[0]... {<empty> ! 1 ! 2 ! 3 ! 4} <digit>
diff {0}...

SPECIAL-NAMES PARAGRAPH

E16 <special-names-paragraph> ::=
←SPECIAL-NAMES <.>
{<copy-entry> !
<special-names-paragraph-body>}

E17 <special-names-paragraph-body> ::=
<special-names-entry>

E18 <special-names-entry> ::=
<special-names-clauses>
[<,> <currency-sign-clause>]
[<,> <decimal-point-clause>] <.> !
<currency-sign-clause>
[<,> <decimal-point-clause>] <.> !
<decimal-point-clause> <.>

E19 <special-names-clauses> ::=
<special-names-clause>
[<,> <special-names-clause>]...

SPECIAL-NAMES CLAUSE

E20 <special-names-clause> ::=
<non-switch-special-names-clause. !
<switch-special-names-clause>

E21 <non-switch-special-names-clause> ::=
<implementor-name-for-individual-io-unit> #
IS #
<mnemonic-name-declaration-for-individual-io-unit>
!
<implementor-name-for-type-of-io-unit> #
IS #
<mnemonic-name-declaration-for-type-of-io-unit> !
<implementor-name-for-paper-advance> #
IS #
<mnemonic-name-declaration-for-paper-advance> !
<implementor-name-for-code-for-report-groups> #
IS #
<mnemonic-name-declaration-for-code-for-report-groups>

E22 <switch-special-names-clause> ::=
<implementor-name-for-individual-switch> #
IS #
<mnemonic-name-declaration-for-individual-switch>
[<,>
<switch-status-name-assignment>] !
<implementor-name-for-individual-switch> #
<switch-status-name-assignment>

E23 <mnemonic-name-declaration-for-individual-io-unit> ::=
<non-reserved-word>

E24 <mnemonic-name-declaration-for-type-of-io-unit> ::=
<non-reserved-word>

E25 <mnemonic-name-declaration-for-paper-advance> ::=
<non-reserved-word>

E26 <mnemonic-name-declaration-for-code-for-report-groups>
::=
<non-reserved-word>

E27 <mnemonic-name-declaration-for-individual-switch> ::=
<non-reserved-word>

E28 <mnemonic-name-for-individual-io-unit> ::=
<non-reserved-word> which appears as a
<mnemonic-name-declaration-for-individual-io-unit>

E29 <mnemonic-name-for-type-of-io-unit> ::=
<non-reserved-word> which appears as a
<mnemonic-name-declaration-for-type-of-io-unit>

E30 <mnemonic-name-for-paper-advance> ::=
<non-reserved-word> which appears as a
<mnemonic-name-declaration-for-paper-advance>

E31 <mnemonic-name-for-code-for-report-groups> ::=
<non-reserved-word> which appears as a
<mnemonic-name-declaration-for-code-for-report-
groups>

E32 <switch-status-name-assignment> ::=
    <on-status> [<,> <off-status>] !
    <off-status> [<,> <on-status>]

E33 <on-status> ::=
    ON # [STATUS #] IS #
    <switch-status-name-declaration>

E34 <off-status> ::=
    OFF # [STATUS #] IS #
    <switch-status-name-declaration>

E35 <switch-status-name-declaration> ::=
    <non-reserved-alpha-word>

E36 <switch-status-name> ::=
    <non-reserved-alpha-word> which appears as a
    <switch-status-name-declaration>

CURRENCY-SIGN CLAUSE

E37 <currency-sign-clause> ::=
CURRENCY # [SIGN #] IS #
" <currency-sign-declaration> "

E38 <currency-sign-declaration> ::=
<possible-character-for-currency-sign>

E39 <possible-character-for-currency-sign> ::=
<computer-character> diff
{<digit> !
{A ! B ! C ! D ! P ! R ! S ! V ! X ! Z ! <space>} !
{+ ! - ! * ! . ! , ! ; ! " ! (!)}}

E40 <currency-symbol> ::=
$! <possible-character-for-currency-sign>
which appears as a <currency-sign-declaration> .

This language element is dependent on the
individual COBOL source program.
It equals $,
if no <currency-sign-declaration> is present
in the <special-names-paragraph> .
It equals
<possible-character-for-currency-sign>
which appears as a
<currency-sign-declaration> ,
if a <currency-sign-declaration> is present
in the <special-names-paragraph> .

E41 <cs> ::=
<currency-symbol>

DECIMAL-POINT CLAUSE

E42 <decimal-point-clause> ::=
DECIMAL-POINT # IS # COMMA

E43 <decimal-point> ::=
. ! ,

This language element is dependent on the individual COBOL source program.
It equals . (period) ,
if no <decimal-point-clause> is present in the <special-names-paragraph> .
It equals , (comma) ,
if a <decimal-point-clause> is present in the <special-names-paragraph> .

E44 <digit-separator> ::=
. ! ,

This language element is dependent on the individual COBOL source program.
It equals , (comma) ,
if no <decimal-point-clause> is present in the <special-names-paragraph> .
It equals . (period) ,
if a <decimal-point-clause> is present in the <special-names-paragraph> .

FILE-CONTROL PARAGRAPH

E45 <file-control-paragraph> ::=
←FILE-CONTROL <.>
{<copy-entry> !
<file-control-paragraph-body>}

E46 <file-control-paragraph-body> ::=
<file-control-entry>...

E47 <file-control-entry> ::=
<file-control-entry-for-non-ms-file> !
<file-control-entry-for-sequential-ms-file> !
<file-control-entry-for-random-ms-file> !
<file-control-entry-for-sort-file>

E48 <file-control-entry-for-non-ms-file> ::=
<select-clause-for-non-ms-file>
{<assign-clause> ! <sort-output-assign-clause>}
[# <multiple-reel-clause>]
[<,> <alternate-area-clause>]
<.>

E49 <file-control-entry-for-sequential-ms-file> ::=
<select-clause-for-sequential-ms-file>
{<assign_clause> ! <sort-output-assign-clause>}
[# <multiple-unit-clause>]
[<,> <alternate-area-clause>]
[<,> <file-limit-clause>]
<,> <access-mode-sequential-clause.
[<,> <processing-mode-sequential-clause>]
[<,> <actual-key-clause>]
<.>

E50 <file-control-entry-for-random-ms-file> ::=
<select-clause-for-random-ms-file>
<assign-clause>
[<,> <file-limt-clause>]
<,> <access-mode-random-clause>
<,> <processing-mode-sequential-clause>
<,> <actual-key-clause>
<.>

E51 <file-control-entry-for-sort-file> ::=
<select-clause-for-sort-file>
<assign-clause> <.>

SELECT CLAUSE

E52 <select-clause-for-non-ms-file> ::=
SELECT
[# <optional-phrase>]
<non-ms-file-name>

E53 <select-clause-for-sequential-ms-file> ::=
SELECT
[# <optional-phrase>]
<sequential-ms-file-name>

E54 <select-clause-for-random-ms-file> ::=
SELECT
<random-ms-file-name>

E55 <select-clause-for-sort-file> ::=
SELECT
<sort-file-name>

E56 <optional-phrase> ::=
OPTIONAL

ASSIGN CLAUSE

```
E57   <assign-clause> ::=
          <assign-type-clause> !
          <assign-individual-units-clause>

E58   <assign-type-clause> ::=
          ASSIGN [# TO]
          [# <integer>]
          # <implementor-name-for-type-of-io-unit>

E59   <assign-individual-units-clause> ::=
          ASSIGN [# TO]
          # <implementor-name-for-individual-io-unit>
          [<,> <implementor-name-for-individual-io-unit>]...

E60   <sort-output-assign-clause> ::=
          ASSIGN [# TO]
          # <implementor-name-for-individual-io-unit>
          [<,> <implementor-name-for-individual-io-unit>]...
          # OR
          # <implementor-name-for-individual-io-unit>
          [<,> <implementor-name-for-individual-io-unit>]...
```

MULTIPLE REEL/UNIT CLAUSE

E61 <multiple-reel-clause> ::=
[FOR #] MULTIPLE
REEL

E62 <multiple-unit-clause> ::=
[FOR #] MULTIPLE
UNIT

ALTERNATE AREA CLAUSE

E63 <alternate-area-clause> ::=
RESERVE # {<integer> ! NO}
[# ALTERNATE]
{[# AREA] ! [# AREAS]}

FILE-LIMIT CLAUSE

E64 <file-limit-clause> ::=
{FILE-LIMIT # IS ! FILE-LIMITS # ARE}
<file-limit> # {THROUGH ! THRU} # <file-limit>
[<,> <file-limit> # {THROUGH ! THRU} # <file-limit>]...

E65 <file-limit> ::=
<data-name-identifier> ! <literal>

ACCESS MODE CLAUSE

E66 <access-mode-sequential-clause> ::=
ACCESS # [MODE #] IS # SEQUENTIAL

E67 <access-mode-random-clause> ::=
ACCESS # [MODE #] IS # RANDOM

PROCESSING MODE CLAUSE

E68 <processing-mode-sequential-clause> ::=
PROCESSING # [MODE #] IS # SEQUENTIAL

KEY CLAUSE

E69 <actual-key-clause> ::=
ACTUAL # [KEY #] IS #
<data-name-identifier>

I-O-CONTROL PARAGRAPH

```
E70   <i-o-control-paragraph> ::=
          ←I-O-CONTROL <.>
          {<copy-entry> !
          <i-o-control-paragraph-body>}

E71   <i-o-control-paragraph-body> ::=
          <i-o-control-entry>

E72   <i-o-control-entry> ::=
          <rerun-clauses>
          [<;> <same-clauses>]
          [<;> <multiple-file-clauses>] <.> !
          <same-clauses>
          [<;> <multiple-file-clauses>] <.> !
          <multiple-file-clauses> <.>

E73   <rerun-clauses> ::=
          <rerun-clause>
          [<;> <rerun-clause>]...

E74   <same-clauses> ::=
          <same-clause>
          [<;> <same-clause>]...

E75   <multiple-file-clauses> ::=
          <multiple-file-clause>
          [<;> <multiple-file-clause>]...
```

RERUN CLAUSE

E76 <rerun-clause> ::=
RERUN # [<rerun-medium> #] <rerun-condition-1> !
RERUN # <rerun-medium> # <rerun-condition-2>

E77 <rerun-medium> ::=
ON # <non-sort-file-name> !
ON # <implementor-name-for-rerun-medium>

E78 <rerun-condition-1> ::=
<end-of-reel-rerun-condition>

E79 <end-of-reel-rerun-condition> ::=
[EVERY #] [END # [OF #]] {REEL ! UNIT} #
[OF #] <non-sort-file-name>

E80 <rerun-condition-2> ::=
<integer-records-rerun-condition> !
<clock-units-rerun-condition> !
<switch-rerun-condition>

E81 <integer-records-rerun-condition> ::=
[EVERY #] <positive-integer> # RECORDS #
[OF #] <non-sort-file-name>

E82 <clock-units-rerun-condition> ::=
[EVERY #] <positive-integer> # CLOCK-UNITS

E83 <switch-rerun-condition> ::=
[EVERY #] <switch-status-name>

SAME CLAUSE

E84 <same-clause> :=
<same-record-area-clause> !
<same-block-area-clause> !
<same-sort-area-clause>

E85 <same-record-area-clause> ::=
SAME # RECORD # [AREA #] [FOR #]
<file-name> {<,> <file-name>}...

E86 <same-block-area-clause> ::=
SAME # [AREA #] [FOR #]
<non-sort-file-name> {<,> <non-sort-file-name>}...

E87 <same-sort-area-clause> ::=
SAME # SORT # [AREA #] [FOR #]
<file-name> {<,> <file-name>}...

MULTIPLE FILE CLAUSE

E88 <multiple-file-clause> ::=
MULTIPLE # FILE # [TAPE #] [CONTAINS #]
<non-sort-file-name> [# POSITION # <integer>]
[<,> <non-sort-file-name> [# POSITION #
<integer>]]...

D. DATA DIVISION

DATA DIVISION STRUCTURE

D1 <data-division> ::=
←DATA # DIVISION <.>
<data-division-body>

D2 <data-division-body> ::=
<file-section>
[<working-storage-section>]
[<report-section>] !
[<working-storage-section>]

FILE SECTION

D3 <file-section> ::=
←FILE # SECTION <.>
<file-section-body>

D4 <file-section-body> ::=
•{<file-specification> !
<sort-file-specification>}...

D5 <file-specification> ::=
<non-ms-file-description>
[<non-ms-record-description>]... !
<ms-file-description>
[<ms-record-description>]...

D6 <sort-file-specification> ::=
<sort-file-description>
<sort-record-description>...

WORKING-STORAGE SECTION

D7 <working-storage-section. ::=
←WORKING-STORAGE # SECTION <.>
<working-storage-section-body>

D8 <working-storage-section-body> ::=
[<77-descriptions>]
<ws-record-descriptions> !
<77-descriptions>

D9 <77-descriptions> ::=
{<77-description>
[<redefining-77-description>]...}...

D10 <ws-record-descriptions> ::=
{<ws-record-description>
[<redefining-ws-record-description>]...}...

REPORT SECTION

D11 <report-section> ::=
←REPORT # SECTION <.>
<report-section-body>

D12 <report-section-body> ::=
<report-specification>...

D13 <report-specification> ::=
<report-description>
<report-group-description>...

FD SKELETON

```
D14   <non-ms-file-description> ::=
          ←FD # ↑
          <non-ms-file-name-declaration>
          {<;> <copy-clause> !
          <fd-clauses>} <.>

D15   <ms-file-description> ::=
          ←FD # ↑
          {<sequential-ms-file-name-declaration> !
          <random-ms-file-name-declaration>}
          {<;> <copy-clause> !
          <fd-clauses>} <.>

D16   <fd-clauses> ::=
          {[<;> <block-clause>] ↓
          [<;> <record-contains-clause>] ↓
          <;> <label-records-clause> ↓
          [<;> <value-of-clause>] ↓
          {[<;> <data-records-clause>] !
          <;> <report-clause>}▷
```

D17 <non-ms-file-name-declaration> ::=
    <non-reserved-alpha-word>

D18 <non-ms-file-name> ::=
    <non-reserved-alpha-word>
    which appears as a
    <non-ms-file-name-declaration>

D19 <sequential-ms-file-name-declaration> ::=
    <non-reserved-alpha-word>

D20 <sequential-ms-file-name> ::=
    <non-reserved-alpha-word>
    which appears as a
    <sequential-ms-file-name-declaration>

D21 <random-ms-file-name-declaration> ::=
    <non-reserved-alpha-word>

D22 <random-ms-file-name> ::=
    <non-reserved-alpha-word>
    which appears as a
    <random-ms-file-name-declaration>

D23 <sequential-file-name> ::=
    <non-ms-file-name> !
    <sequential-ms-file-name>

D24 <ms-file-name> ::=
    <sequential-ms-file-name> !
    <random-ms-file-name>

D25 <non-sort-file-name> ::=
    <non-ms-file-name> !
    <ms-file-name>

D26 <file-name> ::=
    <non-sort-file-name> !
    <sort-file-name>

SD SKELETON

D27 <sort-file-description> ::=
←SD # ↑
<sort-file-name-declaration>
{<;> <copy-clause> !
<sd-clauses>} <.>

D28 <sd-clauses> ::=
{[<;> <record-contains-clause>] ↓
<;> <data-records-clause>}

D29 <sort-file-name-declaration> ::=
<non-reserved-alpha-word>

D30 <sort-file-name> ::=
<non-reserved-alpha-word>
which appears as a
<sort-file-name-declaration>

RD SKELETON

D31 <report-description> ::=
←RD # ↑
<report-name-declaration>
{<;> <copy-clause> !
<rd-clauses>} <.>

D32 <rd-clauses> ::=
{[<;> <code-clause>] ↓
[<;> <control-clause>] ↓
[<;> <page-limit-clause>]}

D33 <report-name-declaration> ::=
<non-reserved-alpha-word>

D34 <report-name> ::=
<non-reserved-alpha-word>
which appears as a
<report-name-declaration>

FILE AND SORT FILE RECORD DESCRIPTION SKELETON

```
D35   <non-ms-record-description> ::=
          ← {Ø ! 0} 1 # ↑
          <non-elem-non-ms-record-name-declaration>
          {<;> <copy-clause> <.> !
          <non-elem-record-spec>} !
          ← {Ø ! 0} 1 # ↑
          <elem-non-ms-record-name-declaration>
          {<;> <copy-clause> <.> !
          <elem-record-spec>}

D36   <ms-record-description> ::=
          ← {Ø ! 0} 1 # ↑
          <non-elem-ms-record-name-declaration>
          {<;> <copy-clause> <.> !
          <non-elem-record-spec>} !
          ← {Ø ! 0} 1 # ↑
          <elem-ms-record-name-declaration>
          {<;> <copy-clause> <.> !
          <elem-record-spec>}

D37   <sort-record-description> ::=
          ← {Ø ! 0} 1 # ↑
          <non-elem-sort-record-name-declaration>
          {<;> <copy-clause> <.> !
          <non-elem-record-spec>} !
          ← {Ø ! 0} 1 # ↑
          <elem-sort-record-name-declaration>
          {<;> <copy-clause> <.> !
          <elem-record-spec>}
```

```
D38  <non-elem-record-spec> ::=
         <non-elem-01-clauses> <.>
         [<88-entry>]...
         <subordinate-entries>
         [<66-entry>]... !
         <;> <usage-is-index-clause> <.>
         <subordinate-entries>
         [<66-entry>]...

D39  <elem-record-spec> ::=
         <elem-01-77-clauses> <.>
         [<88-entry>]... !
         <;> <usage-is-index-clause> <.>
```

D40 <66-entry> ::=
←66 # ↑
<non-elem-66-item-name-declaration>
<;> <non-elem-renames-clause> !
←66 # ↑
<elem-66-item-name-declaration>
<;> <elem-renames-clause>

D41 <88-entry> ::=
←88 # ↑
<condition-name-declaration>
<;> <88-value-clause> <.>

D42 ⟨subordinate-entries⟩ ::=
← # ⟨sub-number⟩ # ↑
⟨sub-spec⟩
[← # ⟨sub-number⟩ # ↑
{⟨sub-spec⟩ !
⟨redefining-sub-spec⟩}]...

D43 ⟨sub-spec⟩ ::=
⟨non-elem-spec⟩
⟨subordinate-entries⟩ !
⟨index-non-elem-spec⟩
⟨subordinate-entries⟩ !
⟨elem-spec⟩ !
⟨index-elem-spec⟩

D44 ⟨redefining-sub-spec⟩ ::=
⟨redefining-non-elem-spec⟩
⟨subordinate-entries⟩ !
⟨redefining-index-non-elem-spec⟩
⟨subordinate-entries⟩ !
⟨redefining-elem-spec⟩ !
⟨redefining-index-elem-spec⟩

D45 ⟨sub-number⟩ ::=
⟨level-number⟩
with a value increased with respect to the entry to which the ⟨subordinate-entries⟩ are subordinate

D46 ⟨level-number⟩ ::=
{1 ! 2 ! 3 ! 4} ⟨digit⟩ !
{Ø ! 0} ⟨non-zero-digit⟩

D47 <non-elem-spec> ::=
    <non-elem-02-48-item-name-declaration>
    <non-elem-clauses> <.>
    [<88-entry>]...

D48 <redefining-non-elem-spec> ::=
    <non-elem-02-48-item-name-declaration>
    <;> <redefines-clause>
    <non-elem-red-clauses> <.>
    [<88-entry>]...

D49 <elem-spec> ::=
    <02-49-name-declaration>
    <elem-clauses> <.>
    [<88-entry>]...

D50 <redefining-elem-spec> ::=
    <02-49-name-declaration>
    <;> <redefines-clause>
    <elem-red-clauses> <.>
    [<88-entry>]...

D51 <index-non-elem-spec> ::=
    <non-elem-02-48-item-name-declaration>
    [<;> <usage-is-index-clause>] <.>

D52 <redefining-index-non-elem-spec> ::=
    <non-elem-02-48-item-name-declaration>
    <;> <redefines-clause>
    [<;> <usage-is-index-clause>] <.>

D53 <index-elem-spec> ::=
    <02-49-name-declaration>
    [<;> <usage-is-index-clause>] <.>

D54 <redefining-index-elem-spec> ::=
    <02-49-name-declaration>
    <;> <redefines-clause>
    [<;> <usage-is-index-clause>] <.>

D55 <non-elem-clauses> ::=
    <non-elem-01-clauses> !
    <non-elem-red-clauses> !
    <var-occurs-non-elem-clauses>

D56 <non-elem-01-clauses> ::=
    {[<;> <usage-clause>] ↓
    [<;> <value-clause>]}

D57 <non-elem-red-clauses> ::=
    {[<;> <usage-clause>] ↓
    [<;> <fixed-occurs-clause>]}

D58 <var-occurs-non-elem-clauses> ::=
    {[<;> <usage-clause>] ↓
    <;> <variable-occurs-clause>}

D59 <elem-clauses> ::=
    <elem-01-77-clauses> !
    <elem-red-clauses> !
    <var-occurs-elem-clauses>

D60 <elem-01-77-clauses> ::=
    {[<;> <usage-clause>] ↓
    <;> <picture-clause> ↓
    [<;> <justified-clause>] ↓
    [<;> <blank-when-zero-clause>] ↓
    [<;> <synchronized-clause>] ↓
    [<;> <value-clause>]}

D61 <elem-red-clauses> ::=
    {[<;> <usage-clause>] ↓
    <;> <picture-clause> ↓
    [<;> <justified-clause>] ↓
    [<;> <blank-when-zero-clause>] ↓
    [<;> <synchronized-clause>] ↓
    [<;> <fixed-occurs-clause>]}

D62 <var-occurs-elem-clauses> ::=
    {[<;> <usage-clause>] ↓
    <;> <picture-clause> ↓
    [<;> <justified-clause>] ↓
    [<;> <blank-when-zero-clause>] ↓
    [<;> <synchronized-clause>] ↓
    <;> <variable-occurs-clause>}

D63 <non-elem-non-ms-record-name-declaration> ::=
    <non-reserved-alpha-word>

D64 <non-elem-non-ms-record-name> ::=
    <non-reserved-alpha-word>
    which appears as a
    <non-elem-non-ms-record-name-declaration>

D65 <elem-non-ms-record-name-declaration> ::=
    <non-reserved-alpha-word>

D66 <elem-non-ms-record-name> ::=
    <non-reserved-alpha-word>
    which appears as a
    <elem-non-ms-record-name-declaration>

D67 <non-elem-ms-record-name-declaration> ::=
    <non-reserved-alpha-word>

D68 <non-elem-ms-record-name> ::=
    <non-reserved-alpha-word>
    which appears as a
    <non-elem-ms-record-name-declaration>

D69 <elem-ms-record-name-declaration> ::=
    <non-reserved-alpha-word>

D70 <elem-ms-record-name> ::=
    <non-reserved-alpha-word>
    which appears as a
    <elem-ms-record-name-declaration>

D71 <non-elem-sort-record-name-declaration> ::=
    <non-reserved-alpha-word>

D72 <non-elem-sort-record-name> ::=
    <non-reserved-alpha-word>
    which appears as a
    <non-elem-sort-record-name-declaration>

D73 <elem-sort-record-name-declaration> ::=
    <non-reserved-alpha-word>

D74 <elem-sort-record-name> ::=
    <non-reserved-alpha-word>
    which appears as a
    <elem-sort-record-name-declaration>

D75 <non-elem-02-48-item-name-declaration> ::=
    <non-reserved-alpha-word>

D76 <non-elem-02-48-item-name> ::=
<non-reserved-alpha-word>
which appears as a
<non-elem-02-48-item-name-declaration>

D77 <02-49-name-declaration> ::=
<elem-02-49-item-name-declaration> !
FILLER

D78 <elem-02-49-item-name-declaration> ::=
<non-reserved-alpha-word>

D79 <elem-02-49-item-name> ::=
<non-reserved-alpha-word>
which appears as a
<elem-02-49-item-name-declaration>

D80 <non-elem-66-item-name-declaration> ::=
<non-reserved-alpha-word>

D81 <non-elem-66-item-name> ::=
<non-reserved-alpha-word>
which appears as a
<non-elem-66-item-name-declaration>

D82 <elem-66-item-name-declaration> ::=
<non-reserved-alpha-word>

D83 <elem-66-item-name> ::=
<non-reserved-alpha-word>
which appears as a
<elem-66-item-name-declaration>

D84 <condition-name-declaration> ::=
<non-reserved-alpha-word>

D85 <condition-name> ::=
<non-reserved-alpha-word>
which appears as a
<condition-name-declaration>

WORKING-STORAGE DATA DESCRIPTION SKELETON

```
D86   <77-description> ::=
          ←77 # ↑
          <77-item-name-declaration>
          <elem-01-77-clauses> <.>
          [<88-entry>]... !
          ←77 # ↑
          <77-item-name-declaration>
          <;> <usage-is-index-clause> <.>

D87   <redefining-77-description> ::=
          ←77 # ↑
          <77-item-name-declaration>
          <;> <77-redefines-clause>
          <elem-01-77-red-clauses> <.>
          [<88-entry>]... !
          ←77 # ↑
          <77-item-name-declaration>
          <;> <77-redefines-clause>
          <;> <usage-is-index-clause> <.>

D88   <ws-record-description> ::=
          ← {Ø ! 0} 1 # ↑
          <non-elem-ws-record-name-declaration>
          {<;> <copy-clause> <.> !
          <non-elem-record-spec>} !
          ← {Ø ! 0} 1 # ↑
          <elem-ws-record-name-declaration>
          {<;> <copy-clause> <.> !
          <elem-record-spec>}

D89   <redefining-ws-record-description> ::=
          ← {Ø ! 0} 1 # ↑
          <non-elem-ws-record-name-declaration>
          {<;> <copy-clause> <.> !
          <redefining-non-elem-record-spec>} !
          ← {Ø ! 0} 1 # ↑
          <elem-ws-record-name-declaration>
          {<;> <copy-clause> <.> !
          <redefining-elem-record-spec>}
```

D90 <redefining-non-elem-record-spec> ::=
<;> <redefines-record-clause>
[<;> <usage-clause>] <.>
[<88-entry>]...
<subordinate-entries>
[<66-entry>]... !
<;> <redefines-record-clause>
<;> <usage-is-index-clause> <.>
<subordinate-entries>
[<6-entry>]...

D91 <redefining-elem-record-spec> ::=
<;> <redefines-record-clause>
<elem-01-77-red-clauses> <.>
[<88-entry>]... !
<;> <redefines-record-clause>
<;> <usage-is-index-clause> <.>

D92 <elem-01-77-red-clauses> ::=
{[<;> <usage-clause>] ↓
<;> <picture-clause> ↓
[<;> <justified-clause>] ↓
[<;> <blank-when-zero-clause>] ↓
[<;> <synchronized-clause>]}

D93 <77-item-name-declaration> ::=
<non-reserved-alpha-word>

D94 <77-item-name> ::=
<non-reserved-alpha-word>
which appears as a
<77-item-name-declaration>

D95 <non-elem-ws-record-name-declaration> ::=
<non-reserved-alpha-word>

D96 <non-elem-ws-record-name> ::=
<non-reserved-alpha-word>
which appears as a
<non-elem-ws-record-name-declaration>

D97 <elem-ws-record-name-declaration> ::=
<non-reserved-alpha-word>

D98 <elem-ws-record-name> ::=
<non-reserved-alpha-word>
which appears as a
<elem-ws-record-name-declaration>

REPORT-GROUP DESCRIPTION SKELETON

```
D99    <report-group-description> ::=
          ← {Ø ! 0} 1 # ↑
          [<non-elem-report-group-name-declaration>]
          {<;> <copy-clause> <.> !
          <non-elem-report-group-spec>} !
          ← {Ø ! 0} 1 # ↑
          [<sum-report-group-name-declaration>]
          {<;> <copy_clause> <.> !
          <sum-report-group-spec>} !
          ← {Ø ! 0} 1 # ↑
          [<elem-non-sum-report-group-name-declaration>]
          {<;> <copy_clause> <.> !
          <elem-non-sum-report-group-spec>}
```

D100 <non-elem-report-group-spec> ::=
<non-elem-group-clauses> <.>
<subordinate-report-entries>

D101 <sum-report-group-spec> ::=
<sum-group-clauses> <.>

D102 <elem-non-sum-report-group-spec> ::=
<elem-non-sum-group-clauses> <.>

D103 <subordinate-report-entries> ::=
{<sub-number> # ↑
<sub-report-spec>}...

D104 <sub-report-spec> ::=
<non-elem-report-spec>
<subordinate-report-entries> !
<elem-report-spec>

D105 <non-elem-report-spec> ::=
[<non-elem-field-name-declaration>]
{[<;> <line-number-clause>] ↓
[<;> <usage-is-display-clause>]} <.>

D106 <elem-report-spec> ::=
<sum-spec> !
<elem-non-sum-spec>

D107 <sum-spec> ::=
[<sum-field-name-declaration>]
{[<;> <line-number-clause>] ↓
<;> <column-number-clause> ↓
[<;> <usage-is-display-clause>] ↓
<;> <picture-clause> ↓
[<;> <justified-clause>] ↓
[<;> <blank-when-zero-clause>] ↓
<;> <sum-clause> ↓
[<;> <reset-clause>]} <.> !
<sum-field-name-declaration>
{[<;> <line-number-clause>] ↓
[<;> <usage-is-display-clause>] ↓
<;> <picture-clause> ↓
<;> <sum-clause> ↓
[<;> <reset-clause>]} <.>

D108 <elem-non-sum-spec> ::=
[<elem-non-sum-field-name-declaration>]
{[<;> <line-number-clause>] ↓
<;> <column-number-clause> ↓
[<;> <group-indicate-clause>] ↓
[<;> <usage-is-display-clause>] ↓
<;> <picture-clause> ↓
[<;> <blank-when-zero-clause>] ↓
{<;> <source-clause> !
<;> <value-clause>}} <.> !
[<elem-non-sum-field-name-declaration>]
<;> <source-clause> <.>

```
D109 <non-elem-group-clauses> ::=
          {<;> <type-clause> +
          [<;> <next-group-clause>] +
          [<;> <line-number-clause>] +
          [<;> <usage-is-display-clause>]}

D110 <sum-group-clauses> ::=
          {<;> <type-clause> +
          [<;> <next-group-clause>] +
          <;> <line-number-clause> +
          <;> <column-number-clause> +
          [<;> <usage-is-display-clause>] +
          <;> <picture-clause> +
          [<;> <justified-clause>] +
          [<;> <blank-when-zero-clause>] +
          <;> <sum-clause> +
          [<;> <reset-clause>]} !
          {<;> <type-clause> +
          [<;> <next-group-clause>] +
          <;> <line-number-clause> +
          [<;> <usage-is-display-clause>] +
          <;> <picture-clause> +
          <;> <sum-clause> +
          [<;> <reset-clause>]}

D111 <elem-non-sum-group-clauses> ::=
          {<;> <type-clause> +
          [<;> <next-group-clause>] +
          <;> <line-number-clause> +
          <;> <column-number-clause> +
          [<;> <group-indicate-clause>] +
          [<;> <usage-is-display-clause>] +
          <;> <picture-clause> +
          [<;> <justified-clause>] +
          [<;> <blank-when-zero-clause>] +
          {<;> <source-clause> !
          <;> <value-clause>+} !
          {<;> <type-clause> +
          [<;> <next-group-clause>] +
          <;> <line-number-clause> +
          [<;> <source-clause>]}
```

D112 <non-elem-report-group-name-declaration> ::=
<non-reserved-alpha-word>

D113 <non-elem-report-group-name> ::=
<non-reserved-alpha-word>
which appears as a
<non-elem-report-group-name-declaration>

D114 <elem-non-sum-report-group-name-declaration> ::=
<non-reserved-alpha-word>

D115 <elem-non-sum-report-group-name> ::=
<non-reserved-alpha-word>
which appears as a
<elem-non-sum-report-group-name-declaration>

D116 <sum-report-group-name-declaration> ::=
<non-reserved-alpha-word>

D117 <sum-report-group-name> ::=
<non-reserved-alpha-word>
which appears as a
<sum-report-group-name-declaration>

D118 <non-elem-field-name-declaration> ::=
<non-reserved-alpha-word>

D119 <non-elem-field-name> ::=
<non-reserved-alpha-word>
which appears as a
<non-elem-field-name-declaration>

D120 <elem-non-sum-field-name-declaration> ::=
<non-reserved-alpha-word>

D121 <sum-field-name-declaration> ::=
<non-reserved-alpha-word>

D122 <sum-field-name> ::=
<non-reserved-alpha-word>
which appears as a
<sum-field-name-declaration>

BLANK WHEN ZERO CLAUSE

D123 <blank-when-zero-clause> ::=
BLANK [# WHEN] # ZERO

BLOCK CLAUSE

D124 <block-clause> ::=
BLOCK # [CONTAINS #]
[<positive-integer> # TO #]
<positive-integer>
{[# CHARACTERS] ! # RECORDS}

CODE CLAUSE

D125 <code-clause> ::=
CODE #
<mnemonic-name-for-code-for-report-groups>

COLUMN NUMBER CLAUSE

D126 <column-number-clause> ::=
COLUMN # [NUMBER #] [IS #]
<positive-integer>

CONTROL CLAUSE

D127 <control-clause> ::=
{CONTROL # [IS #] !
CONTROLS # [ARE #]}
{FINAL !
<item-name-identifier>}
[<,> <item-name-identifier>]...

DATA RECORDS CLAUSE

D128 <data-records-clause> ::=
DATA #
{RECORD # [IS #] !
RECORDS # [ARE #]}
<non-ws-record-name>
[<,> <non-ws-record-name>]...

GROUP INDICATE CLAUSE

D129 <group-indicate-clause> ::=
GROUP [# INDICATE]

JUSTIFIED CLAUSE

D130 <justified-clause> ::=
{JUSTIFIED ! JUST} [# RIGHT]

LABEL RECORDS CLAUSE

D131 <label-records-clause> ::=
LABEL #
{RECORD # [IS #] !
RECORDS # [ARE #]}
{OMITTED ! STANDARD !
<non-ws-record-name>
[<,> <non-ws-record-name>]...}

LINE NUMBER CLAUSE

D132 <line-number-clause> ::=
LINE # [NUMBER #] [IS #]
{<positive-integer> !
PLUS # <positive-integer> !
NEXT # PAGE}

NEXT GROUP CLAUSE

D133 <next-group-clause> ::=
NEXT # GROUP # [IS #]
{<positive-integer> !
PLUS # <positive-integer> !
NEXT # PAGE}

OCCURS CLAUSE

D134 <fixed-occurs-clause> ::=
OCCURS #
<positive-integer> [# TIMES]
[# <key-option>]...
[# <index-option>]

D135 <variable-occurs-clause> ::=
OCCURS #
<integer> # TO #
<positive-integer> [# TIMES]
[# DEPENDING # [ON #]
<elem-item-name-qualified>]
[# <key-option>]...
[# <index-option>]

D136 <key-option> ::=
{ASCENDING ! DESCENDING} #
[KEY #] [IS #]
<elem-item-name-qualified>
[<,> <elem-item-name-qualified>]...

D137 <index-option> ::=
INDEXED # [BY #]
<index-name-declaration>
[<,> <index-name-declaration>]...

D138 <index-name-declaration> ::=
<non-reserved-alpha-word>

D139 <index-name> ::=
<non-reserved-alpha-word>
which appears as a
<index-name-declaration>

PAGE LIMIT CLAUSE

D140 <page-limit-clause> ::=
PAGE #
{[LIMIT #] [IS #] !
[LIMITS #] [ARE #]}
<positive-integer> #
{LINE ! LINES}
[<,> HEADING #
<positive-integer>]
[<,> FIRST # DETAIL #
<positive-integer>]
[<,> LAST # DETAIL #
<positive-integer>]
[<,> FOOTING #
<positive-integer>]

PICTURE CLAUSE

D141 <picture-clause> ::=
{PICTURE ! PIC} # [IS #]
<picture>

D142 <picture> ::=
<numeric-picture> !
<numeric-edited-picture>
not ending with a period or a comma !
<nonnumeric-picture>

D143 <numeric-picture> ::=
[S] <dig-seq>
[V [<dig-seq>]] !
[S] <dig-seq>
[<p>]... [V] !
[S] [V] [<p>]...
<dig-seq>

D144 <numeric-edited-picture> ::=
<sign-float-pict> !
<cs-float-pict> !
<supp-pict> !
<fixed-insert-pict>

D145 <nonnumeric-picture> ::=
[<b09>]...
{<ax>... [<b09>]...}...

D146 <sign-float-pict> ::=
[<cs>] <+-seq>
<point> <+-seq> !
[<cs>] <--seq>
<point> <--seq> !
[<cs>] <sign-float>
[<9-seq>] <point>
[<9-seq-or-b0,>] !
[<cs>] <sign-float>
[<9-seq>] [<p>]... [V] !
[V] [<p>]...
[<cs>] <sign-float>

D147 <cs-float-pict> ::=
[+ ! -] <cs-seq>
<point> <cs-seq> !
[+ ! -] <cs-float>
[<9-seq>] <point>
[<9-seq-or-b0,>] !
[+ ! -] <cs-float>
[<9-seq>] [<p>]... [V] !
[V] [<p>]...
[+ ! -] <cs-float> !
<cs-seq> <point>
<cs-seq> {+ ! - ! CR ! DB} !
<cs-float> [<9-seq>]
<point> [<9-seq-or-b0,>]
{+ ! - ! CR ! DB} !
<cs-float> [<9-seq>]
{+ ! -} [<p>]... [V] !
<cs-float> [<9-seq>]
{CR ! DB} [<p>]... !
[V] [<p>]...
<cs-float> {+ ! - ! CR ! DB}

D148 <supp-pict> ::=
[+ ! -] [<cs>] <z-seq>
<point> <z-seq> !
[+ ! -] [<cs>] <*-seq>
<point> <*-seq> !
[+ ! -] [<cs>] <z-or-*-seq>
[<9-seq>] <point>
[<9-seq-or-b0,>] !
[+ ! -] [<cs>] <z-or-*-seq>
[<9-seq>] [<p>]... [V] !
[V] [<p>]...
[+ ! -] [<cs>] <z-or-*-seq> !
[<cs>] <z-seq> <point>
<z-seq> {+ ! - ! CR ! DB} !
[<cs>] <*-seq> <point>
<*-seq> {+ ! - ! CR ! DB} !
[<cs>] <z-or-*-seq> [<9-seq>]
<point> [<9-seq-or-b0,>]
{+ ! - ! CR ! DB} !

```
      continued....
         [<cs>] <z-or-*-seq> [<9-seq>]
         {+ ! -} [<p>]... [V] !
         [<cs>] <z-or-*-seq> [<9-seq>]
         {CR ! DB} [<p>]... !
         [V] [<p>]... [<cs>]
         <z-or-*-seq> {+ ! - ! CR ! DB}

D149 <fixed-insert-pict> ::=
         [+ ! -] [<cs>] <9-seq>
         <point> [<9-seq-or-b0,>] !
         [+ ! -] [<cs>] <9-seq>
         [<p>]... [V] !
         [V] [<p>]...
         [+ ! -] [<cs>] <9-seq> !
         [<cs>] <9-seq>
         <point> [<9-seq-or-b0,>]
         {+ ! - ! CR ! DB} !
         [<cs>] <9-seq>
         {+ ! -} [<p>]... [V] !
         [<cs>] <9-seq>
         {CR ! DB} [<p>]... !
         [V] [<p>]... [<cs>]
         <9-seq> {+ ! - ! CR ! DB}

D150 <sign-float> ::=
         [<b0,>]... + <+-seq> !
         [<b0,>]... - <--seq> !
         [<b0,>]...
         + (<positive-integer>)
         [<+-seq-or-b0,>] !
         [<b0,>]...
         - (<positive-integer>)
         [<--seq-or-b0,>]

D151 <cs-float> ::=
         [<b0,>]... <cs> <cs-seq> !
         [<b0,>]...
         <cs> (<positive-integer>)
         [<cs-seq-or-b0,>]
```

D152 <z-or-*-seq> ::=
<z-seq> !
<*-seq>

D153 <9-seq-or-b0,> ::=
<9-seq> !
<b0,>...

D154 <+-seq-or-b0,> ::=
<+-seq> !
<b0,>...

D155 <--seq-or-b0,> ::=
<--seq> !
<b0,>...

D156 <cs-seq-or-b0,> ::=
<cs-seq> !
<b0,>...

D157 <dig-seq> ::=
[<0>]...
{<9>... [<0>]...}...

D158 <9-seq> ::=
[<b0,>]...
{<9>... [<b0,>]...}...

D159 <+-seq> ::=
[<b0,>]...
{<+>... [<b0,>]...}...

D160 <--seq> ::=
[<b0,>]...
{<->... [<b0,>]...}...

D161 <cs-seq> ::=
[<b0,>]...
{<ß>... [<b0,>]...}...

D162 <z-seq> ::=
[<b0,>]...
{<z>... [,b0,>]...}...

D163 <*-seq> ::=
[<b0,>]...
{<*>... [<b0,>]...}...

D164 <b0,> ::=
{B ! O ! <digit-separator>}
[(<positive-integer>)]

D165 <b09> ::=
{B ! 0 ! 9}
[(<positive-integer>)]

D166 <ax> ::=
{A ! X}
[(<positive-integer>)]

D167 <9> ::=
9
[(<positive-integer>)]

D168 <0> ::=
0
[(<positive-integer>)]

D169 <p> ::=
P
[(<positive-integer>)]

D170 <+> ::=
+
[(<positive-integer>)]

D171 <-> ::=
-
[(<positive-integer>)]

D172 <$> ::=
<cs>
[(<positive-integer>)]

D173 <z> ::=
Z
[(<positive-integer>)]

D174 <*> ::=
*
[(<positive-integer>)]

D175 <point> ::=
V !
<decimal-point>

RECORD CONTAINS CLAUSE

D176 <record-contains-clause> ::=
RECORD # [CONTAINS #]
[<positive-integer>] # [TO #]
<positive-integer>
[# CHARACTERS]

REDEFINES CLAUSE

D177 <redefines-clause> ::=
REDEFINES #
<02-49-item-name-qualified>

D178 <redefines-record-clause> ::=
REDEFINES #
<ws-record-name>

D179 <77-redefines-clause> ::=
REDEFINES #
<77-item-name>

RENAMES CLAUSE

D180 <elem-renames-clause> ::=
RENAMES #
<elem-item-name-qualified>

D181 <non-elem-renames-clause> ::=
RENAMES #
<item-name-qualified>
[# {THROUGH ! THRU} #
<item-name-qualified>]

REPORT CLAUSE

D182 <report-clause> ::=
{REPORT # [IS #] !
REPORTS # [ARE #]}
<report-name>
[<;> <report-name>]...

RESET CLAUSE

D183 <reset-clause> ::=
RESET # [ON #]
{FINAL !
<item-name-identifier>}
[<,> <item-name-identifier>]...

SOURCE CLAUSE

D184 <source-clause> ::=
SOURCE # [IS #]
<source-item-name-identifier>

SUM CLAUSE

D185 <sum-clause> ::=
SUM #
<summed-item-name-identifier>
[<,> <summed-item-name-identifier>]...
[# UPON #
<report-group-name-qualified>]

SYNCHRONIZED CLAUSE

D186 <synchronized-clause> ::=
{SYNCHRONIZED ! SYNC}
[# {LEFT ! RIGHT}]

TYPE CLAUSE

D187 <type-clause> ::=
TYPE # [IS #]
{REPORT # HEADING !
RH !
PAGE # HEADING !
PH !
CONTROL # HEADING #
{FINAL !
<item-name-identifier>} !
CH #
{FINAL !
<item-name-identifier>} !
DETAIL !
DE !
CONTROL # FOOTING #
{FINAL !
<item-name-identifier>} !
CF #
{FINAL !
<item-name-identifier>} !
PAGE # FOOTING !
PF !
REPORT # FOOTING !
RF}

USAGE CLAUSE

D188 <usage-clause> ::=
[USAGE # [IS #]]
{COMPUTATIONAL !
COMP !
DISPLAY}

D189 <usage-is-display-clause> ::=
[USAGE # [IS #]] DISPLAY

D190 <usage-is-index-clause> ::=
[USAGE # [IS #]] INDEX

VALUE CLAUSE

D191 <value-clause> ::=
VALUE # [IS #] <literal>

D192 <88-value-clause> ::=
{VALUE # [IS #] !
VALUES # [ARE #]}
{<numeric-literal>
[# {THROUGH ! THRU} #
<numeric-literal>]
[<,> <numeric-literal>
[# {THROUGH ! THRU} #
<numeric-literal>]]... !
<nonnumeric-literal>
[# {THROUGH ! THRU} #
<nonnumeric-literal>]
[<,> <nonnumeric-literal>
[# {THROUGH ! THRU} #
<nonnumeric-literal>]]...}

VALUE OF CLAUSE

D193 <value-of-clause> ::=
VALUE # OF #
<elem-item-name-qualified> #
[IS #]
{<elem-item-name-qualified> !
<literal>}
[<,> <elem-item-name-qualified>
[IS #]
{<elem-item-name-qualified> !
<literal>}]...

IDENTIFIERS

D194 <non-ws-record-name> ::=
<non-elem-non-ms-record-name> !
<non-elem-ms-record-name> !
<non-elem-sort-record-name> !
<elem-non-ms-record-name> !
<elem-ms-record-name> !
<elem-sort-record-name>

D195 <ws-record-name> ::=
<non-elem-ws-record-name> !
<elem-ws-record-name>

```
D196 <in-of> ::=
         # IN # ! # OF #

D197 <non-elem-non-ms-record-name-qualified> ::=
         <non-elem-non-ms-record-name>
         [<in-of>
         <non-ms-file-name>]

D198 <elem-non-ms-record-name-qualified> ::=
         <elem-non-ms-record-name>
         [<in-of>
         <non-ms-file-name>]

D199 <non-elem-ms-record-name-qualified> ::=
         <non-elem-ms-record-name>
         [<in-of>
         <ms-file-name>]

D200 <elem-ms-record-name-qualified> ::=
         <elem-ms-record-name>
         [<in-of>
         <ms-file-name>]

D201 <non-elem-sort-record-name-qualified> ::=
         <non-elem-sort-record-name>
         [<in-of>
         <sort-file-name>]

D202 <elem-sort-record-name-qualified> ::=
         <elem-sort-record-name>
         [<in-of>
         <sort-file-name>]
```

```
D203 <sum-report-group-name-qualified> ::=
        <sum-report-group-name>
        [<in-of>
        <report-name>]

D204 <non-ms-record-name-qualified> ::=
        <non-elem-non-ms-record-name-qualified> !
        <elem-non-ms-record-name-qualified>

D205 <ms-record-name-qualified> ::=
        <non-elem-ms-record-name-qualified> !
        <elem-ms-record-name-qualified>

D206 <sort-record-name-qualified> ::=
        <non-elem-sort-record-name-qualified> !
        <elem-sort-record-name-qualified>

D207 <non-elem-record-name-qualified> ::=
        <non-elem-non-ms-record-name-qualified> !
        <non-elem-ms-record-name-qualified> !
        <non-elem-sort-record-name-qualified> !
        <non-elem-ws-record-name>

D208 <elem-record-name-qualified> ::=
        <elem-non-ms-record-name-qualified> !
        <elem-ms-record-name-qualified> !
        <elem-sort-record-name-qualified> !
        <elem-ws-record-name>

D209 <non-elem-report-group-name-qualified> ::=
        <non-elem-report-group-name>
        [<in-of>
        <report-name>]

D210 <elem-non-sum-report-group-name-qualified> ::=
        <elem-non-sum-report-group-name>
        [<in-of>
        <report-name>]

D211 <report-group-name-qualified> ::=
        <non-elem-report-group-name-qualified> !
        <elem-non-sum-report-group-name-qualified> !
        <sum-report-group-name-qualified>
```

```
D212 <non-elem-02-48-item-name-qualified> ::=
          <non-elem-02-48-item-name>
          [<in-of>
          <non-elem-02-48-item-name>]...
          [<in-of>
          <non-elem-record-name-qualified>]

D213 <elem-02-49-item-name-qualified> ::=
          <elem-02-49-item-name>
          [<in-of>
          <non-elem-02-48-item-name-qualified>]

D214 <02-49-item-name-qualified> ::=
          <non-elem-02-48-item-name-qualified> !
          <elem-02-49-item-name-qualified>

D215 <non-elem-66-item-name-qualified> ::=
          <non-elem-66-item-name>
          [<in-of>
          <non-elem-record-name-qualified>]

D216 <elem-66-item-name-qualified> ::=
          <elem-66-item-name>
          [<in-of>
          <non-elem-record-name-qualified>]
```

```
D217 <sum-field-name-qualified> ::=
         <sum-field-name>
         [<in-of>
         <non-elem-field-name>]...
         [<non-elem-report-group-name-qualified>]

D218 <non-elem-item-name-qualified> ::=
         <non-elem-record-name-qualified> !
         <non-elem-02-48-item-name-qualified> !
         <non-elem-66-item-name-qualified>

D219 <elem-item-name-qualified> ::=
         <elem-record-name-qualified> !
         <elem-02-49-item-name-qualified> !
         <elem-66-item-name-qualified> !
         <77-item-name> !
         <special-item-name-qualified>

D220 <item-name-qualified> ::=
         <non-elem-item-name-qualified> !
         <elem-item-name-qualified>

D221 <sum-item-name-qualified> ::=
         <sum-report-group-name-qualified> !
         <sum-field-name-qualified>

D222 <special-item-name-qualified> ::=
         TALLY !
         LINE-COUNTER
         [<in-of> <report-name>] !
         PAGE-COUNTER
         [<in-of> <report-name>]

D223 <condition-name-qualified> ::=
         <condition-name>
         [<in-of>
         <02-49-item-name-qualified>]

D224 <index-name-qualified> ::=
         <index-name>
         [<in-of>
         <02-49-item-name-qualified>]
```

D225 <non-elem-02-48-item-name-identifier> ::=
<non-elem-02-48-item-name-qualified>
[(<subscripts>)]

D226 <elem-02-49-item-name-identifier> ::=
<elem-02-49-item-name-qualified>
[(<subscripts>)]

D227 <non-elem-item-name-identifier> ::=
<non-elem-record-name-qualified> !
<non-elem-02-48-item-name-identifier> !
<non-elem-66-item-name-qualified>

D228 <elem-item-name-identifier> ::=
<elem-record-name-qualified> !
<elem-02-49-item-name-identifier> !
<elem-66-item-name-qualified> !
<77-item-name> !
<special-item-name-qualified>

D229 <item-name-identifier> ::=
<non-elem-item-name-identifier> !
<elem-item-name-identifier>

D230 <source-item-name-identifier> ::=
<item-name-identifier> !
<sum-item-name-qualified>

D231 <summed-item-name-identifier> ::=
<elem-item-name-identifier> !
<sum-item-name-qualified>

D232 <non-elem-data-name-identifier> ::=
<non-elem-item-name-identifier>

D233 <elem-data-name-identifier> ::=
<elem-item-name-identifier> !
<sum-item-name-qualified>

D234 <data-name-identifier> ::=
<non-elem-data-name-identifier> !
<elem-data-name-identifier>

D235 <condition-name-identifier> ::=
<condition-name-qualified>
[(<subscripts>)]

D236 <subscripts> ::=
<elem-item-name-qualified>
[<,>
<elem-item-name-qualified>
]... !
<relative-index-name-qualified>
[<,>
<relative-index-name-qualified>
]...

D237 <relative-index-name-qualified> ::=
<index-name-qualified>
[# {+ ! -} #
<integer>]

P. PROCEDURE DIVISION

PROCEDURE DIVISION STRUCTURE

P1 <procedure-division> ::=
←PROCEDURE # DIVISION <.>
<procedure-division-body>

P2 <procedure-division-body> ::=
[<declarative-portion>]
<non-declarative-portion> !
<paragraph>...

DECLARATIVE PORTION

P3 <declarative-portion> ::=
←DECLARATIVES <.>
<declarative-section>...
←END # DECLARATIVES <.>

NON-DECLARATIVE PORTION

P4 <non-declarative-portion> ::=
<non-declarative-section>...

SECTIONS

P5 <declarative-section> ::=
← <section-name-declaration> # SECTION <.>
{<copy-sentence> !
<declarative-sentence> <section-body>}

P6 <non-declarative-section> ::=
← <section-name-declaration> # SECTION
[# <priority-number>] <.>
{<copy-sentence> !
<section-body>}

P7 <priority-number> ::=
[0]... {<empty> ! <non-zero-digit>} <digit>

SECTION NAME

P8 <section-name-declaration> ::=
<non-reserved-word>

P9 <section-name> ::=
<non-reserved-word> which appears as a
<section-name-declaration>

SECTION BODY

P10 <section-body> ::=
<paragraph>...

PARAGRAPH

P11 ⟨paragraph⟩ ::=
← ⟨paragraph-name-declaration⟩ ⟨.⟩ ↑
{⟨copy-sentence⟩ !
⟨paragraph-body⟩}

PARAGRAPH NAME

P12 <paragraph-name-declaration> ::=
<non-reserved-word>

P13 <paragraph-name> ::=
<non-reserved-word> which appears as a
<paragraph-name-declaration>

P14 <paragraph-name-qualified> ::=
<paragraph-name> [<in-of> <section-name>]

PROCEDURE NAME

P15 <procedure-name> ::=
<section-name> !
<paragraph-name-qualified>

PARAGRAPH BODY

P16 <paragraph-body> ::=
<note-paragraph-body> !
<exit-paragraph-body> !
<alterable-go-to-paragraph-body> !
<regular-paragraph-body>

P17 <note-paragraph-body> ::=
<note-sentence> [<comment-sentence>]...

P18 <exit-paragraph-body> ::=
<exit-sentence>

P19 <alterable-go-to-paragraph-body> ::=
<alterable-go-to-sentence>

P20 <regular-paragraph-body> ::=
{<regular-sentence> !
<other-language-block>}
[<regular-sentence> !
<other-language-block> !
<note-sentence>]...

SENTENCES

P21 <regular-sentence> ::=
<regular-imperative-sentence> !
<conditional-sentence> !
<enter-routine-sentence>

P22 <other-language-block> ::=
<enter-other-language-sentence>
<other-language-string>
<enter-cobol-sentence>

IMPERATIVE SENTENCES

P23 <exit-sentence> ::=
<exit-statement> <.>

P24 <alterable-go-to-sentence> ::=
<alterable-go-to-statement> <.>

P25 <regular-imperative-sentence> ::=
<regular-imperative-statement> <.>

CONDITIONAL SENTENCE

P26 <conditional-sentence> ::=
[<continuable-imperative-statement> <;>]
<conditional-statement> <.>

COMPILER DIRECTING SENTENCES

P27 <note-sentence> ::=
<note-statement> <.>

P28 <comment-sentence> ::=
<comment-string> <.>

P29 <enter-routine-sentence> ::=
<enter-routine-statement> <.>

P30 <enter-other-language-sentence> ::=
<enter-other-language-statement> <.>

P31 <enter-cobol-sentence> ::=
<enter-cobol-statement> <.>

DECLARATIVE SENTENCE

P32 <declarative-sentence> ::=
<declarative-statement> <.>

IMPERATIVE STATEMENTS

P33 <regular-imperative-statement> ::=
<continuable-imperative-statement>
[<;> <terminating-imp-verb-statement>] !
<terminating-imp-verb-statement>

P34 <continuable-imperative-statement> ::=
<continuable-imp-verb-statement>
[<continuable-imp-verb-statement>]...

P35 <continuable-imp-verb-statement> ::=
<imp-arithmetic-statement> !
<move-statement> !
<examine-statement> !
<alter-statement> !
<go-to-depending-statement> !
<perform-statement> !
<stop-literal-statement> !
<set-statement> !
<imperative-i-o-statement> !
<release-statement> !
<sort-statement> !
<generate-statement> !
<initiate-statement> !
<terminate-statement>

P36 <imp-arithmetic-statement> ::=
<imp-add-statement> !
<imp-subtract-statement> !
<imp-multiply-statement> !
<imp-divide-statement> !
<imp-compute-statement>

P37 <imperative-i-o-statement> ::=
<accept-statement> !
<display-statement> !
<open-statement> !
<close-statement> !
<imperative-write-statement> !
<seek-statement>

P38 <terminating-imp-verb-statement> ::=
<simple-go-to-statement> !
<stop-run-statement>

CONDITIONAL STATEMENTS

P39 <conditional-statement> ::=
<if-statement> !
<imp-arithmetic-statement>
<;> <size-error-phrase> !
<search-statement> !
<read-statement> !
<write-invalid-key-statement> !
<return-statement>

DECLARATIVE STATEMENTS

P40 <declarative-statement> ::=
<use-error-statement> !
<use-label-statement> !
<use-before-reporting-statement>

COMMON OPTIONS

P41 <size-error-phrase> ::=
[ON #] SIZE # ERROR #
<regular-imperative-statement>

P42 <at-end-phrase> ::=
[AT #] END #
<regular-imperative-statement>

P43 <invalid-key-phrase> ::=
INVALID # [KEY #]
<regular-imperative-statement>

COMMON TERMS

P44 <result> ::=
<elem-item-name-identifier> [# <rounded>]

P45 <rounded> ::=
ROUNDED

P46 <numeric-operand> ::=
<elem-item-name-identifier> !
<numeric-literal>

P47 <literal-for-display-stop> ::=
<simple-figurative-constant> !
<proper-numeric-literal> !
<proper-nonnumeric-literal>

ARITHMETIC EXPRESSION

P48 <arithmetic-expression> ::=
<term> [# {+ ! -} # <term>]...

P49 <term> ::=
<factor> [# {* ! /} # <factor>]...

P50 <factor> ::=
<primary> [# ** # <primary>]...

P51 <primary> ::=
[{+ ! -} #] <unsigned-primary>

P52 <unsigned-primary> ::=
<numeric-operand> !
(<arithmetic-expression>)

CONDITIONS

P53 <condition> ::=
<condition-term> [# OR # <condition-term>]...

P54 <condition-term> ::=
<condition-factor> [# AND # <condition-factor>]...

P55 <condition-factor> ::=
[<not> #] <condition-primary>

P56 <not> ::=
NOT

P57 <condition-primary> ::=
<simple-condition> !
<abbreviated-relation-condition> !
(<condition>)

P58 <simple-condition> ::=
<relation-condition> !
<class-condition> !
<condition-name-condition> !
<switch-status-condition> !
<sign-condition>

P59 <abbreviated-relation-condition> ::=
<relation-condition>
[# AND # <abbreviation>]...
[# OR # <abbreviation>
[# AND # <abbreviation>]...]...

P60 <abbreviation> ::=
[[<not> #] <relational-operator> #]
<relation-operand>

P61 <relation-condition> ::=
<relation-operand> #
<relational-operator> #
<relation-operand>

P62 <relation-operand> ::=
<item-name-identifier> !
<index-name-qualified> !
<nonnumeric-literal> !
<arithmetic-expression>

P63 <relational-operator> ::=
[IS #] [<not> #] {> ! GREATER [# THAN]} !
[IS #] [<not> #] {< ! LESS [# THAN]} !
[IS #] [<not> #] {= ! EQUAL [# TO]}

P64 <class-condition> ::=
<item-name-identifier> #
[IS #] [<not> #] {NUMERIC ! ALPHABETIC}

P65 <condition-name-condition> ::=
<condition-name-identifier>

P66 <switch-status-condition> ::=
<switch-status-name>

P67 <sign-condition> ::=
<elem-item-name-identifier> #
[IS #] [<not> #] {POSITIVE ! NEGATIVE ! ZERO}

ACCEPT STATEMENT

P68 <accept-statement> ::=
ACCEPT # <item-name-identifier>
[# FROM #
{<mnemonic-name-for-individual-io-unit> !
<mnemonic-name-for-type-of-io-unit>}]

ADD STATEMENT

P69 <imp-add-statement> ::=
ADD # <numeric-operand>
[<,> <numeric-operand>]... #
TO # <result>
[<,> <result>]... !
ADD # <numeric-operand>
{<,> <numeric-operand>}... #
GIVING # <result> !
ADD # {CORRESPONDING ! CORR} #
<non-elem-data-name-identifier> #
TO # <non-elem-data-name-identifier> [# <rounded>]

ALTER STATEMENT

P70 <alter-statement> ::=
ALTER # <alteration> [<,> <alteration>]...

P71 <alteration> ::=
<paragraph-name-qualified> #
TO # [PROCEED # TO #] <destination>

CLOSE STATEMENT

P72 <close-statement> ::=
CLOSE # <closure> [<,> <closure>]...

P73 <closure> ::=
<non-ms-file-name>
[# <reel>] [[# WITH] # {<no-rewind> ! <lock>}] !
<sequential-ms-file-name>
[# <unit>] [[# WITH] # <lock>] !
<random-ms-file-name>
[[# WITH] # <lock>]

P74 <reel> ::=
REEL

P75 <unit> ::=
UNIT

P76 <no-rewind> ::=
NO # REWIND

P77 <lock> ::=
LOCK

COMPUTE STATEMENT

P78 <imp-compute-statement> ::=
COMPUTE # <result> # = # <arithmetic-expression>

DISPLAY STATEMENT

P79 <display-statement> ::=
DISPLAY # <display-operand>
[<,> <display-operand>]...
[# UPON #
{<mnemonic-name-for-individual-io-unit> !
<mnemonic-name-for-type-of-io-unit>}]

P80 <display-operand> ::=
<item-name-identifier> !
<literal-for-display-stop>

DIVIDE STATEMENT

P81 <imp-divide-statement> ::=
DIVIDE # <numeric-operand> #
INTO # <result> !
DIVIDE # <numeric-operand> #
INTO # <numeric-operand> # GIVING # <result>
[# REMAINDER # <elem-item-name-identifier>] !
DIVIDE # <numeric-operand> #
BY # <numeric-operand> # GIVING # <result>
[# REMAINDER # <elem-item-name-identifier>]

ENTER STATEMENT

P82 <enter-routine-statement> ::=
ENTER # <other-language-name> # <routine-name>

P83 <enter-other-language-statement> ::=
ENTER # <other-language-name>

P84 <enter-cobol-statement> ::=
ENTER # COBOL

P85 <routine-name> ::=
<non-reserved-word>

EXAMINE STATEMENT

P86 <examine-statement> ::=
EXAMINE # <item-name-identifier> #
TALLYING #
{ALL ! LEADING ! UNTIL # FIRST} #
<one-character-literal>
[# REPLACING # BY #
<one-character-literal>] !
EXAMINE # <item-name-identifier> #
REPLACING #
{ALL ! LEADING ! FIRST ! UNTIL # FIRST} #
<one-character-literal> #
BY #
<one-character-literal>

P87 <one-character-literal> ::=
" <computer-character> diff <quotation-mark> " !
<digit> !
<simple-figurative-constant>

EXIT STATEMENT

P88 <exit-statement> ::=
EXIT

GENERATE STATEMENT

P89 <generate-statement> ::=
GENERATE # <report-name> !
GENERATE # <report-group-name-qualified>

GO TO STATEMENT

P90 <simple-go-to-statement> ::=
GO # TO # <destination>

P91 <alterable-go-to-statement> ::=
GO # TO [# <destination>]

P92 <go-to-depending-statement> ::=
GO # TO # <destination> {<,> <destination>}... #
DEPENDING # [ON #] <elem-item-name-identifier>

P93 <destination> ::=
<procedure-name>

IF STATEMENT

P94 <if-statement> ::=
IF # <condition>
<;> {<statement> ! NEXT # SENTENCE}
<;> ELSE # {<statement> ! NEXT # SENTENCE}

P95 <statement> ::=
<regular-imperative-statement> !
[<continuable-imperative-statement> <;>]
<conditional-statement>

INITIATE STATEMENT

P96 <initiate-statement> ::=
INITIATE # <report-name> [<,> <report-name>]...

MOVE STATEMENT

P97 <move-statement> ::=
MOVE # {<literal> ! <item-name-identifier>} #
TO # <item-name-identifier>
[<,> <item-name-identifier>]... !
MOVE # {CORRESPONDING ! CORR} #
<non-elem-item-name-identifier> #
TO # <non-elem-item-name-identifier>

MULTIPLY STATEMENT

P98 <imp-multiply-statement> ::=
MULTIPLY # <numeric-operand> # BY # <result> !
MULTIPLY # <numeric-operand> #
BY # <numeric-operand> # GIVING # <result>

NOTE STATEMENT

P99 <note-statement> ::=
NOTE # <comment-string>

OPEN STATEMENT

```
P100  <open-statement> ::=
          OPEN <open-options>

P101 <open-options> ::=
          {<input-option> ↓
          [<output-option>] ↓
          [<i-o-option>]} !
          {<output-option> ↓
          [<i-o-option>]} !
          <i-o-option>

P102 <input-option> ::=
          # INPUT # <input-file> [<,> <input-file>]...

P103 <output-option> ::=
          # OUTPUT # <output-file> [<,> <output-file>]...

P104 <i-o-option> ::=
          # I-O # <i-o-file> [<,> <i-o-file>]...

P105 <input-file> ::=
          <ms-file-name> !
          <non-ms-file-name>
          [# <reversed> ! [# WITH] # <no-rewind>]

P106 <output-file> ::=
          <ms-file-name> !
          <non-ms-file-name> [[# WITH] # <no-rewind>]

P107 <i-o-file> ::=
          <ms-file-name>

P108 <reversed> ::=
          REVERSED
```

PERFORM STATEMENT

```
P109  <perform-statement> ::=
          PERFORM # <range>
          [# <times-option> !
          # <until-option> !
          # <varying-option>]

P110 <range> ::=
          <procedure-name>
          [# {THROUGH ! THRU} # <procedure-name>]

P111 <times-option> ::=
          <integral-operand> # TIMES

P112 <until-option> ::=
          UNTIL # <condition>

P113 <varying-option> ::=
          VARYING # <varying-control-phrase>
          [# AFTER # <varying-control-phrase>
          [# AFTER # <varying-control-phrase>]]

P114 <varying-control-phrase> ::=
          <control-variable> #
          FROM #
          {<index-name-qualified> ! <numeric-operand>} #
          BY #
          <numeric-operand> #
          UNTIL # <condition>

P115 <integral-operand> ::=
          <integer> !
          <elem-item-name-identifier>

P116 <control-variable> ::=
          <elem-item-name-identifier> !
          <index-name-qualified>
```

READ STATEMENT

P117 <read-statement> ::=
READ # <sequential-file-name> [# RECORD]
[# INTO # <item-name-identifier>]
<;> <at-end-phrase> !
READ # <random-ms-file-name> [# RECORD]
[# INTO # <item-name-identifier>]
<;> <invalid-key-phrase>

RELEASE STATEMENT

P118 <release-statement> ::=
RELEASE # <sort-record-name-qualified>
[# FROM # <item-name-identifier>]

RETURN STATEMENT

P119 <return-statement> ::=
RETURN # <sort-file-name> [# RECORD]
[# INTO # <item-name-identifier>]
<;> <at-end-phrase>

SEARCH STATEMENT

P120 <search-statement> ::=
SEARCH # <02-49-item-name-qualified>
[# VARYING # <control-variable>]
[<;> <at-end-phrase>]
{<;> <when-phrase>}... !
SEARCH # ALL # <02-49-item-name-qualified>
[<;> <at-end-phrase>]
<;> <when-phrase>

P121 <when-phrase> ::=
WHEN # <condition> #
{<regular-imperative-statement> !
NEXT # SENTENCE}

SEEK STATEMENT

P122 <seek-statement> ::=
SEEK # <random-ms-file-name> [# RECORD]

SET STATEMENT

P123 <set-statement> ::=
SET # <control-variable>
[<,> <control-variable>]... #
TO #
{<integral-operand> ! <index-name-qualified>} !
SET # <index-name-qualified>
[<,> <index-name-qualified>]... #
{UP ! DOWN} # BY # <integral-operand>

SORT STATEMENT

P124 <sort-statement> ::=
SORT # <sort-file-name> #
<key-clause> [<;> <key-clause>]... #
<sort-input-specification> #
<sort-output-specification>

P125 <key-clause> ::=
[ON #] {ASCENDING ! DESCENDING} # [KEY #]
<sort-key-identifier>
[<,> <sort-key-identifier>]...

P126 <sort-input-specification> ::=
USING # <non-sort-file-name> !
INPUT # PROCEDURE # [IS #] <sort-procedure-range>

P127 <sort-output-specification> ::=
GIVING # <non-sort-file-name> !
OUTPUT # PROCEDURE # [IS #] <sort-procedure-range>

P128 <sort-procedure-range> ::=
<section-name>
[# {THROUGH ! THRU} # <section-name>]

P129 <sort-key-identifier> ::=
<sort-record-name-qualified> !
<non-elem-02-48-item-name-identifier> !
<elem-02-49-item-name-identifier>

STOP STATEMENT

P130 <stop-literal-statement> ::=
STOP # <literal-for-display-stop>

P131 <stop-run-statement> ::=
STOP # RUN

SUBTRACT STATEMENT

P132 <imp-subtract-statement> ::=
SUBTRACT # <numeric-operand>
[<,> <numeric-operand>]... #
FROM # <result> [<,> <result>]... !
SUBTRACT # <numeric-operand>
[<,> <numeric-operand>]... #
FROM # <numeric-operand> # GIVING # <result> !
SUBTRACT # {CORRESPONDING ! CORR} #
<non-elem-item-name-identifier> #
FROM # <non-elem-item-name-identifier>
[# <rounded>]

TERMINATE STATEMENT

P133 <terminate-statement> ::=
TERMINATE # <report-name> [<,> <report-name>]...

USE STATEMENT

```
P134  <use-error-statement> ::=
          USE # AFTER # [STANDARD #] ERROR #
          PROCEDURE # [ON #]
          {<non-sort-file-name>
          [<,> <non-sort-file-name>]... !
          INPUT !
          OUTPUT !
          I-O}

P135 <use-label-statement> ::=
          USE # {BEFORE ! AFTER} # [STANDARD #]
          [<beginning> # ! <ending> #]
          {[<reel> # ! <file> #] LABEL # PROCEDURE # [ON #]
          {<non-ms-file-name>
          [<,> <non-ms-file-name>]... !
          INPUT !
          OUTPUT} !
          [<unit> # ! <file> #] LABEL # PROCEDURE # [ON #]
          {<sequential-ms-file-name>
          [<,> <sequential-ms-file-name>]... !
          INPUT !
          OUTPUT !
          I-O} !
          [<file> #] LABEL # PROCEDURE # [ON #]
          {<random-ms-file-name>
          [<,> <random-ms-file-name>]... !
          INPUT !
          OUTPUT !
          I-O}}

P136 <use-before-reporting-statement> ::=
          USE # BEFORE # REPORTING #
          <report-group-name-qualified>

P137 <file> ::=
          FILE

P138 <beginning> ::=
          BEGINNING

P139 <ending> ::=
          ENDING
```

WRITE STATEMENT

P140 <imperative-write-statement> ::=
WRITE # <non-ms-record-name-qualified>
[# FROM # <item-name-identifier>]
[# <advancing-phrase>]

P141 <write-invalid-key-statement> ::=
WRITE # <ms-record-name-qualified>
[# FROM # <item-name-identifier>]
<;> <invalid-key-phrase>

P142 <advancing-phrase> ::=
{BEFORE ! AFTER} # ADVANCING #
{<integral-operand> [# LINES] !
<mnemonic-name-for-paper-advance>}

L. COBOL LIBRARY

STRUCTURE OF LIBRARY CALLS

```
L1    <copy-entry> ::=
          <copy-clause> <.>

L2    <copy-clause> ::=
          <library-call>

L3    <copy-sentence> ::=
          <copy-statement> <.>

L4    <copy-statement> ::=
          <library-call>

L5    <library-call> ::=
          COPY # <library-name>
          [# REPLACING
          # <word> # BY # <replacement>
          [<,> <word> # BY # <replacement>]...]

L6    <replacement> ::=
          <word> ! <literal> !
          <data-name-identifier> !
          <procedure-name>
```

LIBRARY NAME

L7 <library-name> ::=
<non-reserved-word>

R. RESERVED WORDS

R1 <reserved-word> ::=

ACCEPT ! ACCESS ! ACTUAL ! ADD ! ADDRESS !
ADVANCING ! AFTER ! ALL ! ALPHABETIC ! ALTER !
ALTERNATE ! AND ! ARE ! AREA ! AREAS !
ASCENDING ! ASSIGN ! AT ! AUTHOR ! BEFORE !
BEGINNING ! BLANK ! BLOCK ! BY ! CF ! CH !
CHARACTERS ! CLOCK-UNITS ! CLOSE ! COBOL ! CODE !
COLUMN ! COMMA ! COMP ! COMPUTATIONAL ! COMPUTE !
CONFIGURATION ! CONTAINS ! CONTROL ! CONTROLS !
COPY ! CORR ! CORRESPONDING ! CURRENCY ! DATA !
DATE-COMPILED ! DATE-WRITTEN ! DE !
DECIMAL-POINT ! DECLARATIVES ! DEPENDING !
DESCENDING ! DETAIL ! DISPLAY ! DIVIDE !
DIVISION ! DOWN ! ELSE ! END ! ENDING ! ENTER !
ENVIRONMENT ! EQUAL ! ERROR ! EVERY ! EXAMINE !
EXIT ! FD ! FILE ! FILE-CONTROL ! FILE-LIMIT !
FILE-LIMITS ! FILLER ! FINAL ! FIRST ! FOOTING !
FOR ! FROM ! GENERATE ! GIVING ! GO ! GREATER !
GROUP ! HEADING ! HIGH-VALUE ! HIGH-VALUES !
I-O ! I-O-CONTROL ! IDENTIFICATION ! IF ! IN !
INDEX ! INDEXED ! INDICATE ! INITIATE ! INPUT !
INPUT-OUTPUT ! INSTALLATION ! INTO ! INVALID !
IS ! JUST ! JUSTIFIED ! KEY ! KEYS ! LABEL !
LAST ! LEADING ! LEFT ! LESS ! LIMIT ! LIMITS !
LINE ! LINE-COUNTER ! LINES ! LOCK ! LOW-VALUE !
LOW-VALUES ! MEMORY ! MODE ! MODULES ! MOVE !
MULTIPLE ! MULTIPLY ! NEGATIVE ! NEXT ! NO !
NOT ! NOTE ! NUMBER ! NUMERIC ! OBJECT-COMPUTER !
OCCURS ! OF ! OFF ! OMITTED ! ON ! OPEN !
OPTIONAL ! OR ! OUTPUT ! PAGE ! PAGE-COUNTER !
PERFORM ! PF ! PH ! PIC ! PICTURE ! PLUS !
POSITION ! POSITIVE ! PROCEDURE ! PROCEED !
PROCESSING ! PROGRAM-ID ! QUOTE ! QUOTES !
RANDOM ! RD ! READ ! RECORD ! RECORDS !
REDEFINES ! REEL ! RELEASE ! REMARKS ! RENAMES !
REPLACING ! REPORT ! REPORTING ! REPORTS !
RERUN ! RESERVE ! RESET ! RETURN ! REVERSED !
REWIND ! RF ! RH ! RIGHT ! ROUNDED ! RUN ! SAME !
SD ! SEARCH ! SECTION ! SECURITY ! SEEK !
SEGMENT-LIMIT ! SELECT ! SENTENCE ! SEQUENTIAL !
SET ! SIGN ! SIZE ! SORT ! SOURCE !
SOURCE-COMPUTER ! SPACE ! SPACES !
SPECIAL-NAMES ! STANDARD ! STATUS ! STOP !
SUBTRACT ! SUM ! SYNC ! SYNCHRONIZED ! TALLY !
TALLYING ! TAPE ! TERMINATE ! THAN ! THROUGH !
THRU ! TIMES ! TO ! TYPE ! UNIT ! UNTIL ! UP !
UPON ! USAGE ! USE ! USING ! VALUE ! VALUES !
VARYING ! WHEN ! WITH ! WORDS ! WORKING-STORAGE !
WRITE ! ZERO ! ZEROES ! ZEROS

INDEX
OF THE FORMAL DEFINITION
OF COBOL SYNTAX

INDEX OF THE ECMA TC6 SYNTAX DEFINITION OF COBOL.

META-VARIABLE	DEFN	USED
<->	D171	D160
<--seq>	D160	D146 D150 D155
<--seq-or-bo,>	D155	D150
<.>	G20	G47 T1 I1 I3 I5 I7 I8 I9 I10 I11 I12 I14 E1 E3 E5 E7 E9 E10 E12 E16 E18 E45 E48 E49 E50 E51 E70 E72 D1 D3 D7 D11 D14 D15 D27 D31 D35 D36 D37 D38 D39 D41 D47 D48 D49 D50 D51 D52 D53 D54 D86 D87 D88 D89 D90 D91 D99 D100 D101 D102 D105 D107 D108 P1 P3 P5 P6 P11 P23 P24 P25 P26 P27 P28 P29 P30 P31 P32 L1 L3
<$>	D172	D161
<abbreviated-relation-condition>	P59	P57
<abbreviation>	P60	P59
<accept-statement>	P68	P37
<access-mode-random-clause>	E67	E50
<access-mode-sequential-clause>	E66	E49
<actual-key-clause>	E69	E49 E50
<additional-data-character>	N6	G16
<advancing-phrase>	P142	P140
<alpha-word>	G26	G28
<alter-statement>	P70	P35
<alterable-go-to-paragraph-body>	P19	P16
<alterable-go-to-sentence>	P24	P19
<alterable-go-to-statement>	P91	P24
<alteration>	P71	P70
<alternate-area-clause>	E63	E48 E49
<arithmetic-expression>	P48	P52 P62 P78
<arithmetic-expression-character>	G7	G14
<arithmetic-operator>	G44	T4
<assign-clause>	E57	E48 E49 E50 E51
<assign-individual-units-clause>	E59	E57
<assign-type-clause>	E58	E57
<at-end-phrase>	P42	P117 P119 P120

META-VARIABLE	DEFN	USED
<copy-clause>	L2	D14 D15 D27 D31 D35 D36 D37 D88 D89 D99 L1
<copy-entry>	L1	E7 E10 E16 E45 E70
<copy-sentence>	L3	P5 P6 P11
<copy-statement>	L4	L3
<cs>	E41	D146 D148 D149 D151 D172
<cs-float>	D151	D147
<cs-float-pict>	D147	D144
<cs-seq>	D161	D147 D151 D156
<cs-seq-or-b0,>	D156	D151
<currency-sign>	G9	G14
<currency-sign-clause>	E37	E18
<currency-sign-declaration>	E38	E37 E40
<currency-symbol>	E40	E41
<data-division>	D1	C1
<data-division-body>	D2	D1
<data-name-identifier>	D234	E65 E69 L6
<data-records-clause>	D128	D16 D28
<date-compiled-paragraph>	I7	I2
<date-written-paragraph>	I10	I2
<decimal-fraction>	G32	G33
<decimal-point>	E43	G32 D175
<decimal-point-clause>	E42	E18 E43 E44
<declarative-portion>	P3	P2
<declarative-section>	P5	P3
<declarative-sentence>	P32	P5
<declarative-statement>	P40	P32
<destination>	P93	P71 P90 P91 P92
<dig-seq>	D157	D143
<digit>	G4	G15 G23 G24 G31 E15 E39 D46 P7 P87
<digit-separator>	E44	D164
<display-operand>	P80	P79
<display-statement>	P79	P37
<elem-01-77-clauses>	D60	D39 D59 D86
<elem-01-77-red-clauses>	D92	D87 D91
<elem-02-49-item-name>	D79	D213
<elem-02-49-item-name-declaration>	D78	D77 D79
<elem-02-49-item-name-identifier>	D226	D228 P129
<elem-02-49-item-name-qualified>	D213	D214 D219 D226
<elem-66-item-name>	D83	D216

META-VARIABLE	DEFN	USED
<elem-66-item-name-declaration>	D82	D40 D83
<elem-66-item-name-qualified>	D216	D219 D228
<elem-clauses>	D59	D49
<elem-data-name-identifier>	D233	D234
<elem-item-name-identifier>	D228	D229 D231 D233 P44 P46 P67 P81 P92 P115 P116
<elem-item-name-qualified>	D219	D135 D136 D180 D193 D220 D236
<elem-ms-record-name>	D70	D194 D200
<elem-ms-record-name-declaration>	D69	D36 D70
<elem-ms-record-name-qualified>	D200	D205 D208
<elem-non-ms-record-name>	D66	D194 D198
<elem-non-ms-record-name-declaration>	D65	D35 D66
<elem-non-ms-record-name-qualified>	D198	D204 D208
<elem-non-sum-field-name-declaration>	D120	D108
<elem-non-sum-group-clauses>	D111	D102
<elem-non-sum-report-group-name>	D115	D210
<elem-non-sum-report-group-name-declaration>	D114	D99 D115
<elem-non-sum-report-group-name-qualified>	D210	D211
<elem-non-sum-report-group-spec>	D102	D99
<elem-non-sum-spec>	D108	D106
<elem-record-name-qualified>	D208	D219 D228
<elem-record-spec>	D39	D35 D36 D37 D88
<elem-red-clauses>	D61	D50 D59
<elem-renames-clause>	D180	D40
<elem-report-spec>	D106	D104
<elem-sort-record-name>	D74	D194 D202
<elem-sort-record-name-declaration>	D73	D37 D74
<elem-sort-record-name-qualified>	D202	D206 D208
<elem-spec>	D49	D43
<elem-ws-record-name>	D98	D195 D208
<elem-ws-record-name-declaration>	D97	D88 D89 D98
<empty>	G1	T2 E15 P7
<end-of-reel-rerun-condition>	E79	E78
<ending>	P139	P135
<enter-cobol-sentence>	P31	P22
<enter-cobol-statement>	P84	P31

META-VARIABLE	DEFN	USED
<integral-operand>	P115	P111 P123 P142
<invalid-key-phrase>	P43	P117 P141
<item-name-identifier>	D229	D127 D183 D187 D230 P62 P64 P68 P80 P86 P97 P117 P118 P119 P140 P141
<item-name-qualified>	D220	D181
<justified-clause>	D130	D60 D61 D62 D92 D107 D110 D111
<key-clause>	P125	P124
<key-option>	D136	D134 D135
<label-records-clause>	D131	D16
<letter>	G5	G15 G23 G24 G26
<level-number>	D46	D45
<library-call>	L5	L2 L4
<library-name>	L7	L5
<line-number-clause>	D132	D105 D107 D108 D109 D110 D111
<literal>	G43	E65 D191 D193 P97 L6
<literal-for-display-stop>	P47	P80 P130
<literal-string>	G29	G30
<lock>	P77	P73
<memory-size-clause>	E13	E12
<mnemonic-name-declaration-for-code-for-report-groups>	E26	E21 E31
<mnemonic-name-declaration-for-individual-io-unit>	E23	E21 E28
<mnemonic-name-declaration-for-individual-switch>	E27	E22
<mnemonic-name-declaration-for-paper-advance>	E25	E21 E30
<mnemonic-name-declaration-for-type-of-io-unit>	E24	E21 E29
<mnemonic-name-for-code-for-report-groups>	E31	D125
<mnemonic-name-for-individual-io-unit>	E28	P68 P79
<mnemonic-name-for-paper-advance>	E30	P142
<mnemonic-name-for-type-of-io-unit>	E29	P68 P79
<move-statement>	P97	P35
<ms-file-description>	D15	D5
<ms-file-name>	D24	D25 D199 D200 P105 P106 P107
<ms-record-description>	D36	D5

META-VARIABLE	DEFN	USED
<ms-record-name-qualified>	D205	P141
<multiple-file-clause>	E88	E75
<multiple-file-clauses>	E75	E72
<multiple-reel-clause>	E61	E48
<multiple-unit-clause>	E62	E49
<next-group-clause>	D133	D109 D110 D111
<no-rewind>	P76	P73 P105 P106
<non-declarative-portion>	P4	P2
<non-declarative-section>	P6	P4
<non-elem-01-clauses>	D56	D38 D55
<non-elem-02-48-item-name>	D76	D212
<non-elem-02-48-item-name-declaration>	D75	D47 D48 D51 D52 D76
<non-elem-02-48-item-name-identifier>	D225	D227 P129
<non-elem-02-48-item-name-qualified>	D212	D213 D214 D218 D225
<non-elem-66-item-name>	D81	D215
<non-elem-66-item-name-declaration>	D80	D40 D81
<non-elem-66-item-name-qualified>	D215	D218 D227
<non-elem-clauses>	D55	D47
<non-elem-data-name-identifier>	D232	D234 P69
<non-elem-field-name>	D119	D217
<non-elem-field-name-declaration>	D118	D105 D119
<non-elem-group-clauses>	D109	D100
<non-elem-item-name-identifier>	D227	D229 D232 P97 P132
<non-elem-item-name-qualified>	D218	D220
<non-elem-ms-record-name>	D68	D194 D199
<non-elem-ms-record-name-declaration>	D67	D36 D68
<non-elem-ms-record-name-qualified>	D199	D205 D207
<non-elem-non-ms-record-name>	D64	D194 D197
<non-elem-non-ms-record-name-declaration>	D63	D35 D64
<non-elem-non-ms-record-name-qualified>	D197	D204 D207
<non-elem-record-name-qualified>	D207	D212 D215 D216 D218 D227
<non-elem-record-spec>	D38	D35 D36 D37 D88
<non-elem-red-clauses>	D57	D48 D55
<non-elem-renames-clause>	D181	D40
<non-elem-report-group-name>	D113	D209
<non-elem-report-group-name-declaration>	D112	D99 D113

META-VARIABLE	DEFN	USED
<non-elem-report-group-name-qualified>	D209	D211 D217
<non-elem-report-group-spec>	D100	D99
<non-elem-report-spec>	D105	D104
<non-elem-sort-record-name>	D72	D194 D201
<non-elem-sort-record-name-declaration>	D71	D37 D72
<non-elem-sort-record-name-qualified>	D201	D206 D207
<non-elem-spec>	D47	D43
<non-elem-ws-record-name>	D96	D195 D207
<non-elem-ws-record-name-declaration>	D95	D88 D89 D96
<non-ms-file-description>	D14	D5
<non-ms-file-name>	D18	E52 D23 D25 D197 D198 P73 P105 P106 P135
<non-ms-file-name-declaration>	D17	D14 D18
<non-ms-record-description>	D35	D5
<non-ms-record-name-qualified>	D204	P140
<non-reserved-alpha-word>	G28	E35 E36 D17 D18 D19 D20 D21 D22 D29 D30 D33 D34 D63 D64 D65 D66 D67 D68 D69 D70 D71 D72 D73 D74 D75 D76 D78 D79 D80 D81 D82 D83 D84 D85 D93 D94 D95 D96 D97 D98 D112 D113 D114 D115 D116 D117 D118 D119 D120 D121 D122 D138 D139
<non-reserved-word>	G27	I6 E23 E24 E25 E26 E27 E28 E29 E30 E31 P8 P9 P12 P13 P85 L7
<non-sort-file-name>	D25	E77 E79 E81 E86 E88 D26 P126 P127 P134
<non-switch-special-names-clause>	E21	E20
<non-ws-record-name>	D194	D128 D131
<non-zero-digit>	G2	G4 G36 D46 P7
<nonnumeric-literal>	G41	G43 D192 P62
<nonnumeric-picture>	D145	D142

META-VARIABLE	DEFN	USED
<relation-condition>	P61	P58 P59
<relation-operand>	P62	P60 P61
<relational-operator>	P63	P60 P61
<relative-index-name-qualified>	D237	D236
<release-statement>	P118	P35
<remarks-paragraph>	I12	I2
<replacement>	L6	L5
<report-clause>	D182	D16
<report-description>	D31	D13
<report-group-description>	D99	D13
<report-group-name-qualified>	D211	D185 P89 P136
<report-name>	D34	D182 D203 D209 D210 D222 P89 P96 P133
<report-name-declaration>	D33	D31 D34
<report-section>	D11	D2
<report-section-body>	D12	D11
<report-specification>	D13	D12
<rerun-clause>	E76	E73
<rerun-clauses>	E73	E72
<rerun-condition-1>	E78	E76
<rerun-condition-2>	E80	E76
<rerun-medium>	E77	E76
<reserved-word>	R1	G27 G28
<reset-clause>	D183	D107 D110
<result>	P44	P69 P78 P81 P98 P132
<return-statement>	P119	P39
<reversed>	P108	P105
<rounded>	P45	P44 P69 P132
<routine-name>	P85	P82
<same-block-area-clause>	E86	E84
<same-clause>	E84	E74
<same-clauses>	E74	E72
<same-record-area-clause>	E85	E84
<same-sort-area-clause>	E87	E84
<sd-clauses>	D28	D27
<search-statement>	P120	P39
<section-body>	P10	P5 P6
<section-name>	P9	P14 P15 P128
<section-name-declaration>	P8	P5 P6 P9
<security-paragraph>	I11	I2
<seek-statement>	P122	P37
<segment-limit-clause>	E14	E12
<select-clause-for-non-ms-file>	E52	E48
<select-clause-for-random-ms-file>	E54	E50

META-VARIABLE	DEFN	USED
<select-clause-for-sequential-ms-file>	E53	E49
<select-clause-for-sort-file>	E55	E51
<separator>	T1	T3 T8
<sequential-file-name>	D23	P117
<sequential-ms-file-name>	D20	E53 D23 D24 P73 P135
<sequential-ms-file-name-declaration>	D19	D15 D20
<set-statement>	P123	P35
<sign>	G34	G35
<sign-condition>	P67	P58
<sign-float>	D150	D146
<sign-float-pict>	D146	D144
<simple-condition>	P58	P57
<simple-figurative-constant>	G37	G38 G39 P47 P87
<simple-go-to-statement>	P90	P38
<size-error-phrase>	P41	P39
<skip-into-area-b>	G18	T7
<sort-file-description>	D27	D6
<sort-file-name>	D30	E55 D26 D201 D202 P119 P124
<sort-file-name-declaration>	D29	D27 D30
<sort-file-specification>	D6	D4
<sort-input-specification>	P126	P124
<sort-key-identifier>	P129	P125
<sort-output-assign-clause>	E60	E48 E49
<sort-output-specification>	P127	P124
<sort-procedure-range>	P128	P126 P127
<sort-record-description>	D37	D6
<sort-record-name-qualified>	D206	P118 P129
<sort-statement>	P124	P35
<source-clause>	D184	D108 D111
<source-computer-entry>	E9	E8
<source-computer-paragraph>	E7	E4
<source-computer-paragraph-body>	E8	E7
<source-item-name-identifier>	D230	D184
<space>	G6	G15 G19 E39
<spaces>	G19	G19
<special-character>	G14	G15
<special-item-name-qualified>	D222	D219 D228
<special-names-clause>	E20	E19
<special-names-clauses>	E19	E18
<special-names-entry>	E18	E17
<special-names-paragraph>	E16	E4 E40 E43 E44
<special-names-paragraph-body>	E17	E16
<statement>	P95	P94

META-VARIABLE	DEFN	USED
⟨stop-literal-statement⟩	P130	P35
⟨stop-run-statement⟩	P131	P38
⟨strophe⟩	T8	T9
⟨strophe-mark⟩	G17	T8
⟨structure⟩	T7	T7 T8
⟨sub-number⟩	D45	D42 D103
⟨sub-report-spec⟩	D104	D103
⟨sub-spec⟩	D43	D42
⟨subordinate-entries⟩	D42	D38 D43 D44 D45 D90
⟨subordinate-report-entries⟩	D103	D100 D104
⟨subscripts⟩	D236	D225 D226 D235
⟨sum-clause⟩	D185	D107 D110
⟨sum-field-name⟩	D122	D217
⟨sum-field-name-declaration⟩	D121	D107 D122
⟨sum-field-name-qualified⟩	D217	D221
⟨sum-group-clauses⟩	D110	D101
⟨sum-item-name-qualified⟩	D221	D230 D231 D233
⟨sum-report-group-name⟩	D117	D203
⟨sum-report-group-name-declaration⟩	D116	D99 D117
⟨sum-report-group-name-qualified⟩	D203	D211 D221
⟨sum-report-group-spec⟩	D101	D99
⟨sum-spec⟩	D107	D106
⟨summed-item-name-identifier⟩	D231	D185
⟨supp-pict⟩	D148	D144
⟨switch-rerun-condition⟩	E83	E80
⟨switch-special-names-clause⟩	E22	E20
⟨switch-status-condition⟩	P66	P58
⟨switch-status-name⟩	E36	E83 P66
⟨switch-status-name-assignment⟩	E32	E22
⟨switch-status-name-declaration⟩	E35	E33 E34 E36
⟨synchronized-clause⟩	D186	D60 D61 D62 D92
⟨term⟩	P49	P48
⟨terminate-statement⟩	P133	P35
⟨terminating-character⟩	G10	G13
⟨terminating-imp-verb-statement⟩	P38	P33
⟨times-option⟩	P111	P109
⟨type-clause⟩	D187	D109 D110 D111
⟨unit⟩	P75	P73 P135
⟨unsigned-primary⟩	P52	P51
⟨unsigned-proper-numeric-literal⟩	G33	G35
⟨until-option⟩	P112	P109
⟨usage-clause⟩	D188	D56 D57 D58 D60 D61 D62 D90 D92

META-VARIABLE	DEFN	USED
<usage-is-display-clause>	D189	D105 D107 D108 D109 D110 D111
<usage-is-index-clause>	D190	D38 D39 D51 D52 D53 D54 D86 D87 D90 D91
<use-before-reporting-statement>	P136	P40
<use-error-statement>	P134	P40
<use-label-statement>	P135	P40
<value-clause>	D191	D56 D60 D108 D111
<value-of-clause>	D193	D16
<var-occurs-elem-clauses>	D62	D59
<var-occurs-non-elem-clauses>	D58	D55
<variable-occurs-clause>	D135	D58 D62
<varying-control-phrase>	P114	P113
<varying-option>	P113	P109
<when-phrase>	P121	P120
<word>	G25	G26 G27 T4 N1 N2 N3 N4 N5 N7 N8 N9 L5 L6
<word-element>	G23	G25
<word-terminator>	G24	G25
<working-storage-section>	D7	D2
<working-storage-section-body>	D8	D7
<write-invalid-key-statement>	P141	P39
<ws-record-description>	D88	D10
<ws-record-descriptions>	D10	D8
<ws-record-name>	D195	D178
<z>	D173	D162
<z-or-*-seq>	D152	D148
<z-seq>	D162	D148 D152
<zero-digit>	G3	G4
<zero-figurative-constant>	G40	G42

EXPLANATORY NOTES

EXPLANATORY NOTES

G6 <space>

b represents the cobol-character "space"

G8 <relation-character>

The symbols > (greater than), < (less than) are different from Backus brackets.

G9 <currency-sign>

$ represents the currency-sign defined by the implementor.

G17 <strophe-mark>

← is a cobol-control-character.
In the cobol reference format the strophe-mark is represented by a new line without a hyphen in the continuation area and a skip into area A.

G18 <skip-into-area-b>

↑ is a cobol-control-character.
In the cobol reference format the skip-into-area-b is represented by a skip into area B, if the current position is in area A, otherwise by no skip.

G45 <proper-relational-operator>

The symbols > (greater than), < (less than) are different from Backus brackets.

APPENDIX

APPENDIX

A METALANGUAGE FOR THE DESCRIPTION OF PROGRAMMING LANGUAGES

TABLE OF CONTENTS

2. INTRODUCTION

The syntax of a programming language can be described either with plain English, or in a more formal way, using a formalized metalanguage.

Below, the metalanguage is explained, using English language and examples. Connected examples, demonstrating the use of the metalanguage, can be found at the end of this appendix.

3. TERMINAL-SYMBOLS

The terminal-symbols are the irreducible elements of the language to be described by the metalanguage. Usually the terminal-symbols of a language are its character set and some additional symbols that are used to denote controls and other basic concepts of the language which cannot be defined in terms of the character set.

4. STRINGS

4.1 THE CONCEPT OF A STRING

A string is a sequence of terminal-symbols.

If A and B are terminal-symbols, then A AA BAB are strings.

Strings formed by terminal-symbols are written as such, terminated by space or any symbol that is not a terminal-symbol. ("b" is used to represent the terminal-symbol "space", and "space" is used to terminate strings.) The empty string is a sequence which does not consist of any terminal-symbol, that is, the string which is a sequence of 0 terminal-symbols.

4.2 CONCATENATION OF STRINGS

If A and B are strings, then a new string A B, the concatenation of A and B, is defined as the string, consisting of the ordered sequence of symbols comprising A followed by that comprising B.

Thus, if A is the string MN and B is the string PQR, then the concatenation of A and B is the string MNPQR.

Concatenation is associative: The string resulting from first concatenating A and B and then concatenating the string obtained with C is the same as that resulting from concatenating A with the string obtained by concatenating B and C. To concatenate three or more strings, no brackets of any kind are necessary to show any order of concatenation.

Concatenation is obviously not commutative: The concatenation of A and B is not necessarily the same as the concatenation of B and A. Generally the results will be different.

The empty string is the "identity" string with respect to concatenation: The concatenation of the empty string with any other string is a string identical to the latter; the concatenation of any string with the empty string is a string identical with the former.

4.3 SUBSTRINGS

The string A is a substring of the string C, if there exist strings X and Y such that C equals the concatenatio of X, A and Y. It may be that either or both X and Y are empty strings.

Thus the string RST is a substring of the string PQRSTUVW There are strings PQ and UVW such that the string PQRSTUV is the concatenation of the strings PQ, RST and UVW.

Other examples: MN is a substring of MNPQR, FGH is a substring of EFGH.

The substring relation is reflexive: Any string A contai this same string A as a trivial substring.

The substring relation is transitive: If A is a substring of B, and B is a substring of C, then A is a substring of

The substring relation is not symmetric: If A is a substring of B, then B is not necessarily a substring of A. Generally B will not be a substring of A.

5. SETS OF STRINGS

5.1 THE CONCEPT OF A SET OF STRINGS

Any collection of strings is called a set of strings.

Thus each proper-numeric-literal, permitted in COBOL, is a string of terminals-symbols. All proper-numeric-literals can be grouped together conceptionally to form the set of all proper-numeric-literals.

Further example: Each of the 26 letters A, B, C ... Z is a string, consisting of exactly one terminal-symbol. The 26 letters can be grouped together to form the set of all letters.

The symbol "ϵ" will be used between a string and the name of a set, to indicate that the string on the left-hand side is contained in the set on the right-hand side. Thus "ABCϵ<p>" is an abbreviation for "the string ABC is element of the set <p>". This will only be used in the description of the metalanguage.

The null set of strings is that set that does not contain any strings.

5.2 THE NAME OF A SET OF STRINGS

References to individual sets of strings will be by assigned names. The term meta-variable is used interchangeably with name of a set.

Names of sets will be chosen in such a way that they have some mnemonic relation to the specific set they are assigned to. Any convenient symbols can be used: Terminal-symbols (for instance: digits, period, comma, semi-colon, plus-sign, hyphen, asterisk; upper case letters) or any other additional symbols (for instance: lower case letters). In order to rigorously indicate the begin and the end of the name, each name begins with a left Backus bracket (<) and ends with a right Backus bracket (>).

Examples of names for sets of strings are:
<integer>, <data-division>.

Remark:

The use of Backus brackets is necessary, because it is frequently convenient to utilize terminal-symbols for the formation of names for sets, and further, because it is common practice not to introduce a concatenation operator for sets (similar to multiplication in ordinary arithmetic), thus eliminating the separator between the two operands of concatenation.

Terminal-symbols can be considered as a string consisting of exactly one terminal-symbol. And a set can be formed, consisting of exactly this single string. Though the terminal-symbol and the related set are conceptually different, the terminal-symbol will be used as name for the related one-element set, provided that no misunderstanding is possible.

This slightly extends the conventions for names of sets given above.

Thus B is considered as the name of the set consisting of the single one-character string B, 9 is considered as the name of the set consisting of the single one-character string 9.

Some names of sets have to be used frequently. In this case, unique additional symbols are introduced as abbreviations for the names of the set.

Again, this further extends the conventions for names of sets given above.

Thus # is used as abbreviation for <spaces>, representing the set consisting of strings of one or more spaces.

<null> will be used as name for the null set of strings.

<empty> will be used as name for the set, consisting of the empty string only.

5.3 CONSTRUCTIVE DEFINITION OF SETS

Sets of strings will be defined as intermediate steps for the definition of the set of all syntactically correct programs.

Sets chosen and defined will reflect the syntactic structure of the language being described, and possibly facilitate the description of its semantics in an accompanying English-language document. In other words, sets chosen usually will correspond to concepts or logical entities in the language.

5.3.1 TERMINAL SETS

The construction starts with sets consisting of exactly one string, which in turn consists of exactly one terminal-symbol.

As described above, the terminal-symbol will be used as name for the related terminal set.

5.3.2 STEPS OF THE CONSTRUCTION

Each individual step of the construction is the definition of a new set of strings, in terms of sets already known. It is represented by a meta-definition: The name of the set to be defined, followed by the definition-symbol (::=), which means "is defined as", followed by names of sets already defined and meta-operators representing the principles of construction to be applied.

Example:
<number> ::= <unsigned-number> | <signed-number>

The detail explanation of the operators representing the principles of construction follows below.

5.4 UNION-OPERATION FOR SETS

The union-operation permits the constructive definition of a new set in terms of two other sets already known. The principles of construction represented by the union-operator are explained below.

If <a> and <b> are sets of strings, then a new set <c>::= <a>|<b>, the union of <a> and <b>, is defined as the set consisting of exactly those strings, that are present either in <a> or in <b> or in both.

Or more formally:

$\underline{X} \in$ <a>|<b>, if and only if $\underline{X} \in$ <a> or $\underline{X} \in$ <b>

Thus, if <set-1> consists of the 3 strings AB, CDE, F and if <set-2> consists of the 2 strings X, YZ then

<new-set>::=<set-1>|<set-2>

defines a new set consisting of the following 5 strings:

AB CDE F
X YZ

Further example:

<number>::=<unsigned-number>|<signed-number>

The concept "number" is defined in terms of the concepts "unsigned-number" and "signed-number", that are assumed to be known. The principle of construction is determined by the union-operator: Any unsigned-number or any signed-number is to be considered as number, that is as a string of the set <number>.

The union-operation for sets is associative:
The set resulting from first combining <a> and <b> and then combining the set obtained with <c> is the same as the set resulting from combining <a> with the set obtained by combining <b> and <c>. To combine three or more sets with the union-operation, no brackets of any kind are necessary to show any order of the union-operations.

The union-operation for sets is commutative: The set

resulting from combining <a> and <b> is the same as the set resulting from combining <b> and <a>.

The set <null> is the "identity" with respect to the union-operation: The union of <null> and any other set is a set identical to the latter.

Note. In the other parts of the present book, the union operator is represented by an exclamation mark ! for typographical reasons.

5.5 CONCATENATION-OPERATION FOR SETS

The concatenation-operation permits the constructive definition of a new set in terms of two other sets already known. The principles of construction represented by the concatenation-operator are explained below.

If <a> and <b> are sets of strings, then a new set <c>::=<a>‖<b>, or abbreviated <c>::=<a><b>, the concatenation of <a> and <b>, is defined as the set consisting of exactly those strings, that can be formed by concatenation of any string of <a> with any string of <b>.

Or more formally:

$\underline{Z} \in$ <a><b>, if and only if there are strings $\underline{X} \in$ <a> and $\underline{Y} \in$ <b> such that $\underline{Z}=\underline{X}\underline{Y}$.

Thus, if <set-1> consists of the 3 strings AB, CDE, F and if <set-2> consists of the 2 strings X, YZ then

<new-set>::=<set-1><set-2>

defines a new set consisting of the following 6 strings:

ABX ABYZ
CDEX CDEYZ
FX FYZ

Further example:

<signed-number>::=<sign><unsigned-number>

The concept "signed-number" is defined in terms of the concepts "sign" and "unsigned-number", that are assumed to be known. The principle of construction is determined by the concatenation-operator: Any sign (+ or -) followed by any unsigned-number is a string that is to be considered as a signed-number, that is, as a string of the set <signed-number>.

The concatenation-operation for sets is associative: The set resulting from first concatenating <a> and <b> and then concatenating the set obtained with <c> is the same as the set resulting from concatenating <a> with the set obtained from concatenating <b> and <c>.

To concatenate three or more sets, no brackets of any kind are necessary to show any order of concatenation.

Concatenation of sets is obviously not commutative: The concatenation of <a> and <b> is not necessarily the same as the concatenation of <b> and <a>. Generally the results will be different.

The set <empty> is the "identity" with respect to concatenation: The concatenation of <empty> with any other set is a set identical to the latter; the concatenation of any set with <empty> is a set identical to the former.

The set <null> is the"zero" with respect to concatenation: The concatenation of <null> with any other set is a set identical to <null>; the concatenation of any set with <null> is a set identical to <null>.

Union-operation and concatenation-operation are distributive:

The set resulting from first forming the union of <a> and <b> and then concatenating the set obtained with <d> is the same as the set resulting from forming the union of the set obtained from concatenating <a> and <d> and of the set obtained from concatenating <b> and <d>.

Similarly the set resulting from first forming the union of <a> and <b> and then concatenating <d> with the set obtained is the same as the set resulting from forming the union of the set obtained from concatenating <d> and <a> and of the set obtained from concatenating <d> and <b>.

5.6 CONTAINING-OPERATION FOR SETS

The "containing"-operation permits the constructive definition of a new set in terms of two other sets already known. The principles of construction represented by the "containing"-operator are explained below:

If <a> and <b> are sets of strings, then a new set <c>::=<a> containing <b>, or abbreviated <c>::=<a> containing <b>, is defined as the set consisting of exactly those strings of <a>, that contain a substring that belongs to <b>.

Or more formally:

$\underline{X} \in$ <a> containing <b>, if and only if $\underline{X} \in$ <a> and there is a string $\underline{Y}$ such that $\underline{Y}$ substring of $\underline{X}$ and $\underline{Y} \in$ <b>.

Thus, if <set-1> consists of the 5 strings AX, YBZ, 17YZ, X32, Z3 and if <set-2> consists of the 2 strings X, YZ then

<new-set>::=<set-1> containing <set-2>

defines a new set consisting of the following 3 strings

AX, 17YZ, X32

Further example:

<alphaword>::=<word> containing <letter>

The concept "alphaword" is defined interms of the concepts "word" and "letter", that are assumed to be known. The principle of construction is determined by the "containing" operator: Any word that contains a letter is to be considered as alpha-word, that is as a string of the set <alpha-word>.

The example above is representative for the main application of the "containing"-operator: To characterize subclasses of names in programming languages, the members of which are requested to contain for instance at least one letter, but not necessarily as the first or last symbol.

<x> containing <null> equals <null>, for any set <x>. The set <null> is a "right-hand zero" with respect to the "containing"-operation.

<null> containing <x> equals <null>, for any set <x>. The set <null> is a "left-hand zero" with respect to the "containing"-operation.

<x> containing <empty> equals <x>, for any set <x>. This is implied by the fact that every string contains the empty string as a substring. The set <empty> is a "right-hand identity" with respect to the "containing"-operation.

<empty> containing <x> equals <empty>, if <x> contains the empty string.

<empty> containing <x> equals <null>, if <x> does not contain the empty string.

5.7 NOT CONTAINING-OPERATION FOR SETS

The "not-containing"-operation permits the constructive definition of a new set in terms of two other sets already known. The principles of construction represented by the "not-containing"-operator are explained below.

If <a> and <b> are sets of strings, then a new set <c>::=<a> not-containing <b>, abbreviated <c>::=<a> not-containing <b>, is defined as the set consisting of exactly those strings of <a>, that do not contain a substring that belongs to <b>.

Or more formally:

X ∈ <a> not-containing <b>, if and only if X ∈ <a> and there is no string Y such that Y substring of X and Y ∈ <b>.

Thus, if <set-1> consists of the 5 strings AX, YBZ, 17YZ, X32, Z3, and if <set-2> consists of the 2 strings X, YZ then

<new-set>::=<set-1> not-containing <set-2>

defines a new set consisting of the following 2 strings:

YBZ, Z3

Further example:

<special-label>::= <word> not-containing <letter>

The concept "special-label" is defined in terms of the concepts "word" and "letter", that are assumed to be known. The principle of construction is determined by the "not-containing"-operator: Any word that does not contain a letter is to be considered as special-label, that is as a string of the set<special-label>.

The main application of the "not-containing"-operator is to characterize strings, that are terminated by a definite symbol, or a definite sequence of symbols (for instance comments or non-numeric literals).

<x> not-containing <null> equals <x>, for any set <x>. The set <null> is a "right-hand identity" with respect to the "not-containing"-operation.

<null> not-containing <x> equals <null>, for any set <x>. The set <null> is a "left-hand identity" with respect to the "not-containing"-operation.

<x> not-containing <empty> equals <null>, for any set <x>. This is implied by the fact that every string contains the empty string as a substring.

<empty> not-containing <x> equals <null>, if <x> contains the empty string.

<empty> not-containing <x> equals <empty>, if <x> does not contain the empty string.

5.8 DIFFERENCE-OPERATION FOR SETS

The difference-or "different-from"-operation permits the constructive definition of a new set in terms of two other sets already known. The principles of contructions represented by the difference-operator are explained below.

If <a> and <b> are sets of strings, then a new set <c>::=<a> diff <b>, or abbreviated <c>::=<a> diff <b>, is defined as the set consisting of exactly those strings of <a>, that are not identical with any string of <b>.

Or more formally:

X ∈ <a> diff <b>, if and only if X ∈ <a> and X ∉ <b>.

Thus, if <set-1> consists of the 3 strings AX, YBZ, 17YZ and if <set-2> consists of the 2 strings AX, YZ then

<new-set>::=<set-1> diff <set-2>

defines a new set consisting of the following 2 strings:

YBZ, 17YZ

Further example:

<non-reserved-word>::=<word> diff <reserved-word>

The concept "user-defined-name" is defined in terms of the concepts "word" and "reserved-word", that are assumed to be known. The principle of construction is determined by the difference-operator: Any word that is not identical with a reserved-word can be utilized as a user-defined-name, that is it is a string of the set <user-defined-name>.

The main application of the difference-operator is demonstrated by the example above.

<x> diff <null> equals <x>, for any set <x>. The set <null> is a "right-hand identity". With respect to the difference operation.

<null> diff <x> equals <null>, for any set <x>. The set <null> is a "left-hand zero" with respect to the difference operation.

5.9 OPTION-OPERATION FOR SETS

The option-operation permits the constructive definition of a new set in terms of another set already known. The principles of construction represented by the option-operator are explained below.

If <a> is a set of strings, then a new set <c>::=[<a>], is defined as the set consisting of the

strings of <a> and the empty string.

Or more formally:

$\underline{X} \in [<a>]$, if and only if $\underline{X} \in <a>$ or $\underline{X}$ is the empty string.

Thus, if <set-1> consists of the 3 strings AB, CDE, F then

<new-set>::=[<set-1>]

defines a new set consisting of the following 4 strings:

AB, CDE, F, empty string

Further example:

<optional-sign>::=[<sign>]

The concept "optional-sign" is defined in terms of the concept "sign", that is assumed to be known.

The principle of construction is determined by the option-operator: Any sign, omitted or optionally present, is to be considered as optional-sign, that is a string of the set <optional-sign>.

Combined example:

<number>::=[<sign>] <unsigned-number>

The concept "number" is defined in terms of the concepts "sign" and "unsigned-number", that are assumed to be known. The principle of construction is determined by the option-operator and the concatenation-operator: Any unsigned-number optionally preceded by a sign is to be considered as number, that is as a string of the set <number>.

The following 3 constructions are equivalent:

a) <number>::=<optional-sign><unsigned-number>
<optional-sign>::=<sign>|<empty>

b) <number>::=
<sign><unsigned-number>|<unsigned-number>

c) <number>::=[<sign>]<unsigned-number>

It is obvious, that construction c) is most convenient for the reader. It is very close to the natural language ("optionally preceded by").

It might be useful to note, that the unary option-operation, an operation or the function with a single

operand or argument, is represented in a way different from the one conventionally used in algebra.

In algebra, the unary operators + and - usually refer to the number following the operator, as for instance in -112. If the operators are intended to refer to a more extended expression, the scope of the operator has to be indicated by parentheses, supplementing the operator, as for instance in -(112+0.5).

The same method would be possible for the option-operation as well, and an operator-symbol "opt" could be introduced. If required, the scope could be indicated by some type of meta-brackets.

However, it seems to be more convenient for the reader to use option-brackets. In that way, the scope of the operation is indicated for all cases, even for the simple ones, and the form of the brackets defines the type of operation to be performed (that is the option-operation and not any other unary set operation).

5.10 REPETITION-OPERATION FOR SETS

The repetition-operation permits the constructive definition of a new set in terms of another set already known. The principles of construction represented by the repetition-operator are explained below.

If <a> is a set of strings, then a new set <c>::=<a>●●● is defined as the set consisting of the string of <a> and all those strings, which can be formed by concatenation of two or more strings, each of them belonging to the set <a>.

Or more formally:

$\underline{X} \in$ <a>●●●, if and only if there is an integer $n \geq 1$ and strings $\underline{X}_1, \underline{X}_2, \ldots \underline{X}_n$ such that $\underline{X}_1 \in$ <a>, $\underline{X}_2 \in$ <a>,... $\underline{X}_n \in$ <a> and $\underline{X} = \underline{X}_1 \underline{X}_2 \ldots \underline{X}_n$.

It should be noted that the set <a>●●● is identical to the set

<a>|<a><a>|<a><a><a>|etc.

Thus, if <set-1> consists of the 2 strings AB, XYZ then

<new-set>::=<set-1>●●●

defines a new set consisting of the following strings:

AB XYZ

ABAB ABXYZ XYZAB XYZXYZ

ABABAB ABABXYZ ...

Further example:

<integer>::=<digit>●●●

The concept "integer" is defined in terms of the concept "digit", that is assumed to be known. The principle of construction is determined by the repetition-operator: A sequence of one or more digits is to be considered as integer, that is as a string of the set <integer>.

It is important to realize, that<digit>●●● consists of all digit strings, not of just strings of all zeros, all ones, etc.. Further it should be noted, that <digit> ●●● does not include the empty string.

The following 3 constructions are equivalent:

a) <integer>::=<digit>|<integer><digit>

b) <integer>::=<digit>|<digit><digit>|<digit><digit><digit>|etc..

c) <integer>::=<digit>●●●

It is obvious, that construction c) is most convenient for the reader. It is very close to the natural language ("any number of"; "a sequence of"). Construction b) is unconvenient because of the "etc", construction a) because of the recursivity.

It might be useful to note, that the unary repetition-operation, an operation or function with a single operand or argument is represented in a way slightly different from the one conventionally used in algebra.

In algebra, the unary operators + and - usually refer to the number following the operator, as for instance in -112 (pre-fixed operator).

The same method would be possible for the repetition-operation as well. However, for the convenience of the reader, the unary repetition-operator ●●● is defined to refer to the set preceding the operator, as for instance <set-1>●●● (post-fixed operator).

5.11 OPERATION OF PERMUTATIONS

Concatenated sets can be permutated and the permutations can be connected with the union-operator. This operation permits the construction of new sets in terms of sets already known.

The operation will be explained for permutations of two

sets and for permutations of three sets. Analogously, the operation can be defined for permutations of any number of sets.

If <a> and <b> are sets of strings then a new set

<p>::=⦃ <a>⫲<b> ⦄

is defined as being equivalent to

<p>::=<a><b>|<b><a>

If <a> <b> and <c> are sets of strings then a new set

<p>::=⦃ <a>⫲<b>⫲<c> ⦄

is defined as being equivalent to

<p>::=<a><b><c>|<a><c><b>|
<b><a><c>|<b><c><a>|
<c><a><b>|<c><b><a>

The meta-symbols ⦃,⦄ are called permutation-brackets, and the meta-symbol ⫲ is called permutation-separator.

It might be useful to note that

<p>::=⦃ [<a>]⫲<b>⫲<c> ⦄

includes 6 triples and 2 doubles <b><c> and <c><b>, that is, the set being defined is equivalent to

<p>::=<a><b><c>|<a><c><b>|
<b><a><c>|<b><c><a>|
<c><a><b>|<c><b><a>|
<b><c>|<c><b>

5.12 NESTED OPERATIONS

All the operations that have been defined for sets in the previous chapters are complete. That is any set of strings can be used as operand. The operation will always be defined and there will always be a unique set of strings that is the result of the operation.

Consequently, any result can be used as operand for another operation, and any level of nesting is possible.

For instance,

<a> can be concatenated with <b>, giving <a><b>. The

result can be combined by the union-operator with <c>, thus giving <a><b>|<c>.

It becomes immediately clear, that meta-brackets are required to define the sequence of the operations and to avoid ambiguities. The meta-symbols {and} will be used for that purpose.

Thus the combined operation mentioned above would have to be described by

{<a><b>} | <c>

The 3 operands could however have been combined as follows:

<a>{<b>|<c>}

If <a> consists of a single string AXA, <b> of the single string BYB, and <c> of the single string CZC, then it is obvious, that the two results are different from each other: The first result is a set consisting of the 2 strings AXABYB and CZC, the second result is a set consisting of the 2 strings AXABYB and AXACZC.

Meta-brackets can be avoided to some extent, by assigning priorities to the operators, which determine the sequence of operations, in case meta-brackets are not present.

For the meta-operators defined in the preceding chapters, the following sequence of priorities shall be valid (highest priority first, lowest priority last):

repetition-operator ●●●

concatenation-operator ‖ , abbreviated by concatenating operands.

union-operator |

containing-operator containing, abbreviated by "containing".

not-containing-operator not-containing, abbreviated by "not-containing".

not-identical-operator diff, abbreviated by "diff".

The following convention determines the sequence of operations if meta-brackets are not present:

1. If there is a conflict between unary and binary operations, the unary operation is executed first. Thus <a>|<b>●●● is equivalent to <a>|{<b>●●●} and <a><b>●●● is equivalent to <a>{<b>●●●}

2. If two binary operations have different priority, the operation with the higher priority is executed first. Thus <a>|<b><c> is equivalent to <a>|{<b><c>}.

3. If two binary operations have the same priority, the order of execution is from left to right.
   Thus <a><b><c> is equivalent to ⦃ <a><b>⦄<c>.

No priorities are assigned to the following operators:

option-operator
operators for permutation

Their operands are clearly located in the string of operators and operand-names. First the operands have to be evaluated, afterwards the indicated operation has to be executed. Because of the structure of the operator symbols no meta-brackets are required to enclose the operands.

Thus, [<a><b>] requires first the evaluation of <a><b>, that is the evaluation of the operand of the option-operation, and then adding the empty string to the set, that is the execution of the option-operation.

Similarly, ⦃ <a><b>|<x><y> ⦄ requires first the evaluation of <a><b> and of <x><y>, that is the evaluation of the operands of the permutation-operation, and then the execution of the permutation-operation.

That is

<result>::=⦃ <a><b>|<x><y> ⦄

is equivalent to

<result>::=⦃<a><b>⦄⦃<x><y>⦄| ⦃<x><y>⦄⦃<a><b>⦄

ENGLISH LANGUAGE EXTENSIONS OF THE METALANGUAGE

If required, the metalanguage described above is extended by the use of the English language.

SETS OF CONNECTED EXAMPLES

The following three sets of examples illustrate the utilization of the metalanguage.

Example 1

Formal Syntactic Definition of the Concept "Possibly-Signed-Integer"

Formal Definition	English Equivalent
<digit>::= 0|1|2|3|4|5|6|7|8|9	A digit is any one of the listed symbols.

<integer>::= <digit>●●●	An integer is a sequence of one or more digits.
<sign>::= +|-	A sign is either the symbol "+" or the symbol "-".
<possibly-signed-integer>::= [<sign>]<integer>	A possibly-signed-integer is an integer, optionally preceded by a sign.

Example 2

Formal Syntactic Definition of the Concept "Signed-Positive-Integer"

Formal Definition	English Equivalent
<digit>::= 0|1|2|3|4|5|6|7|8|9	A digit is any one of the listed symbols.
<non-zero-digit>::= <digit> diff 0	A non-zero-digit is a digit different from "0".
<integer>::= <digit>●●●	An integer is a sequence of one or more digits.
<positive-integer>::= <integer> containing<non-zero-digit>	A positive-integer is an integer that does contain a non-zero-digit.
<signed-positive-integer>::= +<positive-integer>	A signed-positive-integer is a positive-integer, preceded by the symbol "+".

Example 3

Formal Syntactic Definition of the Concept "Comment-Sentence"

Formal Definition	English Equivalent
.....	.....
<computer-character>::= <cobol-character>| <additional-data-character>	Acomputer-character is either a cobol-character or an additional-data-character.
<delimiting-period>::= . {<space>●●●}	A delimiting-period is a period followed by a sequence of one or more spaces.

<comment-string>::=
  {<computer-character>●●● }
  not containing
  <delimiting-period>
	A comment-string is a sequence of one or more computer-characters, that does not contain a delimiting-period.

<comment-entry>::= <comment-string> <delimiting-period>	A comment-entry is a comment-string followed by a delimiting-period.