CA-MetaCOBOL™ +

Macro Facility Reference

Release 1.1

R105M+11MRP

-- PROPRIETARY AND CONFIDENTIAL INFORMATION --

Release 1.1, January, 1992
Updated March, 1992

Contents

Figures

1. About This Manual

This manual is a reference for writing CA-MetaCOBOL+ macros. The most widely used facilities for the CA-MetaCOBOL+ macro processor are described in detail.

1.1 Purpose

The CA-MetaCOBOL+ *Macro Facility Reference* is the authoritative reference to the complete CA-MetaCOBOL+ language. This manual focuses on creating and testing macros and macro sets.

The CA-MetaCOBOL+ *User Guide* and *Macro Facility Tutorial* also provide useful information.

1.2 Organization

The Macro Facility Reference is organized as follows:

Chapter	Description
1	Discusses the purpose of the manual, gives a list of CA-MetaCOBOL+ documentation, and explains notation conventions for CA-MetaCOBOL+.
2	Describes the organization and contents of the CA-MetaCOBOL+ Translator.
3	Explains the general format of macros and describes user-defined macros, event-dependent macros, and macro nesting.
4	Gives a full explanation of symbolic words, including symbolic operands, variables, and concatenation.
5	Describes how to begin programming using the CA-MetaCOBOL+ Macro Language.
6	Discusses how to use the CA-MetaCOBOL+ Input Exit Facility.
Appendix A	Glossary. Defines common terms in CA-MetaCOBOL+.
Appendix B	Describes how to install and use the CA-MetaCOBOL+ Procedure Documentor, which aids in the debugging and maintenance of macro sets.
Appendix C	Provides instruction for debugging macro sets.
Appendix D	Describes older forms of CA-MetaCOBOL+ macros that are retained for compatibility but can be replaced by newer, more efficient forms.

1.3 Publications

In addition to this manual, the following publications are supplied with CA-MetaCOBOL+.

Title	Contents
Introduction to CA-MetaCOBOL+	Introduces the CA-MetaCOBOL+ Work Bench, Structured Programming Facility, Quality Assurance Facility, CA-DATACOM/DB Facility, Macro Facility, Panel Definition Facility, the Online Programming Language and the Chart Writer Language.
Installation Guide - MVS	Explains how to install CA-MetaCOBOL+ in the MVS environment.
CA-ACTIVATOR Installation	Explains how to install CA-MetaCOBOL+ in the

Supplement - MVS	MVS environment using CA-ACTIVATOR
Installation Guide - VSE	Explains how to install CA-MetaCOBOL+ in the VSE environment.
Installation Guide - CMS	Explains how to install CA-MetaCOBOL+ in the VM environment.

Title	**Contents**
User Guide	Explains how to customize, get started, and use CA-MetaCOBOL+. Includes information on keyword expansion, the CA-MetaCOBOL+ translator, and CA macro sets and programs.
Macro Facility Tutorial	Introduces the Macro Facility. Includes information on writing basic macros, model programming, macro writing techniques, and debugging.
Quality Assurance Guide	Introduces the Quality Assurance Facility. Includes all the necessary information to perform quality assurance testing on COBOL source programs.
Program Development Guide CA-DATACOM/DB	Includes all the information necessary to develop programs that make full use of the functions and features of the CA-DATACOM/DB environment.
Program Development Reference CA-DATACOM/DB	Contains all CA-DATACOM/DB Facility constructs and statements.
Panel Definition Facility Command Reference	Contains all Panel Definition Facility commands.
Panel Definition Facility User's Guide	Includes all the information necessary to create, edit, duplicate, rename, delete, index, and print panel definitions and members. Also describes how to generate BMS source.
Online Programming Language Reference	Contains all Online Programming Language statements.
Structured Programming Guide	Introduces the Structured Programming Facility. Includes information on creating, testing, and maintaining structured programs.
PC User Guide	Explains how to use CA-MetaCOBOL+/PC. Includes information on the CA-MetaCOBOL+ translator and CA macro sets and programs. Also describes the relationship between CA-MetaCOBOL+ and CA-MetaCOBOL+/PC.
Program Development Guide CA-DATACOM/PC	Describes how to develop programs that use the CA-DATACOM/PC environment.
String Manipulation Language Guide	Introduces the String Manipulation Language, which provides string handling and inspection capabilities unavailable in COBOL.

All manuals are updated as required. Instructions accompany each update package.

1.4 Notation Conventions

The following conventions are used in the command formats throughout this manual:

UPPERCASE BOLD	is used to display commands or keywords you must code exactly as shown.
lowercase italic	is used to display information you must supply. For example, DASD space parameters may appear as *xxxxxxx xxxxxxx xxxxxxx*.
Underscores	either show a default value in a screen image, or represents the highlighting of a word in a screen image.
Brackets [ ]	mean that you can select one of the items enclosed by the brackets; none of the enclosed items is required.
Braces {}	mean that you must select one of the items enclosed by the braces.
Vertical Bar |	separates options. One vertical bar separates two options, two vertical bars separate three options, and so on. You must select one of the options.
Ellipsis **. . .**	means that you can repeat the word or clause that immediately precedes the ellipsis.

1.5 Summary of Revisions

The following modifications and enhancements have been made to this manual for Version 1.1 of MetaCOBOL+.

The Section on Symbolic Operand Atributes (Section 4.3) has been revised and updated.

Information about the &SETR directive (Section 5.8.9) has been revised and updated.

The Glossary has been updated.

Minor technical and editorial revisions have been made throughout the manual.

2. Introduction

The CA-MetaCOBOL+ macro language is a flexible programming facility that can be used to customize the CA-MetaCOBOL+ Translator to solve a broad range of specific user problems.

2.1 CA-MetaCOBOL+ Translator Organization

An execution of the CA-MetaCOBOL+ Translator proceeds through three logical phases:

1. Initialization
2. Translation
3. Source Generation

The principal inputs, outputs, and support facilities of CA-MetaCOBOL+ are described in the CA-MetaCOBOL+ *User Guide*.

2.1.1 Initialization

Initialization prepares CA-MetaCOBOL+ for translation. The input, output, and work files are opened, translate-time options are established, and the macros are loaded into tables. Options may be specified on OPTION cards, or on the PARM list (MVS systems). Macros can be read from the primary input file and/or CA-LIBRARIAN and/or system source libraries.

The macros are loaded into three tables:

- Each macro-name is stored in the Symbol Table (TSY)
- Each macro model is stored in the Macro Table (TMS)
- Macro variables are stored in the Variable Table (TVA)

As the macros are loaded into the tables, they are listed on the INPUT Listing. Translator-directing statements (see the CA-MetaCOBOL+ *User Guide*) can be used to manipulate the macro listing, complete the specification of certain options, specify the source of input, define input format, establish macro regions, and control the macro tracing facility.

When a proper COBOL division header (IDENTIFICATION DIVISION, ENVIRONMENT DIVISION, DATA DIVISION, PROCEDURE DIVISION) or a CA-MetaCOBOL+ reserved macro-name for these headers ($ID, $ED, $DD, $PD) is read from the input, initialization is complete.

Figure 1 shows the flow of CA-MetaCOBOL+ Initialization:

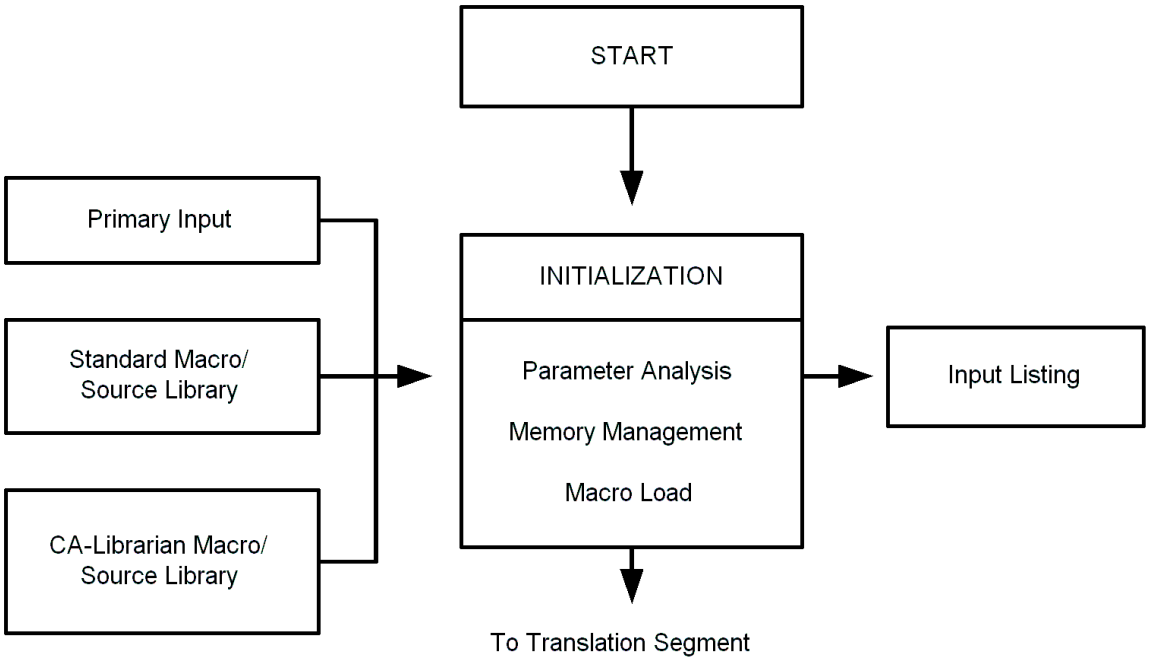

Figure 1. CA-MetaCOBOL+ Initialization

2.1.2 Translation

Translation depends on the interaction of COBOL source words and CA-MetaCOBOL+ macros. Input words matching macros, and certain COBOL language characteristics (division, level numbers, paragraph headers, verbs) determine where translation will take place. Actual translation is defined by rules within the macros. Macros determine how output will differ from input.

Translation of input from the primary input file, CA-LIBRARIAN, or system source libraries is the same. Each word of the COBOL input is processed separately. COBOL non-numeric literals which are "continued" (designated by a hyphen in the continuation column, column 7) are processed as if they were actually a single word. COBOL REMARKS, NOTEs, "debugging" statements (a D in column 7), and comments (an asterisk "*" or slash "/" in column 7) are also processed as if they were a single word even though they may contain several words.

Each COBOL division header initializes an internal code which permits individual macro translation to be restricted to a specific COBOL division, or divisions. During translation of DATA DIVISION input, all COBOL data-names (excluding FILLER) in the File, Working-Storage, Linkage, and Communication Sections are added to the Symbol Table (TSY). The attributes of each data definition (USAGE, PICTURE, etc.) are stored in the Data Structure Table (TDS).

COBOL COPY and CA-LIBRARIAN -INC statements are processed automatically during translation according to the option specified. If the COPY=ACTIVE or -INC option is specified, the COPY/-INC statement is replaced by the copied (included) source code, and the COPY/-INC statement is not available for macro processing.

If the COPY=PASSIVE, COPY=IGNORE, or -INC option is not specified, the COPY/-INC statement is retained, and the COPY/-INC statement is available for macro processing. In the DATA DIVISION, the COBOL COPY library (COPY=PASSIVE) or CA-LIBRARIAN master may be accessed and the specified source code retrieved in order to complete the Data Structure Table. However, the source code will not appear in the output.

Two intermediate work files are created by macro translation. Translated and untranslated COBOL source code, auxiliary output, and translation diagnostics (optionally) are written to these work files for merging by the Source Generation phase. The in-line work file contains all un-processed input words, and words inserted or replaced by macros at the original location of the input word. The out-of-line work file contains all input words and macro insertions or replacements directed by macros to locations in the generated source that are different from the location of the original input word.

When all input source has been processed, translation is completed. Figure 2 shows the flow of CA-MetaCOBOL+ translation:

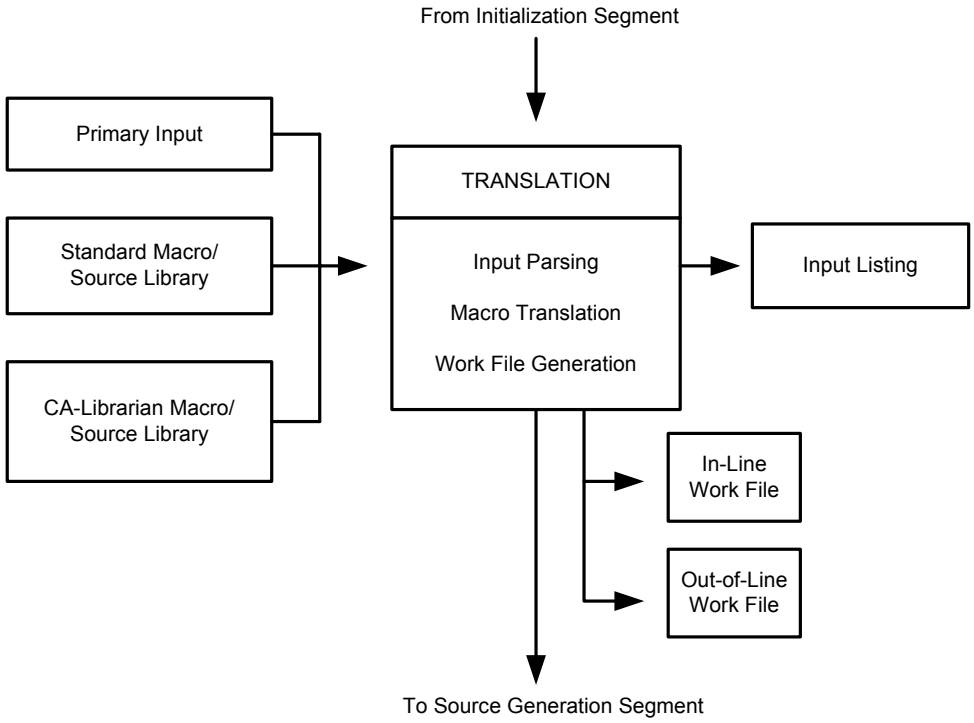

Figure 2. CA-MetaCOBOL+ Translation

2.1.3 Source Generation

Source Generation completes CA-MetaCOBOL+ processing. The in-line and out-of-line work files are merged to create formatted output for the OUTPUT listing and the COBOL file, output for the AUXILIARY file and listing, and output for the Accounting file.

Out-of-line work file text is merged into predefined and macro-defined merge locations in the in-line work file.

Segments of out-of-line work file text are merged with one another whenever macro defined merge locations are detected within the out-of-line work file. There are no predefined merge locations in the out-of-line work file.

A single out-of-line text segment may be merged into an unlimited number of macro defined merge locations within in-line and out-of-line work files. An unlimited number of different out-of-line text segments may be merged into one in-line work file location. Merged text sequence depends on the order in which macros creating out-of-line text segments are executed. The first text segment generated will be the initial segment merged into any given location.

The AUXILIARY file and listing are merged in the same manner as the COBOL statements except there are no predefined merge locations in these files.

Any translation output in the out-of-line work file which cannot be merged causes the LOST TEXT report to be prepared.

The merged COBOL statements are formatted according to options specified at installation, which may be modified at initialization, and further altered by macro processing.

When source generation is complete, CA-MetaCOBOL+ terminates with a predefined return code or a macro-defined return code which may be used to regulate the execution of subsequent job steps. Figure 3 is a block diagram of CA-MetaCOBOL+ Source Generation.

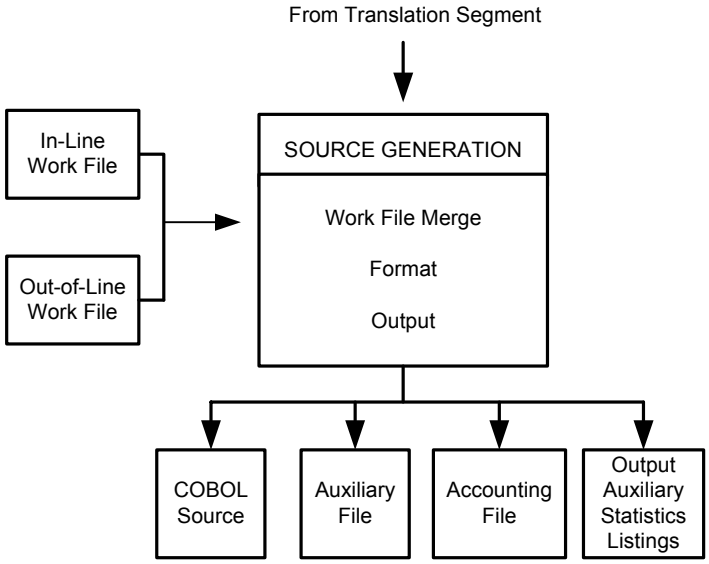

Figure 3. CA-MetaCOBOL+ Source Generation

2.2 Translation Tables

There are five tables used by CA-MetaCOBOL+ during the Initialization and Translation segments -- the Symbol Table (TSY), the Macro Table (TMS), the Variable Table (TVA), the Attribute Table (TDS), and the Stack Table.

2.2.1 Symbol Table (TSY)

The Symbol Table (TSY) is a dictionary of words and names that are important to the translation process. Initialization creates entries in the Symbol Table for CA-MetaCOBOL+ reserved words and CA-MetaCOBOL+ Word and String macro-names. Initialization also creates entries for COBOL reserved words, verbs, figurative constants, and special registers. However, the precise set of entries made for COBOL words depends upon the TARGET and DIALECT specified at installation, or specified as an option.

COBOL File, Working-Storage, Linkage, and Communication Section data-names in the primary input file, including source statements referenced but not actually copied or included by COBOL COPY and CA-LIBRARIAN -INC statements, are entered into the Symbol Table unless specifically excluded by macro programming. Data definitions created by macros may have entries in the Symbol Table depending on macro programming. In either case, the Symbol Table entry for a COBOL data item is not complete until the item has been completely processed by macros, or has completely bypassed macro processing. This is also true for group items with the difference that the entire group structure must be fully processed by macros or have competely bypassed macro processing before the group item entry is completed. The table update routines, which necessarily execute after macro processing, only recognize that an item is complete when a new item becomes available.

For data-names, Symbol Table entries include a pointer into TDS which permits the retrieval of data item attributes.

For Word and String macro-names, Symbol Table entries include a pointer into TMS which permits the Translator to retrieve the macro model for interpretation.

The other entries in the Symbol Table contain sufficient information to support lexical analysis of the input, primarily word length and codes identifying the word type.

Macro programming uses TSY to resolve symbolic operand attributes. However, when the macros do not include symbolic operand attributes, the Translator does not perform the necessary table updating.

2.2.2 Macro Table (TMS)

The Macro Table (TMS) contains the procedural portion (directives and constants) of macro models.

The procedural code (directives) in macro models is translated during initialization into a more compact language which conserves table space and simplifies interpretation of the macro model during translation. Individual directives are checked for correct syntax during initialization but are not checked for semantic accuracy.

As the macro model is loaded into TMS, references to variables (stored in the Variable Table; see 1.2.3) are resolved, and the variable's address is stored in the macro table. References to Tags and Labels are similarly resolved. In effect, these references are "compiled."

In many ways, the interpretation of a macro model is a macro process within a macro process. A source word matching a Word macro prototype in the Symbol Table causes the retrieval of a macro model for interpretation from the Macro Table.

2.2.3 Variable Table (TVA)

Initialization creates entries in the Variable Table (TVA) for all user defined variables. Memory is allocated to the variable according to their specification, and the variables are initialized to either default values or values provided by the programmer as part of the variable definitions. The Translator does not reinitialize these variables on later invocations of the same macro name during translation.

When initialization is complete, the structure of TVA is also complete. The table content will change continuously during translation as the values of variables change.

2.2.4 Attribute Table (TDS)

The Attribute Table (TDS), also known as the Data Structure Table, contains the data attributes (e.g., level number, usages, size, etc.) of all COBOL special registers and user-defined data items.

The Initialization segment primes the TDS with COBOL special register data attributes.

During translation of DATA DIVISION items, all data-names except FILLER are placed in TSY, and the corresponding data item attributes are placed in TDS. After they are in TDS and the current group item is computed, the attributes may be accessed. A group item and its attributes are complete when the next level number of the same or lower value has bypassed or been output by macro processing.

2.2.5 Stack Table

The Stack Table is both an internal work area used to store Prefix-type macro prototypes, and a comment stack for comments encountered during macro model processing.

During macro loading, Prefix macro prototypes are permanently placed at one end of the Stack Table. Macro models containing variable definitions and logical destinations, unique to the models, temporarily use the other end of the Stack Table. Here they resolve addresses between definitions and references as each model is placed in the Macro Table. Exceptionally complex macro models can overflow the Stack Table, resulting in immediate CA-MetaCOBOL+ termination. Such errors can often be avoided by loading Prefix macros after all those containing complex models or by increasing the size of the Stack Table during installation.

The Stack Table is also used during translation as a temporary word storage area for symbolic operands.

2.3 Standard Format

Standard COBOL format is used in CA-MetaCOBOL+ macro writing. The four divisions of a COBOL source program, and their functions are:

IDENTIFICATION DIVISION	names the program and, optionally, gives the date the program was written, the compilation date, and other information.
ENVIRONMENT DIVISION	describes the computer(s) to be used and specifies the machine equipment features used by the program. A description of the relationship of data files with input/output devices is included.
DATA DIVISION	defines the nature and characteristics of all data in the program. This includes both the data used in input/output operations and the data developed for internal processing.
PROCEDURE DIVISION	consists of executable statements that process the data in the manner defined by the programmer.

CA-MetaCOBOL+ reserved abbreviations corresponding to the preceding divisions must be placed in Area A of CA-MetaCOBOL+ input, and include:

$ID expands into the IDENTIFICATION DIVISION header.

$ED expands into the ENVIRONMENT DIVISION header.

$DD expands into the DATA DIVISION header.

$PD expands into the PROCEDURE DIVISION header.

These symbols, when encountered in Area A of the primary input will expand into their respective division headers. No other abbreviations can be defined for division headers. Users may use these symbols as macro prototypes and include other optional processing in the model portion of the macro.
Columns are:

Columns 1-6	represent the sequence number area.
Column 7	represents the continuation column.

Columns 8-11 represent Area A. COBOL elements that may begin in Area A are:

division headers
section headers
paragraph headers or paragraph-names
level indicator (FD, SD, RD, CD)
level numbers
key word DECLARATIVES
key words END DECLARATIVES

Columns 12-72 represent Area B.

Columns 73-80 are used to identify the program.

2.4 Comment Definition

"Comments" are used to add descriptive information to macros and are ignored by the translator.

Format 1:

```
[text] /* comment
```

/*

Comments are any words in an input record preceded by and including a /* and a space in Area A or Area B, and is not part of a literal. This provides a comment facility on lines with macro text.

In the second format shown below, an *, /, T, or X must be in the continuation column (column 7). Words beginning with *$ are Translator-directing statements (refer to the CA-MetaCOBOL+ *User Guide* for a complete description).

Format 2:

```
{ * }
{ / }
{ T } comment
{ X }
```

/

A slash (/) in the continuation column 7 also causes a page eject before the comment is printed on the INPUT & DIAGNOSTICS listing when the listing is invoked. Consecutive eject commands result in a single page eject.

T or **X**

In addition to all other commenting facilities, a T or X in the continuation column 7 causes the previous macro to end, and causes all other lines until the next macro definition or division header to be ignored. This comment can be used to temporarily delete a macro from a set while retaining the code. Note that external and global definitions within the macro are ignored.

When both forms of comments occur before the first division header, they are printed on the INPUT & DIAGNOSTICS Listing when the listing is invoked, but not passed to the Translator as input and are therefore never output.

Example:

```
/-------------------------------------------------*
*                                                 *
*                                                 *
*              PROCEDURE DIVISION                 *
*              AT TOP OF PAGE                     *
*                                                 *
*                                                 *
*-------------------------------------------------*

PROCEDURE DIVISION.
         .
         .
         .
```

3. Macro Definition

A CA-MetaCOBOL+ macro definition equates a "prototype" (the source code or conditions which might occur in a program) to a "model" (the rules which govern the processing of the source code). All macro definitions must be placed after the OPTION card(s) and before the first COBOL division header in the input stream.

3.1 General Format

Format:

```
t[c] macro-name [prototype-word-1...] [:]
[model-word-1 [...model-word-n]]
```

t

is the macro type code and must be entered as:

Code	Macro Type
P	Prefix
S	String
U	Un-verb
V	Verb
W	Word

The code must appear in the continuation column (column 7) and begin each macro definition.

c

is the division code and must be entered as a word in Area A (columns 8-11) consisting of:

Code	Division
I	Identification
E	Environment
D	Data
P	Procedure

or any combination of these four characters. If no division code is specified, the macro definition applies to all divisions.

The order in which the entries appear is not significant, but if more than one entry is specified, the codes must be contiguous (without any intervening characters and/or spaces). For example, ED is equivalent to DE, both meaning that the macro is only active for the ENVIRONMENT and DATA DIVISIONS.

prototype

is a word or sequence of words beginning in Area B (columns 12-72) and must be separated from the division code by one or more spaces. Its appearance in the source program causes a call to the macro definition. In case of duplicate prototypes, the last macro loaded by the Translator is the only macro processed.

Input stream words and word sequences cause a word correspondence search of the macro definitions of the currently active division in the following order:

1. Verb, Word, or String macros in the reverse order in which they are loaded.

2. Prefix macros in the reverse order in which they are loaded.

3. Un-verb macros are not considered in the search chain.

macro-name

is the first and/or only word of the prototype. The macro-name may consist of any COBOL keyword (except the keyword NOTE), data-name, or procedure-name; and it may be used as a macro-name, or prototype-word. Punctuation, non-numeric literals, numeric literals, picture character-strings, and pseudo-text character-strings are excluded from use as macro-names and prototype-words. The exceptions to these rules are:

* For Word and String Macro Macro-names

The following single character macro-names may be used: currency symbol ($); number symbol (#); at symbol (@); greater than symbol (>); less than symbol (<); cent symbol (¢); exclamation mark (!); percent symbol (%); and question mark (?). Multi-character macro-names may be composed of the following characters in combination with the preceding characters: Asterisk (*); stroke (/); numbers (0-9); hyphen (-); equals symbol (=); period, leading or imbedded, but not trailing (.); and plus symbol (+).

* For Prefix Macro Macro-names

In addition to a keyword or a partial keyword, data-name, or procedure-name, the following single character macro-names may be used: asterisk (*); stroke (/); currency symbol ($); number symbol (#); at symbol (@); greater than symbol (>); less than symbol (<); numbers (0-9); hyphen (-); equals (=); cent symbol ([); period (.); plus symbol (+); exclamation mark (!); semi-colon (;); comma (,); percent symbol (%); and question mark (?). Multi-character Prefix macro macro-names may be composed by using combinations of any of the preceding characters.

In addition there are eleven reserved macro-names available to the user:

Macro-name	Meaning
$ID	IDENTIFICATION DIVISION header
$ED	ENVIRONMENT DIVISION header
$DD	DATA DIVISION header
$PD	PROCEDURE DIVISION header (See Section 2.3.)
$-LEVEL	}
$-PROC	}
$DDE/$DDX	} (See Section 4.3.)
$PDE/$PDX	}
$-VERB	}

prototype-word-1...
are additional prototype words that are permitted in String-type (S) macros only.

In addition to any COBOL keywords, data-names, and procedure-names, single and multi-character prototype-words may be composed of the following characters: Asterisk (*); stroke (/); currency symbol ($); number symbol (#); at symbol (@); greater than symbol (>); less than symbol (<); numbers (0-9); hyphen (-); equals symbol (=); cent symbol ([); period (.); left parenthesis ((); plus symbol (+); exclamation (!); right prenthesis ()): Percent symbol (%); question mark (?).

The prototype must begin in Area B preceded by at least one space. Subsequent prototype-words can appear anywhere in Area B. Prototype-words cannot begin with an "&" unless it is a symbolic operand representing a source word or word sequence. Refer to Sections 3.2.4 and 3.2.5 for more information.

: (colon)
is a separator (colon). It defines the end of the prototype and the beginning of the macro model. Except for Verb and Un-verb macros, a colon terminates the prototype and the source program words causing the macro call to be removed from the input stream. A space must follow the colon.

model
represents the rules to be acted upon whenever a call is made to the macro. A model can consist of substitution words, variable definitions, directives, logical constructs, and logical destinations, all of which make up the action to be taken and/or the code to be generated when the macro prototype is called.

model-word-1, -n

are additional model-words. Those beginning with an "&" are CA-MetaCOBOL+ directives which cause the Translator to take specific, logical action. A model-word not beginning with an "&" is a constant and is used as it appears in the model. Any valid COBOL character string may be a constant.

A "directive" is a special word, embedded within a CA-MetaCOBOL+ macro model, which causes the Translator to take explicit action but does not appear in the output. A directive is similar in function to a COBOL verb which defines the action to be taken by a COBOL program. Directives always begin with an ampersand (&).

A "directive expression" is a series of words, beginning with a directive, which defines explicit Translator action. A directive expression is similar in function to a COBOL statement.

A "substitution word" is any symbolic or constant word in a macro model which is substituted directly or indirectly in the generated output, and serves as word storage areas, as work areas during composition, as switches, etc.

A substitution word appearing in Area A of the model is output in Area A. If the input source program word that matches macro-name is coded in Area A, the first model-word substituted is aligned at Area A, regardless of where the model-word is coded. Otherwise a substitution word coded in Area B is output to Area B in the generated program.

"Variables" are defined by the macro writer, or are predefined within the Translator.

Note: Any compiler-directing statement (i.e., EJECT, slash "/", SKIP1, SKIP2, and SKIP3) specified within the macro division advances the INPUT & DIAGNOSTICS listing as appropriate, but does not appear in the macro generated output.

Example: The following example shows several simple macro definitions and how they are used in a program. All model words are constant substitution words in this example. No directives, etc., are shown.

CA-MetaCOBOL+ Input

```
 WD   RMF : RECORDING MODE IS F
 WD   LRS : LABEL RECORD IS STANDARD
 WD   DR : DATA RECORD IS

FD  FILEA RMF LRS DR RECA.
01  RECA.
    .
    .
    .
FD  FILEB RMF LRS DR RECB.
01  RECB.
    .
    .
    .
```

CA-MetaCOBOL+ Output

```
FD  FILEA  RECORDING MODE IS F
           LABEL RECORD IS STANDARD
           DATA RECORD IS RECA.
01  RECA.
    .
    .
    .
FD  FILEB  RECORDING MODE IS F
           LABEL RECORD IS STANDARD
           DATA RECORD IS RECB.
01  RECB.
    .
    .
    .
```

3.2 User Defined Macros

User-defined macros consist of Verb, Un-verb, Word, Prefix, and String-type macros.

3.2.1 Verb Macro Definition

The Verb macro is used to define a word as a verb.

Format:

```
V[c] macro-name
```

V

is the macro type code for Verb. V appears in the continuation column (column 7). A call to a Verb macro does not cause replacement of the word causing the call. (See the CA-MetaCOBOL+ *User Guide* for the list of CA-MetaCOBOL+ reserved words.)

Verb definitions must appear ahead of all other macro definitions. As with all other macro definitions, those for the Verb-type must appear before the first input division header.

c

is the division code which may be I, E, D, P, or any combination of these four characters, or blank (all divisions).

macro-name

represents any string of 30 characters or less which is not a literal or a single special character (see Section 3.1 for more information).

Note: There is no model for this macro type.

Examples: PLAY is verbed as follows.

```
V     PLAY
```

The translator will now regard PLAY as a verb like all other COBOL verbs.

3.2.2 Un-verb Macro Definition

The Un-verb macro is used to delete a word from the list of words defined as verbs or to cancel a previously loaded Verb macro.

Format:

```
U[c] macro-name
```

U

is the macro type code for Un-verb. U appears in the continuation column (column 7). A call to an Un-verb macro does not cause replacement of the word causing the call. (See the CA-MetaCOBOL+ *User Guide* for the list of CA-MetaCOBOL+ reserved words.)

c

is the division code which may be I, E, D, P, or any combination of these four characters, or blank (all divisions).

macro-name

represents any string of 30 characters or less which is not a literal or a single special character (see Section 3.1 for more information).

Note: There is no model for this macro type.

Example: MOVE is un-verbed as follows.

```
U       MOVE
```

The translator will no longer regard MOVE as a COBOL verb.

3.2.3 Word Macro Definition

The Word macro substitutes zero, one, or more words for a single source or substitution word.

Format:

```
W[c]  macro-name : [model-word-1 [...model-word-n]]
```

W

is the macro type code for Word. W appears in the continuation column (column 7).

c

is the division code which may be I, E, D, P, or any combination of these four characters, or blank (all divisions).

macro-name

represents any string of 30 characters or less which is not a literal or a single special character (see Section 3.1 for more information).

:

is the separator (required).

model-word-1, -n

represents the rules to be acted upon whenever a call is made to the macro (see Section 3.1 for more information).

Example:

```
WD   -WS :
 WORKING-STORAGE SECTION.
```

In this example, only the DATA DIVISION of the source program is searched for the word "-WS". Each time "-WS" is found, this macro is called, and the words "WORKING-STORAGE SECTION." are output in its place.

Since "WORKING-STORAGE SECTION." is coded in Area A in the model, it is always output in Area A in the CA-MetaCOBOL+ output file, beginning a new line, regardless of the location of the "-WS" in the DATA DIVISION of the input source program.

Even if the word "-WS" in the source is followed by a period, only one terminating period appears in CA-MetaCOBOL+ output (i.e., CA-MetaCOBOL+ suppresses consecutive terminating periods).

3.2.4 Prefix Macro Definition

The Prefix macro substitutes zero, one, or more words for a single source substitution word and uses the last portion of that source substitution word as variable data.

Format:

```
                              [&          ]      [&             ]
      P[c]  macro-name[&] : [model-word] ... [model-word-n]...
```

P

is the macro type code for Prefix. P appears in the continuation column (column 7).

c

is the division code which may be I, E, D, P, or any combination of these four characters, or blank (all divisions).

macro-name[&]

represents any string of 30 characters or less which is not a literal or a single special character (see Section 3.1 for more information).

This macro prototype is actually the first few characters of a word (the prefix). The "&" (suffix) will match any character following the prefix.

: (colon)

is the separator (required).

```
[&          ]      [&             ]
[model-word] ... [model-word-n]...
```

model word

represents the rules to be acted upon whenever a call is made to the macro (see Section 3.1 for more information). The symbolic operand "&" may occur any number of times within the model or may not appear in the model at all.

Example: In the following example, the prototype can be specified as "O-" or "O-&". The optional use of the "&" in the prototype identifies the suffix symbolically for clarity only.

CA-MetaCOBOL+ Input

```
      .
      .
      .
PP    O- :  & OF OLD-MASTER
      .
      .
      .
 PROCEDURE DIVISION
      .
      .
      .
      MOVE O-PART TO PRINT-PART
```

CA-MetaCOBOL+ Output

```
     .
     .
     .
    MOVE PART OF OLD-MASTER TO PRINT-PART
```

3.2.5 String Macro Definition

The String macro replaces the original source word(s) you specify with the variable data you specify. The original source may be zero, one, or more words. The variable data may also be zero, one, or more substitution words. You may specify a one-to-one correspondence between the number of original source words and the number of substitution words; however, you are not required to do so.

For example, you could specify that a specific source word will be replaced by one substitution word. Alternatively, you could specify that a specific source word will be replaced by a string of substitution words. Conversely, you could choose to replace a string of original source words with one substitution word.

The macro name must always be the first word of the prototype. The prototype of the String macro is the only macro prototype that may consist of more than one word.

Format:

```
S[c]  macro-name[&]  [ &n                ]       [ &n          ]
                     [ prototype-word]... :  [ model-word]...
```

S
is the macro type code for String. S appears in the continuation column (column 7).

c
is the division type code which is I, E, D, P, or any combination of these four characters, or blank (all divisions).

macro-name
represents any string of 30 characters or less which is not a literal or a single special character (see Section 3.1 for more information). Macro-name cannot be specified as a symbolic operand.

&n
prototype-word
are additional prototype words that are permitted in String-type macros only and can be specified as constants and/or in symbolic operands. A maximum of 15 symbolic operands and any number of constant word sequences can be specified.

"Symbolic operands" are represented by "&n", where "n" represents any integer 1-15 (i.e., &1 to &15).

The actual word or sequence of words represented by a symbolic operand is defined by the location of the symbolic operand in the macro prototype. A call to a String macro is made whenever a word string or condition in the input stream matches (by location) the macro-name and all prototype words and symbolic operands described in the macro prototype, including all type restrictions placed on each symbolic operand. Therefore, each symbolic operand must be uniquely defined (a different integer for each) in the prototype.

"Recognition codes" qualify the types of words which can be accepted from the input stream as the current value of a symbolic operand. These codes are specified only in the prototype as follows:

&n(modifier)

The &n symbolic operand modifier in the prototype allows a single operand to represent an item other than a single source word. It also allows the inclusion of qualifiers and subscripts/indices within the source word sequence, thus allowing the current value of a String symbolic operand to contain more than one word. The modifier also restricts the input source which qualifies for prototype matching, and thus restricts String macro calls to more rigidly defined word sequences.

The modifier is optional, but, when used, must always appear within parentheses. Valid modifiers include:

(**Q**) A single data-name or procedure-name plus any COBOL qualifying phrases, if present (OF ..., IN ...).

Example: COMPUTE-NEW-BALANCE
COMPUTE-NEW-BALANCE OF SECTION-1
COMPUTE-NEW-BALANCE IN SECTION-1

(**S**) Identical to Q plus any COBOL subscripts/indices, if present.

Example: AMOUNT
AMOUNT OF MASTER-RECORD
AMOUNT OF MASTER-RECORD (SUB1)

(**L**) A literal or figurative constant only.

Example: 'TOTAL'
+50
SPACES

(**R**) A data-name that may be qualified, subscripted, indexed, and reference modified.

Example: NAME
NAME OF ADDR-RECORD (SUB1)
NAME OF ADDR-RECORD (SUB1) (4:20)

(**Q,L**)
A word or word sequence that satisfies either Q or L.

Example: COMPUTE-NEW-BALANCE
COMPUTE-NEW-BALANCE OF SECTION-1
5000

(**S,L**)
A word or word sequence that satisfies either S or L; in effect, Q, S, L.

Example: AMOUNT
AMOUNT OF MASTER-FILE (SUB1)
'TOTAL'
+50
SPACES

(**R,L**)
A data-name that may be qualified, subscripted, indexed, and reference modified, or it may be a literal.

Example: NAME
NAME OF ADDR-RECORD (SUB1) (10:15)
`UNDEFINED`

The absence of any recognition code indicates that only the next single word in the input stream, whatever it might be, is used for the symbolic operand.

The codes Q, R, and S cannot be specified if the anticipated source word represents a COBOL reserved word, since COBOL reserved words cannot be qualified, subscripted, or indexed.

: (colon)
is the separator (required).

&n
model-word
represents the rules to be acted upon whenever a call is made to the macro (see Section 3.1 for more information).

CA-MetaCOBOL+ substitutes the current value of each symbolic operand for each occurrence of the symbolic operand in the model of the macro definition. The same symbolic operand may appear any number of times in the model or not at all.

Note: COBOL comments intervening between source words as they are matched to a String macro prototype have no effect upon the macro calling function. Such comments are passed through the Translator immediately, and will appear in the generated output immediately before code generated by the macro model.

A String macro can be called from the input stream only, never from the substituted words of another macro.

String macros can only be called by COPY or -INC text if the COPY or -INC translate-time option is specified.

Examples: In the following example, the prototype, "&1" symbolically represents the first word following "PACKED" (in this case, "S999") and "&2" represents the second word following "PACKED" (in this case, "ZERO"). The call is made to the macro only because of the "PACKED" word match; CA-MetaCOBOL+ uses the source program words to replace the proper model symbolic operand in the output.

CA-MetaCOBOL+ Input

```
      .
      .
      .
SD    PACKED &1 &2 : PIC &1 COMP-3 VALUE &2
      .
      .
      .
 DATA DIVISION.
      .
      .
      .
      02 NAME PACKED S999 ZERO.
```

CA-MetaCOBOL+ Output

```
       .
       .
       .
      02 NAME PIC S999 COMP-3 VALUE ZERO.
```

In the following example, &1 is defined as the first word after "-PRINT" (i.e., "A-LINE"); &2 is defined as the first word after "FROM" plus any qualifying phrases and/or subscripts/indices "DIAGNOSTIC OF ERROR-TABLE (SUB1)"; and &3 is defined as the literal after the last word of &2. The call is made to the macro definition because of the "-PRINT' match, the "FROM" match, and the literal match for &3. Note that the proper word sequences from the source program appear as replacements for the corresponding symbolic operands in the model.

CA-MetaCOBOL+ Input

```
SP   PRINT &1 FROM &2(S) &3(L) :
         WRITE &1 FROM &2 AFTER ADVANCING &3 LINES.
         MOVE SPACES TO &1.
     .
     .
     .
 PROCEDURE DIVISION.
     .
     .
     .
     PRINT A-LINE FROM DIAGNOSTIC OF ERROR-TABLE
(SUB1) 1.
```

CA-MetaCOBOL+ Output

```
     WRITE A-LINE FROM
        DIAGNOSTIC OF ERROR-TABLE (SUB1)
        AFTER ADVANCING 1 LINES.
     MOVE SPACES TO A-LINE.
```

Since &1 in the above example is not coded in the prototype to accept qualifications, the lines

```
PRINT A-LINE OF FILE-A
   FROM DIAGNOSTIC OF ERROR-TABLE (SUB1) 1
```

would not match the prototype and therefore no call would be made to this macro.

3.3 Event Dependent Macro Prototypes

Certain macro prototypes are matched to special conditions or events in the source input rather than to specific words.

The recognition of the following events in the source input can cause macro calls:

- The beginning of a data item definition in the DATA DIVISION ($-LEVEL).
- A procedure header in the PROCEDURE DIVISION ($-PROC).
- The end of the DATA DIVISION ($DDE, $DDX).
- The end of source input program ($PDE, $PDX).
- Any word, within the appropriate division, defined as a verb ($-VERB).

3.3.1 $-LEVEL

The $-LEVEL causes a macro call before each DATA DIVISION level number (preceded by a period on the last statement) recognized in the source input program.

Format:

```
SD  $-LEVEL :
```

S
is the type code for String macro. This prototype must be a String-type, and does not attempt to match any words in the source input.

D
is the division code for DATA DIVISION macro.

$-LEVEL
is a macro prototype which should not appear as a word in the source input program.

No additional words can appear in the prototype.

:
is the separator (required).

Note: $-LEVEL can be used to change level numbers to a consistent installation standard, to rearrange the clauses in an item description to a standard sequence, to ensure a VALUE clause in all Working-Storage items for initialization, etc.

"Level number" is defined as a numeric literal in the DATA DIVISION immediately following a period or immediately following, consecutively, a period and an Area A indicator.

No words, including the level number, are acquired by the macro call. It is the responsibility of the user to &GET any or all words in the item definition, beginning with the level number.

When using &GET/&STORE/&STOW logic in an $-LEVEL macro, the period terminating a data item definition must not be removed, or the subsequent data item level number does not cause a call upon the $-LEVEL macro and Translator analysis of data attributes is incorrect.

Caution must be exercised in the use of this macro in conjunction with other macros, since it overrides or is overridden by another $-LEVEL macro, possibly causing improper translation.

Example: In the following example, level-77 items in the DATA DIVISION are changed to level-01. Note that 77 is not valid as a macro-name since it contains only numeric characters. (It would be difficult to distinguish between the level number 77 and a numeric literal if 77 were valid as a macro-name.)

```
SD   $-LEVEL :
     &GET &1                    /* INSPECT NEXT WORD
     &IF &1 = 77                /* 77?
         &STORE                 /* REMOVE 77 FROM INPUT
         01                     /* OUTPUT 01
     &ENDIF
```

3.3.2 $-PROC

The $-PROC causes a macro call before each PROCEDURE DIVISION paragraph/section header is recognized in the source input program.

Format:

```
SP  $-PROC :
```

S

is the type code for String macro. This prototype must be a String-type, and does not attempt to match any words in the source input.

P

is the division code for PROCEDURE DIVISION macro.

$-PROC

is a macro prototype which should not appear as a word in the source input program.

No additional words can appear in the prototype.

:

is the separator (required).

Note: This macro can be used to prefix procedure headers with sequential numbers, to insert spacing commands ahead of procedures, to insert statements at the beginning of paragraphs, to count paragraphs, to place EJECT commands ahead of sections, to audit the PROCEDURE DIVISION for proper Area A usage, etc.

"Procedure header" is defined as one or more words in the PROCEDURE DIVISION preceded by an Area A indicator.

No words of the procedure header are acquired by the macro call. It is the responsibility of the user to &GET any or all words in the procedure header beginning with the word following the Area A indicator (the Area A indicator itself has already been processed and is not available to this macro call).

The DECLARATIVES and END DECLARATIVES headers trigger a $-PROC macro call.

Caution must be exercised in the use of this macro in conjunction with other macros, since it overrides or is overridden by another $-PROC macro, causing improper translation.

Example: In the following, an EJECT card is output ahead of each procedure header.

```
SP   $-PROC :
     &GET &1                          /* INSPECT NEXT WORD
                                      /* BYPASS
DECLARATIVES
     &IF &1 NE 'END' &AND &1 NE 'DECLARATIVES'
                                      /* OUTPUT EJECT
L           EJECT
     &A                               /* OUTPUT AREA A
INDICATOR
     &ENDIF
```

3.3.3 $DDE/$DDX

$DDE and $DDX cause a macro call at the end of the DATA DIVISION when the PROCEDURE DIVISION header is recognized in the source input program but before it is processed.

Format:

```
SD  {$DDE}
    {$DDX} :
```

S

is the type code for String macro. These prototypes must be String-type, and do not attempt to match any words in the source input.

D

is the division code for DATA DIVISION macro.

$DDE
$DDX

are macro prototypes which should not appear as words in the source input program.

If both are used, $DDE is called before $DDX, regardless of the sequence in which they are coded in the macro set. $DDX is called after any macro call to $DDE is complete.

A maximum of 9 (the first 9 loaded) $DDX macros are called by the Translator in the sequence in which they are loaded. Only the last $DDE loaded is called.

No additional words can appear in the prototype.

:

is the separator (required).

Notes: These macro prototypes are used to finish processing the DATA DIVISION. They can be used to create one or more section headers that may not have been coded in the input source program, to generate data definitions, to guarantee in-line placement of DATA DIVISION entries, etc.

No words are acquired by the macro call, including the Area A indicator preceding the PROCEDURE DIVISION header. If the user codes an &GET in the model to acquire additional words, these words are acquired as PROCEDURE DIVISION text and may trigger calls to PROCEDURE DIVISION macros but not DATA DIVISION macros.

This macro should not be used to initialize processing for the PROCEDURE DIVISION.

Example: The following example shows a series of macros used to generate a Working-Storage Section header if one is not already in the DATA DIVISION.

```
SD   WORKING-STORAGE SECTION. :
         &DO &LWS-HEADER
SD   LINKAGE SECTION. :
         &DO &LWS-HEADER
 LINKAGE SECTION.
SD   REPORT SECTION. :
         &DO &LWS-HEADER
 REPORT SECTION.
SD   COMMUNICATION SECTION. :
         &DO &LWS-HEADER
 COMMUNICATION SECTION.
SD   $DDE :
         &DO &LWS-HEADER
         &GOBACK
     &LWS-HEADER
         &GLOBAL &BWS-HEADER = FALSE
         &IF NOT &BWS-HEADER
             &SET &BWS-HEADER = TRUE
             &A
             WORKING-STORAGE SECTION.
     &ENDIF
     &EXIT
```

3.3.4 $PDE/$PDX

$PDE and $PDX cause a macro call at the end of source input during translation.

Format:

```
S[P]  {$PDE}
      {$PDX} :
```

S is the type code for String macro. These prototypes must be String-type, and do not attempt to match any words in the source input.

P is the division code for PROCEDURE DIVISION macro. If the division code P is specified and there is no PROCEDURE DIVISION, this macro is not called.

$PDE
$PDX are macro prototypes which should not appear as words in the source input program.

If both are used, $PDE is called before $PDX, regardless of the sequence in which they are coded in the macro set. $PDX is called after any macro call to $PDE is complete.

A maximum of 9 (the first 9 loaded) $PDX macros are called by the Translator in the sequence in which they are loaded. Only the last $PDE loaded is called.

No additional words can appear in the prototype.
sss
: (colon)
is the separator (required).

Note: These macros are used to finish processing the source program. They can be used to map the DATA DIVISION, to add code to the DATA or PROCEDURE DIVISION as dictated by requirements of the PROCEDURE DIVISION, to end acquisition of source words, to summarize translation results, to perform complex reporting to the AUXILIARY listing, etc.

No words are acquired by the macro call. &GET should not be coded in the model as this may cause CA-MetaCOBOL+ to try to read beyond the end of the input file.

Example: The following creates a record on the Accounting File to indicate the use of 'MACROSET', the name of the macro set including this macro.

```
S $PDE :
       &ACCT 'MACROSET'
```

3.3.5 $-VERB

The $-VERB causes a macro call every time a word with a verb type attribute (&n'T = 'V', including a terminating period) is recognized in the source input program.

Format:

```
S[C]  $-VERB :
```

S
is the type code for String macro. This prototype must be a String type, and does not attempt to match any words in the source input.

c
is the division code which may be I, E, D, P, or any combination of these four characters, or blank (all divisions).

$-VERB
is a macro prototype which should not appear as a word in the source input program. The Type attribute of a word qualifying as a verb (&n'T = 'V') is not affected by the $-VERB macro. (See Section 4.3 on Symbolic Operand Attributes.)

No additional words can appear in the prototype.

: (colon)
is the separator (required).

Notes: These macros are used to enhance normal CA-MetaCOBOL+ formatting, to count statements, to insert commands, etc.

No words are acquired by the macro call. It is the responsibility of the user to &GET the verb or period.

Any COBOL verb also defined as a macro-name does not invoke this macro.

Caution must be exercised in the use of this macro in conjunction with other macros, since it overrides or is overridden by another $-VERB macro, causing improper translation.

Example: The following processes all PROCEDURE DIVISION verbs, except periods, that are not processed by explicit macro calls.

```
SP   $-VERB :
&GET &1
&IF &1 NOT = '.'
   &STORE
      .
      .
      .
&ENDIF
```

3.4 Macro Nesting

Each word substituted by a called macro is searched via the macro definitions for Prefix and Word macros in the same order in which a source input word is searched but with the calling macro-name removed from the list for this "nest level." Any call to a String macro is ignored. If a match is made, an additional macro call is initiated. This entire process is called macro nesting.

As many as nine levels of macro nesting per source word are possible. Specifically, the rules for nesting are:

- Each qualifying word substituted from a Word macro model can invoke a single Prefix macro and a maximum of 8 additional levels of Word macros.
- Each qualifying word substituted from a Prefix macro model can invoke a maximum of 8 levels of Word macros.
- Each qualifying word substituted from a String macro model can invoke a single Prefix macro and a maximum of 9 levels of Word macros.

If these rules are violated, a CA-MetaCOBOL+ diagnostic is produced, the search for further nested macros is terminated, and the word which failed to "nest" is output.

Instead of repeating identical macro code in each of several macros, it is often convenient to have these macros substitute a special word (a word which cannot logically appear in the input stream) which acts as a call to a Word macro. This Word macro contains the "common" macro code. When used in this manner, Word macros can be thought of as closed-loop subroutines.

If a Word or Prefix macro is to act on a word substituted from another macro directed explicitly out-of-line to another division, the nested macro must have a division code for the other division. Out-of-line modes are not changed by calling such a Word or Prefix macro unless done so explicitly.

At the completion of the translation of a nested macro, control is returned to the next higher macro in the nesting hierarchy.

Example: "DO" causes a call to the Word macro "DO", and is translated to "PERFORM". "DOTHRU" causes a call to the String macro "DOTHRU". "BETA" and "GAMMA" become the current value of the symbolic operands &1 and &2, respectively. The first word output by "DOTHRU" ("DO") causes a nested call to the Word macro "DO", which outputs "PERFORM" and returns to the "DOTHRU" macro, which then outputs "BETA THRU GAMMA".

CA-MetaCOBOL+ Input

```
WP   DO : PERFORM
     .
     .
     .
SP   DOTHRU &1 &2 : DO &1 THRU &2
     .
     .
     .
 PROCEDURE DIVISION.
     .
     .
     .
     DO ALPHA. DOTHRU BETA GAMMA.
```

CA-MetaCOBOL+ Output

```
     .
     .
     .
 PROCEDURE DIVISION.
     .
     .
     .
     PERFORM ALPHA.
     PERFORM BETA THRU GAMMA.
```

4. Symbolic Words

"Symbolic words" are the notation used in CA-MetaCOBOL+ macros to represent variable information. During the execution of a macro call, the current values of symbolic words are determined and used in place of the notation. There are four classes of symbolic words:

1. Symbolic operands, which generally represent one or more words acquired from the source input (see Section 4.1).

2. Variables, which are defined in macro models by the macro writer or which are predefined within the Translator (see Section 4.2).

3. Symbolic Operand Attributes, which represent a characteristic of the current value of a symbolic operand (see Section 4.3).

4. Concatenation, which is the combination of one or more words into a new word (see Section 4.4).

4.1 Symbolic Operands

"Symbolic operands" are defined within the Translator. They are a means of directly introducing variable source program words into macro models. There are three types of symbolic operands.

4.1.1 &

Format:

```
&
```

& is a symbolic operand that can obtain a value only by the call of a Prefix-type macro, and can appear any number of times in the model. It assumes the value of the remainder (suffix) of the source word that causes the call to a Prefix macro.

Example: The following macro is used to generate RECORD CONTAINS n CHARACTERS clauses.

```
PD   RCC=& : RECORD CONTAINS & CHARACTERS
```

4.1.2 &n

Format:

```
&n
```

&n is a symbolic operand that may represent fifteen symbolic operands (&1, &2, ..., &15).

An &n symbolic operand can be used in the prototype of a String-type macro and in the model of any macro type. A specific &n symbolic operand can appear only once in a prototype, but can appear any number of times in a model.

If used in a String-type macro prototype, an &n symbolic operand can represent a single word in the source program input, a qualified and/or subscripted word, or a literal (see Section 3.2.5 for more information).

If used in a macro model, an &n symbolic operand often represents the value that an &n symbolic operand acquired in the corresponding String macro prototype.

An &n symbolic operand in any macro model can also be set to the value of another symbolic operand, variable, word, or series of words. Each has an implicit maximum length of 510 characters.

Example: The following macro is used to convert a 1968 ANSI COBOL EXAMINE statement to an equivalent 1974 ANSI COBOL INSPECT statement.

```
SP   EXAMINE &1(S) REPLACING UNTIL FIRST &2(L) BY &3(L) :
     INSPECT &1 REPLACING CHARACTERS BY &3 BEFORE INITIAL
&2
```

4.1.3 &0

Format:

```
&0
```

&0 is a symbolic operand that can only be used in the model of any macro definition and can appear any number of times in a model. Its value becomes the name of an identifier through the action of &SCAN-type directives, and can be referenced only while the function of one of these directives remains unterminated.

Example: Assuming that &1 contains the name of a level-88 condition-name, the following macro code will place the name of the associated data item in &0.

```
.
.
.
&IF &1'L = '88'
     &SCANF &1
     &NOTE &( 'CONDITION-NAME ' &1 &)
     &NOTE &( 'IS ASSOCIATED WITH ' &0 &)
&ENDIF
.
.
.
```

4.2 Variables

"Variables" are defined by the macro writer, and serve as word storage areas, as work areas during computations, as switches, and for many other purposes. Variables may be "external" (known to all macros in all regions), "global" (known to all macros in the region), or "local" (known only to the macro in which defined).

If duplicate definitions are provided for the same variable name (i.e., have the same scope, class, and maximum size), CA-MetaCOBOL+ uses the first definition only. Note that this use is different from CA-MetaCOBOL+ using the last duplicate macro definition.

A variable must be defined via an &EXTERN, &GLOBAL, or &LOCAL directive expression prior to any reference.

4.2.1 Non-numeric, Numeric, and Symbolic Operand Variables

"Non-numeric and numeric variables" can be used as word storage areas, as work areas for computations, and as switches to control macro logic.

"Symbolic operand variables" can be used to store the current value of a symbolic operand.

Format:

```
{&EXTERN}
{&GLOBAL} &Vname[(integer)] [ = literal ] {picture}
{&LOCAL }                   [...NULL    ] {S      }
```

&EXTERN
defines an "external variable" which can be referenced by all macros in all regions. (For a complete description of regions, see the Glossary in Appendix A.)

&GLOBAL
defines a "global variable" which can be referenced by all macros in a single region. A global variable may have the same name as an external variable. In that case, references to the variable subsequent to the global definition and within the region containing this definition always refer to the global variable rather than the external variable.

&LOCAL

defines a "local variable" which can be referenced only in the macro in which it is defined. A local variable may have the same name as an external or global variable. In that case, references to the variable subsequent to the local definition and within the macro containing this definition always refer to the local variable rather than the external or global variable.

&Vname

is the name of a non-numeric, numeric, or symbolic operand storage variable. Maximum length is 30 characters.

integer

defines multiple occurrences of a non-numeric, numeric, or symbolic operand storage variable by attaching an occurrence number integer in parentheses without intervening spaces. Each reference to a multiple symbolic operand storage variable must indicate the occurrence number requested; otherwise, CA-MetaCOBOL+ assumes a default of the first occurrence.

literal
NULL

initializes every occurrence of a variable by the value literal. The literal must be valid for the class of variable it is to initialize.

NULL can be specified only for non-numeric variables and represents "no value" (&Vname is empty). Other directives can set a variable to, or test for, the NULL state. A variable with a NULL value which is output takes no space in the output.

Unless otherwise specified, the initial value of a numeric variable is a single 0; that of an non-numeric variable is a single blank (hex 40); that of a symbolic operand storage variable is NULL. The values repeat for all occurrences.

picture

indicates the nature of data and size of field.

For non-numeric variables, the picture must be specified as X[...X] or X[(n)], as for a non-numeric data item definition in COBOL. No more than 128 characters can be defined and stored.

For numeric variables, the picture must be specified as 9[...9] or 9[(n)], as for a numeric data item definition in COBOL. Only integer picture definitions are allowed, the sign is implicit. No more than 9 digits can be defined and stored, exclusive of high-order zeros.

If a defined variable name appears outside a directive expression, the current value of the variable becomes a substitution word in the output.

A non-numeric or numeric variable contains only a single word. If multiple words are placed in a variable by concatenation, the integrity of the separate words is not maintained. If a multi-word symbolic operand is &SET into a variable, the non-numeric or numeric variable receives only the first word.

The size of a variable changes to fit the single word stored in the variable from no characters (NULL) to the maximum size as defined by the picture. Leading zeros are suppressed in numeric variables, trailing spaces are suppressed in non-numeric variables. If an attempt is made to store more characters in a variable than the maximum defined size will allow, excess characters to the right are truncated.

S

is a symbolic operand storage variable (a variable defined with a picture S) that reserves an area large enough to hold all the words in a symbolic operand. The integrity of the separate words is maintained.

A symbolic operand storage variable may be used only as the sending item-2 in an &EQU or the receiving item (item-1) in a Format 1 &SET directive expression. It cannot be referenced outside of these directive expressions.

Examples: In the following, each of the three occurrences of &VSUM will have an initial value of the character 'A'.

```
&LOCAL &VSUM(3) = 'A' X
```

Each reference to a multiple variable must indicate the occurrence number requested; otherwise, CA-MetaCOBOL+ assumes a default of the first occurrence. Below, the three characters "ABC" are placed into the second occurrence of &VNAME.

```
&LOCAL &VNAME(3) X(10)
&LOCAL &VINDEX 9
&SET &VINDEX = 2
&SET &VNAME(&VINDEX) = 'ABC'
```

Below, assume &1 contains A OF B, then

```
&LOCAL &V-OPER S
    .
    .
    .
&SET &V-OPER = &1
```

results in &V-OPER equal to A OF B.

```
&EQU &2 &V-OPER
```

results in &2 equal to A OF B.

4.2.2 Initializing Variable Tables

The directive expression &INIT...&IEND establishes initial values when defining a variable table.

Format:

```
&INIT {literal}
      {NULL   } ...&IEND
```

&INIT...&IEND
is a directive expression that delimits table values.

literal
NULL
initializes every occurrence of a variable by the value literal, and when specified between &INIT and &IEND are recognized only if the defined variable table has one or more occurrences.

The literal must be valid for the class of variable it is to initialize. The literals are applied to the table occurrences in the order in which they are specified.

NULL can be specified only for non-numeric variables and represents "no value."

Example: The format of non-numeric and numeric variables provides for the definition of a variable table and the initialization of each occurrence of the variable to a given value.

```
&LOCAL &VSUM(3) = 'A' X
```

Each of the three occurrences of &VSUM has the initial value of 'A'.

```
&LOCAL &VSUM(3) X
&INIT 'A' 'B' 'C' &IEND
```

'A' is the value of the first occurrence, 'B' is the value of the second, and 'C' is the value of the third occurrence of the variable.

If fewer values are specified via &INIT...&IEND than are indicated in the definition, the value specified in the definition is used for each additional occurrence, as in:

```
&LOCAL &VSUM(5) = 'A' X
&INIT 'B' 'C' &IEND
```

The first occurrence of the table has the value 'B', the second has the value 'C', and all subsequent occurrences have the value 'A'. If the number of values equal that indicated in the definition, the value in the definition is not used.

4.2.3 Boolean (TRUE/FALSE) Control Variables

A Boolean variable can be used to set and test for the true and false states. It must be defined prior to any reference to its name.

Format:

```
{&EXTERN}
{&GLOBAL} &Bname =  {TRUE }
{&LOCAL }           {FALSE}
```

&EXTERN defines an external variable which can be referenced by all macros in all regions. Generally, the definition is placed in the first, or only, region. (For a complete description of regions, see the Glossary in Appendix A.)

&GLOBAL defines a global variable which can be referenced by all macros in a single region. Generally, the definition is placed in the first macro in the region. A global variable may have the same name as an external variable. In that case, references to the variable subsequent to the global definition and within the region containing this definition always refer to the global variable rather than the external variable.

&LOCAL defines a local variable which can be referenced only in the macro in which it is defined. Generally, the definition is placed at the beginning of the macro model. A local variable may have the same name as an external or global variable. In that case, references to the variable subsequent to the local definition and within the macro containing this definition always refer to the local variable rather than the external or global variable.

&Bname is the name of a Boolean variable.

= is the separator (required).

TRUE
FALSE initializes the Boolean variable to either the "true" or "false" state.

Example:

```
&LOCAL &B-FIRST-TIME = TRUE
    .
    .
    .
&IF &B-FIRST-TIME
   &SET &B-FIRST-TIME = FALSE
    .
    .
    .
&ENDIF
```

4.2.4 Reserved Variables

Several variables are predefined within the Translator and have specific uses. &VUPSI1-8, &VSOURCE, &VTARGET, and &VDIALECT are predefined as external non-numeric variables. They cannot be specified within &EXTERN, &GLOBAL, or &LOCAL directives or as the receiving item in an &SET or &SETR directive. (Refer to the description of options in the CA-MetaCOBOL+ *User Guide*.)

- The &VSOURCE value indicates the compiler level of the input source program.
- The &VTARGET and &VDIALECT indicate the compiler level of the output source program.
- The &VUPSIn variables have no special meaning to the Translator, they are available for user-defined functions.

4.2.5 Variable Naming Conventions

CA-defined external and global variables contained within many supplied and supported macro sets begin with the characters &V@ by convention. User-defined variables that do not contain the "@" character, therefore, are protected from inadvertently duplicating variable names in CA-supplied macro sets.

4.3 Symbolic Operand Attributes

Symbolic operand attributes support the analysis of symbolic operand values. Symbolic operand attributes can be used to determine the type and other properties of a word held in a symbolic operand.

Symbolic operand attributes for COBOL data items are extracted from the Attribute Table.

Notation Example: &n' Attribute

In the notation example, 'n' is an integer from 0-15 and completes the standard notation for a symbolic operand. An apostrophe and attribute (a 1-character code for the requested attribute) are included.

The 18 symbolic operand attributes are listed below.

Attribute Name	Macro Notation	Macro Picture	Description
Address	&n'A	9(5)	Attribute Table Position
Displacement	&n'D	9(5)	Zero Relative Off-set
Display Size	&n'9	9(8)	Numeric Item Display Size
External	&n'E	X(1)	External Item
Global	&n'G	X(1)	Global Item
Name Check	&n'B	X(1)	"Ambiguous" and Unique Names
Name Size	&n'N	9(3)	Symbolic Operand Word Length
Level	&n'L	X(2)	Translator Level Indicator
Occurs	&n'O	9(5)	Maximum Occurs value
Occurs Level	&n'K	9(1)	Subscript Requirements
Point	&n'P	9(2)	Decimal and Scaling Position
Redefines	&n'R	9(1)	Redfines and renames clause
Sign	&n'-	9(1)	Picture sign and Sign is Separate
Size	&n'S	9(8)	Item Storage Requirements
Synchronized	&n'Y	9(1)	Synchronized clause
Type	&n'T	X(1)	Symbolic Operand Word Categories
Usage	&n'U	X(1)	Translator Usage Indicator
Value	&n'V	9(1)	Value clause

The Name Check, Name Size, Type, and Usage attributes never produce a translator diagnostic. All other attributes produce a diagnostic when evaluating a data item name that is missing from the Symbol Table or Attribute Table. The diagnostic, named N06 DATA-NAME UNDEFINED, terminates the macro and sets the translator return code to 0012.

Use Name Check first to guarrantee error-free retrieval of data-item attributes. Name Check verifies that the name of the data item is in the Symbol Table and has attributes (return value Q). You should only retrieve an attribute for a unique data-item name, as designated by Name Check.

The Usage attribute does not verify that a data-item name is unique. When multiple attribute sets exist for a duplicate data-item name, Usage always retrieves attributes from the last attribute set defined.

The attributes of a data item are available only after macro processing finishes for the the entire data definition. More importantly, the attributes of a group data item are not accurate until after macro processing finishes for the entire data structure subordinate to the group item.

The attributes of literals and figurative constants are always available. Use Type or Name Check to determine if the symbolic operand value is a literal. Then use Level, Point, Sign, Size, or Usage to find more information about the value if it is a literal.

The Name Size attribute is always available, regardless of the contents of the symbolic operand.

The following sections describe each attribute. The macro notation and macro picture are given for each attribute. For some attributes, a table of returned values and descriptions of these values are included. The attribute return values, shown as nonnumeric literals, do not include the apostrophes as part of the code value.

Though all values returned by attributes are character values, those returning exclusively numeric values have a numeric macro picture (9). The value will be zero suppressed and left justified. The significance of these macro pictures is that the value returned will always be numeric and will potentially have the maximum length shown in the picture.

A character macro picture (X) implies that the return value may be either numeric or character return values. If a numeric value is a possible return value, it will have leading zeros. The Level attribute returns two character codes, some of which are COBOL level numbers with an initial zero as part of the code value. The leading zero will be present.

The Point attribute returns an algebraic value that may be negative; therefore, you must allow for the additional sign character. Otherwise, the return value might be truncated.

4.3.1 Attribute Descriptions

Name	Notation	PIC	Description
Address	&n'A	9(5)	Attribute Table Position

returns an integer value that is a relative position within the Attribute Table. The attributes of a data item can be found there. See the &SCANA macro directive for directions to use the value returned by Address.

Displacement	&n'D	9(5)	Zero Relative Off Set

returns an integer value that is an offset from the beginning of a data structure to where an item within the structure begins. The displacement for a structure's first item is zero.

The displacement value for COBOL level 77 and level 01 items is zero (0). The first item of a level 01 record also has a zero displacement value.

Items after the first have displacement values greater than zero. Displacement locates the relative position of the first byte of the item in relation to the beginning of the record (relative position zero).

Display Size	&n'9	9(5)	Numeric Item Display Size

returns an integer value for the COBOL, USAGE DISPLAY size for numeric data items. The actual COBOL USAGE is not considered: for example, PIC 9999 USAGE COMP-3 returns 4.

Nonnumeric items return a zero (0) value. Use the Size attribute to obtain the display size of nonnumeric items.

External	&n'E	X(1)	External Item

returns a character value with a 1-character length for items that do not specify the External clause. The character 'E' is returned for all items specifying the External clause and for all items subordinate to a Level 01 item specifying the External clause.

Global	&n'G	X(1)	Global Item

returns a character value with a 1-character length for items that do not specify the Global clause. The character 'G' is returned for all items specifying the External clause and for all items subordinate to an FD or Level 01 item specifying the Global clause.

Name	Notation	PIC	Description

Level &n'L X(2) Translator Level Indicator

returns a 2-character value for the Translator Level Indicator. Elements of the data division lacking a standard level number have unique codes. Data items return their COBOL level numbers. Values returned by Level and their descriptions are shown below.

Value Description

COBOL Examples

'01' COBOL Level 01 items, COBOL mnemonic-names, and user-defined mnemonic-names have a code value of the level number '01'.

'02'-'49' COBOL Level items 02-49 have their level number as a code value.

CA-MetaCOBOL+ Examples

' ' (spaces) COBOL DATA DIVISION SECTION headers, FILE, WORKING-STORAGE, LINKAGE, COMMUNICATION, and REPORT are coded as two spaces.

'00' COBOL Level indicators FD, CD, SD, and RD are coded as two zeros.

'63' Literal and figurative constants are coded as '63'.

'88' COBOL Level 88 condition names are coded as their level number.

'89' Special-Names condition names are coded as '89'.

'97' COBOL Special Registers are coded as '97'.

'98' COBOL Index names are coded as '98'.

Occurs &n'O 9(5) Maximum Occurs Value

returns a zero for an item that does not require subscripts. Occurs returns the maximum value for items that specify the COBOL Occurs clause.

Occurs Level &n'K 9(1) Subscript Requirements

returns a zero for an item that does not require subscripts. Occurs Level returns the number of subscripts needed for a correct reference to 1) an item specifying a COBOL Occurs clause or 2) an item subordinate to an item specifying a COBOL Occurs clause.

Name	Notation	PIC	Description
Point	&n'P	9(2)	Decimal and Scaling Positions

indicates the number of decimal positions for numeric literals and the number of implied decimal positions and scaling specifications in picture character strings.

Point returns an algebraic value. A positive sign is implied. Negative values are preceded by a leading minus sign (-).

Point has a value of zero when the picture of a data item does not specify decimal or scaling positions or when a literal lacks decimal positions.

Point is positive:

- when the picture of a data item specifies implied decimal positions (e.g., 9V99) or decimal scaling positions (e.g., P9).
- when a literal lacks decimal positions
- for a numeric literal with decimal positions (e.g., 5.65).

Point is negative when the picture of a data item specifies magnitude scaling positions (e.g., 9PPP).

Redefines &n'R 9(1) Redefines and Renames clauses

indicates the presence of the Redefines or Renames clauses.

Redefines is zero (0) for items that do not specify a Redefines or Renames clause and for items not subordinate to an item specifying the Redefines clause.

Redefines is one (1) for items specifying a COBOL Redefines or Renames clause and for items subordinate to an item specifying the Redefines clause.

Sign &n'- 9(1) Picture sign and Sign is Separate

Sign returns an integer value for the sign specification in picture strings and the Sign is Separate clause.

The value of Sign is:

0 for unsigned numeric items, unsigned numeric literals, nonnumeric items, and nonnumeric literals.

1 for signed numeric items and signed numeric literals

2 for items specifying a Sign is Separate clause.

Name	Notation	PIC	Description
Size	&n'S	9(8)	Item Storage Requirements

Size indicates the storage requirements (in number of bytes allocated) for a data item. The storage requirements for an item are determined by its picture character string and COBOL Usage. For nonnumeric items, the storage requirements and display size are equivalent. Use the Display size attribute to obtain the display size of numeric data items.

For numeric and nonnumeric literals, Size returns a character count. For nonnumeric literals, the count includes the delimiting characters and the prefix character.

For figurative constants and user-defined Symbolic-Characters, Size returns a one (1). One is the implied length for these literals.

Synchronized	&n'Y	9(1)	Synchronized Clause

returns a zero (0) for items that do not specify a COBOL synchronized clause. Synchronized returns a one (1) for items that do specify a COBOL synchronized clause.

Type	&n'T	X(1)	Symbolic Operand Word Categories

Non-numeric value representing the syntax type of the current value of the symbolic operand, as follows:

' '	blank space. None of the following.
'A'	Next word begins in Area A (Area A indicator).
'L'	Word is a literal or figurative constant.
'N'	Word is a NOTE or comment.
'S'	Word matches a String-type macro name.
'V'	Word is a verb or terminating period.

Name	Notation	PIC	Description
USAGE	&n'U	X(1)	

returns a character value that is a Translator Usage Indicator for a COBOL Usage or a COBOL item type. By analyzing characteristics of the item, Usage determines that the current value of the symbolic operand fits a category below.

Value	Description
'A'	Alphabetic data items with a picture character string in the form PICTURE A(n).
'C'	Condition-name defined as level 88 item or user-defined ON STATUS and OFF STATUS item in the SPECIAL-NAMES paragraph.
'E'	DBCS (originally EGCS) data item name with USAGE DISPLAY-1.
'F'	Figurative constant HIGH-VALUE, HIGH-VALUES, LOW-VALUE, LOW-VALUES, SPACE, SPACES, ZERO, ZEROS, ZEROES, or user-defined Symbolic Characters in the SPECIAL-NAMES paragraph.
'G'	Group data item name.
'I'	Index data item with USAGE is INDEX clause.
'K'	Class tests are NUMERIC, ALPHABETIC, and user-defined Class.
'J'	Index item defined in the INDEXED phrase of the OCCURS clause.
'M'	Mnemonic-names or user-defined names in the SPECIAL-NAMES paragraph.
'N'	Numeric literals including signed and unsigned integer literals (e.g., +123), signed and unsigned decimal literals (e.g., 123.456), and floating point literals (+12.3E+12).
'Q'	Non-numeric, DBCS (originally EGCS) or hexadecimal literals using quotes or apostrophes to enclose the literal value.

Value	Description
'R'	Report (edited) data item names having a picture with Special, Fixed, or Floating insertion editing, Zero Suppression, Replacement Editing, or have been defined with the clause Blank When Zero.
'S'	Sign conditions are NEGATIVE and POSITIVE.
'U'	Undefined. The operand value is either not in the Symbol Table or has no attributes in the Attribute Table.
'X'	Alphanumeric data item name having a picture character strings in the form Picture X(n).
'0'	Binary Numeric data item name with USAGE COMP and COMP-4.
'1'	Single Precision Internal Floating Point Numeric data item name with USAGE COMP-1.
'2'	Double Precision Internal Floating Point Numeric data item name with USAGE COMP-2.
'3'	PACKED Decimal Numeric data item name with USAGE COMP-3.
'4'	External floating-point data item names with a picture character string in the form PICTURE +9(n).+9(n)E+99 and USAGE DISPLAY.
'5'	Pointer data item name USAGE POINTER.
'9'	Numeric data item name with USAGE DISPLAY.

VALUE &n'V 9(1)

Numeric value indicating the presence or absence of a VALUE clause, as follows:

0 for items that do not specify a VALUE clause in this data item or a subordinate item.

1 for items that specify a VALUE clause in this data item or a subordinate item.

4.4 Concatenation

"Concatenation" links one or more constants and/or the current value of symbolic operands and/or variables. The concatenation operation begins and ends with the Open Concatenation and Close Concatenation Delimiters, respectively.

4.4.1 &(... &)

This form of the concatenation is used to create a single word from one or more separate items.

Format:

```
&( expression &)
```

&(
is the Open Concatenation Delimiter.

expression
consists of all words between the Open Concatenation Delimiter, "&(", and the Close Concatenation Delimiter, "&)".

The expression consists of one or more words, symbolic operands, literals, and variables. For variables, the current value is used in forming the concatenation. For symbolic operands, only the first word of the current value is used. Quotation marks bounding embedded literals are removed before concatenation, but any blanks contained within quotation marks are retained.

The maximum size of the resulting item is 128 characters. If the resulting item exceeds 128 characters, it causes a diagnostic and is truncated from the right to 128 characters.

&)
is the Close Concatenation Delimiter.

Examples: Variables which contain blanks will also retain blanks within concatenated words. NULL variables have no current value.

The following form of concatenation can also be used to create unique names by appending an incremented counter to a constant name.

```
&( THIS WILL BE ' RUN ' TOGETHER &)
```

produces

```
THISWILLBE RUN TOGETHER
```

In the following, one is added to &VCOUNT and the word "NAME-n", where "n" is the current value in &VCOUNT (with leading zeros suppressed), is substituted.

```
&SET &VCOUNT = &VCOUNT + 1
&( NAME- &VCOUNT &)
```

Concatenation can be used to add characters to the current value of a variable, as in:

```
&SET &VWORD = &( &VWORD &VCHAR &)
```

Words substituted directly from a concatenation prohibit a nested macro call. Thus, concatenation can be used to substitute a word which would normally cause an embedded call to another macro without making this call. The word is placed in a concatenation as shown:

```
&( WORD &)
```

Other uses of this form of concatenation are to remove quotes and to create a single word from multiple words for use in &NOTE and &ACCT directives.

A terminating punctuation character within a concatenation loses its identity as a terminating punctuation character. The following example shows a procedure header incorrectly output:

```
&SET &VCOUNT = &VCOUNT + 1
&A &( NAME- &VCOUNT . &)
```

To create the procedure header correctly, move the terminating punctuation character outside the concatenation following the Close Concatenation Delimiter.

```
&SET &VCOUNT = &VCOUNT + 1
&A &( NAME- &VCOUNT &) .
```

4.4.2 &(Q ... &)

This form of the concatenation is used to create a single non-numeric literal from one or more separate items.

Format:

```
&(Q expression &)
```

&(Q
is the Open Concatenation Delimiter.

expression
consists of all words between the Open Concatenation Delimiter, "&(Q", and the Close Concatenation Delimiter, "&)". The resulting item, unlike that of &(...&), is bounded by quotes.

The expression consists of one or more words, symbolic operands, literals, and variables. Variables which contain blanks also retain blanks within concatenated words. NULL variables contain no value. For variables, the current value is used in forming the concatenation. For symbolic operands, only the first word of the current value is used. Quotation marks bounding embedded literals are removed before concatenation, but any blanks contained within quotation marks are retained.

&)
is the Close Concatenation Delimiter.

Example: The maximum size of the resulting item is 130 characters including the bounding quotes. If the resulting item exceeds 130 characters in length, it causes a Translator diagnostic and the item is truncated from the right to 130 characters.

```
&(Q THIS WILL BE RUN TOGETHER IN QUOTES &)
```

produces

```
'THISWILLBERUNTOGETHERINQUOTES'
```

4.4.3 &(E ... &)

This form of the concatenation is used only as the sending item in an &EQU directive to string together multiple words into a single operand while retaining the integrity of each word.

Format:

```
&(E expression &)
```

&(E
is the Open Concatenation Delimiter.

expression
consists of all words between the Open Concatenation Delimiter, "&(E", and the Close Concatenation Delimiter, "&)".

The expression consists of one or more words, symbolic operands, literals, and variables. For symbolic operands and variables, the current value is used in forming the concatenation.

&)
is the Close Concatenation Delimiter.

Notes: Unlike the &(and &(Q formats, all words in the the current value of a symbolic operand are used in forming the concatenation.

Examples:

In the following,

```
&EQU &1 &(E NAME OF MASTER &)
```

places the qualified data-name NAME OF MASTER in a symbolic operand.

Quotation marks are not removed by this form of the concatenation. For example:

```
&EQU &1 &(E 'A B C' &)
```

results in the non-numeric literal, 'A B C', becoming the current value of &1, and is equivalent to:

```
&EQU &1 'A B C'
```

The "&(E" form of concatenation can be used to build and store a string of words in a symbolic operand in the form of a qualified and/or subscripted name, while retaining the integrity of the individual words which form the name.

```
&SET &VNAME = 'A'
&EQU &1 &(E &VNAME OF B &)
```

places the words "A OF B" in &1.

5. Macro Model Programming

Directives, directive expressions, constructs, logical destinations, substitution words, and variable definitions all constitute macro programming language.

5.1 General Format

A "directive" is a special word, embedded within a CA-MetaCOBOL+ macro model, which causes the Translator to take explicit action but does not appear in the output. A directive is similar in function to a COBOL verb which defines the action to be taken by a COBOL program. Directives always begin with an ampersand (&).

A "directive expression" is a series of words, beginning with a directive, which defines explicit Translator action. A directive expression is similar in function to a COBOL statement.

5.2 Types of Directives

The following functional categories of directives described in this chapter are:

- Branching directives
- Constructs which include selection and repetition directives
- Data manipulation directives
- Input directives

- Out-of-line generation which includes auxiliary, COBOL, general purpose, and ending directives
- Message directive
- Condition code directive
- Clock directive
- Accounting directive
- Formatting directives
- Data structure analysis directives
- Sub-program directives

5.3 Conditions

The definition of conditional expressions include simple conditions and combined conditions.

5.3.1 Simple Conditions

Simple conditions include relation conditions, state conditions, and Boolean conditions.

Relational Conditions

The relation condition compares the current values of two items.

Format:

```
                  Simple Relation Condition

{&Vname-1       }            {&Vname-2       }
{&n-1           }            {&n-2           }
{concatenation-1} relational {concatenation-2}
{literal-1      } operator   {literal-2      }
{&n'attribute-1 }            {&n'attribute-2 }
                             {NULL           }
```

&Vname-1, -2
is the name of a non-numeric or numeric variable.

&n-1, -2
is a symbolic operand &0 through &15, including &. If a symbolic operand is specified, only the first word of its current value is used in the comparison.

concatenation-1, -2
is a construction within a macro model which combines one or more words into a single word.

literal-1, -2
is a character string which may be a non-numeric literal, hexadecimal literal, or numeric literal (see the Glossary in Appendix A).

&n'attribute-1, -2
is a symbolic operand attribute with "n" having a numeric value of 0-15.

NULL
represents "no value."

relational operators
are the symbols for less than "<", equal to "=", greater than ">". In addition the following abbreviations are supported:

"LT"	less than
"LE"	less than or equal to
"EQ"	equal to
"NE"	not equal to
"GE"	greater than or equal to
"GT"	greater than

Any relational operator may be negated by inclusion of the preceding word NOT.

Notes: If both items are numeric, a numeric comparison is done; otherwise, a non-numeric comparison is done.

If an item is a non-numeric literal or a symbolic operand with a non-numeric literal value, bounding quotes are ignored in the comparison.

If the items are not the same length, the shorter one is expanded with trailing spaces or leading zeros, as appropriate to the class of the comparison.

Examples: The following are relational expressions that are valid simple relational conditions.

```
&1'U NOT = 'U'

&Vname EQ NULL

&1 NE '.'
```

State Conditions

The state condition is used to determine the "at end" state of an &SCAN-type directive.

Format:

```
                        Simple State Condition

      [NOT] ENDSCAN
```

NOT
is the false state.

ENDSCAN
is a keyword that can be specified in both simple and combined conditional expressions to determine the "at end" state of an &SCAN-type directive.

Note: The ENDSCAN test must be coded directly after the &SCAN-type directive in order to shelter against errors where &0 is not available for processing.

Example: The macro in the following example MOVEs SPACES to a group item and MOVEs ZEROS to each numeric elementary item of that group.

CA-MetaCOBOL+ Input

```
      .
      .
      .
SP    INIT &1(Q) :
      MOVE SPACES TO &1
      &REPEAT
          &SCAN &1 &1
      &UNTIL ENDSCAN
          &IF &0'U GE '0'
              MOVE ZEROS TO &0
          &ENDIF
      &ENDREP
      .
      .
      .
 DATA DIVISION.
      .
      .
      .
 01   CUSTOMER-RECORD.
      02  NAME           PICTURE X(30).
      02  ADDRESS        PICTURE X(30).
      02  CITY           PICTURE XX.
      02  ZIP            PICTURE 9(5).
      02  ACCOUNT-INFO   USAGE COMPUTATIONAL-3.
        03  PREVIOUS-BAL PICTURE S9(5)V99.
        03  CURRENT-BAL  PICTURE S9(5)V99.
      .
      .
      .
 PROCEDURE DIVISION.
      .
      .
      .
      INIT CUSTOMER-RECORD.
      .
      .
      .
```

CA-MetaCOBOL+ Output

```
      MOVE SPACES TO CUSTOMER-RECORD
      MOVE ZEROS TO ZIP
      MOVE ZEROS TO PREVIOUS-BAL
      MOVE ZEROS TO CURRENT-BAL
```

Boolean Conditions

The true (&Bname) or false (NOT &Bname) state of a Boolean variable can be determined by the following conditional expression.

Format:

```
Boolean Condition

[NOT] &Bname
```

NOT is the false state.

&Bname is the name of a Boolean variable.

Example:

```
&LOCAL &B-FIRST-TIME = TRUE
    .
    .
    .
&IF &B-FIRST-TIME
   &SET &B-FIRST-TIME = FALSE
       .
       .
       .
&ENDIF
```

5.3.2 Combined Conditions

A combined condition is formed by combining simple conditions with the logical connectives &AND or &OR.

Format:

```
                        Combined Condition

    simple-condition [ [&OR simple-condition]...  ]
                     [ [&AND simple-condition]... ]
```

simple-condition
is a relation, state, or Boolean condition.

&OR
is a directive that "separates" sets of simple conditions.

&AND
is a directive that "connects" simple conditions.

Note: &ORs and &ANDs cannot be intermixed in a single combined condition.

Examples:

```
&1'T EQ 'S' &OR &1'T EQ 'V' &OR &1'T EQ 'A'

&B-EDIT-OK &AND NOT &B-PREVIOUS-ERRORS

&B-PREVIOUS-WORD-TO &AND &1'T = 'L'
```

5.4 Logical Destinations

A "logical destination" is a user-defined name for a location in a macro model. The user can control which portions of a macro model are used during a macro call via branch directives &GO and &DO. Thus, the output of a macro call can be generated conditionally or in some sequence other than that in which the macro was written. The non-sequential points to which the Translator branches are called destinations.

As a class, logical destinations are defined only within the macro models and are never substituted.

The occurrence of a logical destination in a macro model represents the definition of that logical destination, unless the destination specified occurs within a CA-MetaCOBOL+ directive expression. There are two classes of logical destinations - "tags" and "labels."

Format:

{&T*name*}
{&L*name*}

&Tname
is a tag known only to the macro model in which it appears. A tag name must be a maximum of 30 contiguous alphanumeric characters, inclusive of the leading characters "&T", and exclusive of punctuation and parentheses.

&Lname
is a label known to all models of all macros within a region. A label name must be a maximum of 30 contiguous alphanumeric characters, inclusive of the leading characters "&L", and exclusive of punctuation and parentheses.

Notes: Branching from the model of a Word- or Prefix-type macro into the model of a String-type macro should be avoided because Translator results are unpredictable.

The word "&LOCAL" cannot be used as a label.

If the same tag is defined more than once in a macro model or the same label is defined more than once in a region, the Translator displays a diagnostic.

A special tag, "&T$", is predefined by the Translator as a tag beyond the end of the macro model and cannot be defined by the user. It is used to branch out of a macro, ending the call to that macro. (Refer to Appendix D.)

Tags and labels can be defined in Area A or Area B of macro models.

The same logical location may be known by more than one name by defining multiple tags and/or labels consecutively.

Many directives and directive expressions are supported only within the models of String-type macros. As a result, transfer from the models of Word-type and Prefix-type macros to the models of String-type macros may yield unpredictable results. It is suggested that global labels should be defined and referenced only within the models of String-type macros.

Example: Labels and tags are most often used to define macro subroutines, as when subdividing a complex macro model for functional clarity, as follows.

```
SP   prototype :
     &LOCAL variable definitions
     &DO &T-VALIDATE-OPERANDS
     &DO &T-GENERATE-CODE
     &GOBACK
 &T-VALIDATE-OPERANDS
     .
     .
     .
     &EXIT
 &T-GENERATE-CODE
     .
     .
     .
     &EXIT
```

CA-defined labels contained within many supplied and supported macro sets begin with the characters "&L@" by convention. User-defined labels that do not contain the "@" character, therefore, are protected from inadvertently duplicating labels in CA-supplied macro sets.

5.5 Line Output

Line output is used to output a complete line without additional formatting.

Format:

```
L[text]
```

L[text] is used to output a complete line without additional formatting. L appears in the continuation column (column 7).

Lines within a macro model with an L in the continuation column (column 7) are passed to the output without reformatting except:

1. The text is shifted left so that the position immediately following the continuation column (column 7) becomes column 1 on the output.

2. Any numeric and non-numeric variables are replaced by their current values, with appropriate columnar adjustment if the current value is longer or shorter than the variable name.

Notes: The maximum size of the output line is 110 positions. If substitution of variable values causes the line to expand beyond column 110, the expanded line is truncated after column 110, or, if directed to 80-column output, after column 80.

Line output directed to the output COBOL program (not &ANTE, &POST, &AUXN, or &AUX) has columns 1-6; if blank, it is overlaid by the sequence number and is truncated after column 72.

Line output directed out-of-line to &ANTE, &POST, &AUXN, or &AUX is not eligible for resequencing in columns 1-6 or inclusion of identification in 73-80. In addition, Line output records directed to &ANTE, &POST, or &AUX and containing either ** or */ in columns 1-2 are directed only to the Listing File, where ** defines a comment line and */ will cause a page eject when displayed on the Listing File.

The line is examined from the left to the right. Whenever the character string &V (not necessarily preceded by a space) is encountered, it is assumed to be the beginning of a variable name. The variable name is built by concatenating the &V with the first character to the right of the V. If the character to the right of the character last concatenated is greater than or equal to A, it is concatenated to the name and the process is repeated. When a character less than A is found, the name built by the concatenation process is examined to see if it conforms to the rules for a variable name. If the name does conform, a search is made for a matching known variable. If no match is found, or the name does not conform, a diagnostic is issued and output from this macro may be incorrect. If a match is found, the current value of the variable is substituted for the variable name in the Line output.

The first character of a variable name used in Line output can be an alphabetic, numeric, or special character. All other characters must be alphabetic or numeric. Alphabetic and numeric characters placed immediately to the right of a variable name may cause errors or incorrect translation, and, therefore, should be concatenated into the variable before executing the Line output. A variable containing a special character in other than the position immediately following &V cannot be used in Line output.

Unlike other output, concatenation is not performed in Line output nor is current value substituted for any symbolic operand (or attribute). These must be &SET into a variable prior to the variable being output.

Quotes have no special meaning in Line output; no attempt is made to examine the line for non-numeric literals.

Since the Line output text begins in column 8 and ends at column 72 (becoming columns 1-65 in the output), to output beyond column 65 requires variables containing current values large enough to carry the output line out to the desired position. The current value of such variables can consist of "multiple words" by building them up through a series of concatenations.

Words found in Line output are not subject to "reparsing" or macro nesting.

Line output to the DATA DIVISION does not cause attributes to be built for any item defined on that line.

The compiler-directing statements EJECT, SKIP1, SKIP2, and SKIP3, COBOL comment records, and blank lines must be substituted via Line output.

Symbolic operand storage variables cannot be used in Line output.

A variable defined with multiple occurrences cannot be used in Line output.

The subscript for a multiple defined variable is not recognized because the open parenthesis (is less than A.

Subscripts are not allowed in line output.

Examples: In the following, the macro computes the size of Working-Storage (including inter-record slack bytes) and generates a comment specifying this size.

```
WP   DIVISION :
     DIVISION
     &LOCAL &VWORK = 0 9(7)
     &LOCAL &VSIZE = 0 9(7)
     &REPEAT
         &SCAN WORKING-STORAGE WORKING-STORAGE
     &UNTIL ENDSCAN
         &IF &0'R = 0
             &IF &0'L = '01' &OR &0'L = '77'
                 &SET &VWORK = &0'S + 7
                 &SET &VWORK = &VWORK / 8
                 &SET &VWORK = &VWORK * 8
                 &SET &VSIZE = &VSIZE + &VWORK
             &ENDIF
         &ENDIF
     &ENDREP
     &DATAW
L     * WORKING-STORAGE SIZE IS &VSIZE BYTES
     &END
```

The number of spaces between words in Line output is always maintained. Thus, if "&VA word" appears in such a line with a single blank between "VA" and "word", the output will also have a single space regardless of the size of the current value of &VA. To achieve consistent columnar alignment, it is often necessary to ensure that variables always contain a current value of maximum size. This can be achieved by "padding" the variable with blanks by concatenating the variable with a blank literal as follows:

```
    .
    .
    .
    &LOCAL &VA X(6)
    .
    .
    .
    &SET &VA = &( &VA '      ' &)
    .
    .
    .
L       &VA
    .
    .
    .
```

5.6 Branching Directives

Branching directives allow for unconditional branching to logical destinations and unconditional termination.

5.6.1 &GO

The &GO directive allows an unconditional branch to a specified point in either the same or another macro model. The &GO directive expression is equivalent in function to the COBOL "GO TO" statement.

Format:

```
&GO {&Tname}
    {&Lname}
```

&GO
is a directive for unconditionally branching to a logical destination.

&Tname
is a tag known only to the macro model in which it appears.

&Lname
is a label known to all models of all macros within a region.

Example: The following indicates that the macro model execution continues with the model word after the tag "&TDONE".

```
&GO &TDONE
```

5.6.2 &GOBACK

The &GOBACK directive causes an unconditional termination of the macro model processing. The &GOBACK directive is similar in function to the COBOL "GOBACK" statement.

Format:

```
&GOBACK
```

&GOBACK
is a directive for unconditionally terminating the macro in which it appears.

Example: Labels and tags are most often used to define macro subroutines, as when subdividing a complex macro model for functional clarity. The &GOBACK is then used to terminate macro execution after the subroutines have been involved, but before their definitions, as follows.

```
SP   prototype :
     &LOCAL variable definitions
     &DO &T-VALIDATE-OPERANDS
     &DO &T-GENERATE-CODE
     &GOBACK        /* GETS OUT OF THE MACRO
 &T-VALIDATE-OPERANDS
     .
     .
     .
     &GOBACK        /* GETS OUT OF THIS ROUTINE AND MACRO
     .
     .
     .
     &EXIT
 &T-GENERATE-CODE
     .
     .
     .
     &EXIT
```

&GOBACK may further be used in labeled or tagged subordinate routines to exit not only from the subroutine but also from the invoking macro.

5.6.3 &DO

The &DO directive allows an unconditional branch to a logical destination.

Format:

```
&DO {&T name}
    {&L name}
```

&DO
is a directive for unconditional branching to a logical destination until &EXIT or &GOBACK are detected in the branched-to model statements. It can be coded only in a String macro model.

&Tname
is a tag known only to the macro model in which it appears.

&Lname
is a label known to all models of all macros within a region.

Notes: A segment of code in a macro model can be executed "out-of- line" by the &DO directive. This directive expression is similar in function to a COBOL "PERFORM" statement. When the &DO is executed, the macro call continues from the destination specified by name, and returns to the word immediately following the latest outstanding &DO directive expression when an &EXIT directive is encountered.

An &DO may appear within code executed by another &DO. Such nesting may continue for 32 levels.

If the code executed by an &DO contains branches to multiple destinations, multiple &EXIT/&GOBACK directives may be required to end the &DO.

Example: Labels and tags are most often used to define macro subroutines, as when subdividing a complex macro model for functional clarity. &DO is then used to invoke the subroutines, and &EXIT ends each subroutine and returns control following the invoking &DO, as follows.

```
SP   prototype :
     &LOCAL variable definitions
     &DO &T-VALIDATE-OPERANDS
     &DO &T-GENERATE-CODE
     &GOBACK
 &T-VALIDATE-OPERANDS
     .
     .
     .
     &EXIT
 &T-GENERATE-CODE
     .
     .
     .
     &EXIT
```

5.6.4 &EXIT

The &EXIT directive returns control to a model statement following an &DO.

Format:

```
&EXIT
```

&EXIT
is a directive that returns control to a model statement following an &DO. It can be coded only in a String macro model.

Note: If no &EXIT/&GOBACK directive is encountered before reaching the end of a macro model entered by means of an &DO, a diagnostic is generated and processing continues as though an &EXIT has been encountered. An &EXIT directive can be executed at any time by any macro and acts as a NULL directive unless an &DO is active.

Example: Labels and tags are most often used to define macro subroutines, as when subdividing a complex macro model for functional clarity. &DO is then used to invoke the subroutines, and &EXIT ends each subroutine and returns control following the invoking &DO, as follows.

```
SP   prototype :
     &LOCAL variable definitions
     &DO &T-VALIDATE-OPERANDS
     &DO &T-GENERATE-CODE
     &GOBACK
 &T-VALIDATE-OPERANDS
     .
     .
     .
     &EXIT
 &T-GENERATE-CODE
     .
     .
     .
     &EXIT
```

5.7 Constructs

"Constructs" are combinations of directives that define the logic and scope of selection and repetition functions.

These constructs are consistent with the control structures of modern structured programming practice that ensure a single entry and single exit for each logical function.

In all cases, these constructs begin with an explicit directive which is logically paired to an explicit ending directive. For example, &IF begins a construct which selects between two logical paths. The &IF construct is terminated by &ENDIF. Additional directives and directive expressions are explicitly defined in the construct syntax to define alternate logical paths, etc. For example, &ELSE defines the beginning of the false path of an &IF construct.

Additional constructs, directives, and substitution words can be placed within the scope of a construct, and will be acted upon when the logical path of the construct is executed.

5.7.1 Selection Directives

Selection constructs are used to choose between two (&IF) and multiple (&SELECT) alternate actions.

&IF/&ELSE/&ENDIF

The &IF construct selects between two logical paths, and does comparisons and conditional branching, and is terminated by an &ENDIF.

Format:

```
 &IF condition
      process-1
[&ELSE
      process-2]
 &ENDIF
```

&IF
is a directive that identifies the physical and logical beginning of the &IF construct.

condition
is a simple or combined condition.

process-1, -2
is one or more CA-MetaCOBOL+ directives, constructs, substitution words, or can be omitted.

&ELSE
is a directive that identifies the false path as a result of the truth test(s) in the condition.

&ENDIF
is a directive that identifies the physical and logical end of the &IF construct. It terminates only the most recently unterminated &IF construct.

Logic:
If the condition is true, processing continues with process-1. If the condition is not true, processing continues with process- 2.

Example: The following processes a symbolic operand containing a numeric or non-numeric literal, or determines an error.

```
&IF &1'T NOT = 'L' &OR &1'U = 'F'
    &DO &T-SYNTAX-ERROR
&ELSE
    &IF &1'U = 'Q'
        &DO &T-NON-NUMERIC
    &ELSE
        &DO &T-NUMERIC
    &ENDIF
&ENDIF
```

&SELECT/&WHEN/&ENDSEL - Format 1

The &SELECT construct provides a multi-path decision capability based on the value of a single selection subject, and is terminated by &ENDSEL. This format is similar to the classic structured CASE statement. It consists of three parts -- a heading that specifies the type of test that follows, a body that contains a series of tests followed by the action to be taken if the test is true, and optional postscripts that specify actions when none or any of the tests are true, or all are false.

Format:

```
 &SELECT select-subject
 &WHEN select-object [&OR select-object] ...
      process-1
[&WHEN select-object [&OR select-object] ...
      process-2]...
[&WHEN OTHER
      process-3]
[&WHEN ANY
      process-4]
 &ENDSEL
```

&SELECT
is a directive that identifies the physical and logical start of the &SELECT construct.

select-subject
is a symbolic operand (&n), non-numeric or numeric variable (&Vname), or attribute (&n'attribute).

&WHEN
is a directive that identifies an occurrence of the select-object, and defines the beginning of a logical path that will be executed when a select-object is equal to a select-subject.

select-object
is a symbolic operand (&n), non-numeric or numeric variable (&Vname), attribute (&n'attribute), literal, concatenation, or NULL which represents "no value" (&Vname is empty). The select-subject and select-object must be of the same class (numeric or non-numeric).

&OR
is a directive that connects optional select-objects.

process-n
is one or more directives, constructs, substitution words, or can be omitted.

&WHEN OTHER
is a directive that specifies process-3 is executed only if no select-object equals the select-subject.

&WHEN ANY
is a directive that specifies process-4 is executed if one of the select-objects is equal to the select-subject.

&ENDSEL
is a directive that terminates the &SELECT construct.

Logic: When a select-object is equal to the select-subject, the subsequent process is executed and control passes either to the &WHEN ANY, if specified, and process-4 is executed, or to the next executable model statement following the &ENDSEL. If no select-object is equal to the select-subject and if the &WHEN OTHER postscript is specified, process-3 is executed. If there is no match and the &WHEN OTHER postscript is not specified, control passes to the next executable model statement after &ENDSEL.

Example: The following compares the Type attribute of a symbolic operand:

```
&SELECT &1'T
&WHEN 'V' &OR 'S'
    &DO &TVERB-STRING-PROCESSING
&WHEN 'A'
    &DO &TAREA-A-PROCESSING
&WHEN 'N'
    &DO TNOTE-PROCESSING
&WHEN OTHER
    &DO &TWORD-PROCESSING
&ENDSEL
```

&SELECT/&WHEN/&ENDSEL - Format 2

This &SELECT construct is a more general version of the &SELECT construct, and is terminated by &ENDSEL. It consists of three parts -- a heading that specifies the type of test that follows, a body that contains a series of tests followed by the action to be taken if the test is true, and optional postscripts that specify actions when none or any of the tests are true, or all are false.

Format:

```
 &SELECT
 &WHEN condition-1
      process-1
[&WHEN condition-2
      process-2]...
[&WHEN OTHER
      process-3]
[&WHEN ANY
      process-4]
 &ENDSEL
```

&SELECT
is a directive that identifies the physical and logical start of the &SELECT construct.

&WHEN
is a directive that defines the beginning of a logical path that will be executed when the corresponding condition is true.

condition-n
is a simple or combined condition.

process-n
is one or more directives, constructs, substitution words, or can be omitted.

&WHEN OTHER
is a directive that specifies process-3 is executed only if no condition is true.

&WHEN ANY
is a directive that specifies process-4 is executed if one of the conditions is true.

&ENDSEL
is a directive that terminates the &SELECT construct.

Logic When a condition is true, the appropriate process is executed and control passes either to the &WHEN ANY, if specified, and process-4 is executed, or to the next executable model statement following the &ENDSEL. If none of the conditions are true and the &WHEN OTHER postscript is specified, process-3 is executed. If none of the conditions are true and the &WHEN OTHER postscript is not specified, control passes to the next executable model statement following &ENDSEL.

Example: The following illustrates that the order in which conditions are specified may be critical:

```
&SELECT
&WHEN &1'T = 'V' &AND &1 = '.'
    &DO &TPERIOD-PROCESSING
&WHEN &1'T = 'V' &OR &1'T = 'S'
    &DO &TVERB-STRING-PROCESSING
&WHEN &1'T = 'A'
    &DO &TAREA-A-PROCESSING
&WHEN &1'T = 'N'
    &DO &TNOTE-PROCESSING
&WHEN OTHER
    &DO &TWORD-PROCESSING
&WHEN ANY
    &DO &TEXIT
&ENDSEL
```

5.7.2 Repetition Directives

Repetition constructs are used to repeat a logical path until a condition is satisfied.

&REPEAT/&ENDREP

The &REPEAT construct provides an in-line looping structure, and is terminated by &ENDREP.

Format:

```
 &REPEAT
     [process-1]
[&UNTIL condition]
     [process-2...]
 &ENDREP
```

&REPEAT
is a directive that identifies the physical and logical beginning of the looping structure.

process-n
is one or more directives, constructs, substitution words, or can be omitted.

&UNTIL
is a directive that tests the condition, and terminates the &REPEAT construct when the condition is true.

condition
is a simple or combined condition.

&ENDREP
is a directive that terminates &REPEAT.

Logic Using &REPEAT, processing proceeds until the condition is tested. If the condition is true, control passes to the next executable model statement following the &ENDREP. If the condition is not true, process-2 is executed and control returns to process-1. This loop is repeated until the condition is true. Multiple &UNTIL clauses can be specified.

Example: The following illustrates an &GET loop. The loop continues to &GET/&STOW and process words until the end of the statement. Here the condition is defined as a verb, Area A indicator or a String macro-name.

```
&REPEAT
    &GET &1
&UNTIL &1'T = 'V' &OR &1'T = 'A' &OR &1'T = 'S'
    &STOW
    &SELECT &1'T
    &WHEN 'N'
         &DO &TNOTE-PROCESSING
    &WHEN 'L'
         &DO &TLITERAL-PROCESSING
    &WHEN OTHER
         &DO &TWORD-PROCESSING
    &ENDSEL
&ENDREP
```

&ESCAPE

The &ESCAPE directive terminates unconditionally any &REPEAT...&ENDREP construct.

Format:

```
&ESCAPE
```

&ESCAPE
is a directive that when executed within an &REPEAT...&ENDREP, causes the innermost &REPEAT in which it occurs to terminate. Control passes directly to the next executable statement after the corresponding &ENDREP.

Example: The following illustrates an &GET loop. The loop continues to &GET/&STOW and process words until the end of the statement. Here the condition is defined as a verb, Area A indicator or a String macro-name.

```
&REPEAT
    &GET &1
    &IF &1'T = 'V' &OR &1'T = 'A' &OR &1'T = 'S'
         &ESCAPE
    &ENDIF
    &STOW
    &SELECT &1'T
    &WHEN 'N'
         &DO &TNOTE-PROCESSING
    &WHEN 'L'
         &DO &TLITERAL-PROCESSING
    &WHEN OTHER
         &DO &TWORD-PROCESSING
    &ENDSEL
&ENDREP
```

5.8 Data Manipulation Directives

Data manipulation directives are those that modify the contents of numeric and non-numeric variables and symbolic operands.

5.8.1 &SET - Format 1

The &SET directive initializes the left operand to the value of the right operand.

Format:

```
                    {&Vname-2     }
                    {&n           }
      &SET &Vname-1 = {concatenation}
                    {literal      }
                    {&n'attribute }
                    {NULL         }
```

&SET
is a directive that stores a value in a variable.

&Vname-1
is the name of an alphanumeric or numeric variable that will be initialized to the value of the right operand.

= is the required separator.

&Vname-2
is the name of an alphanumeric or numeric variable.

&n
is an initialized symbolic operand &0 through &15.

concatenation
is a construction which combines one or more words into a single word. Concatenation may be &(... &), an "ordinary" concatenation, or &(Q ... &), a "literal" concatenation.

literal
is a character string which may be an alphanumeric, hexadecimal (see the Glossary in Appendix A), or numeric literal.

&n'attribute
is a symbolic operand attribute where "n" represents a numeric value of 0-15.

NULL

represents "no value" (i.e., &Vname is empty). Only valid when &Vname is an alphanumeric variable.

Notes: &Vname is initialized left to right.

When the initialization value is longer than &Vname, the right-most excess characters are truncated.

When &Vname is numeric, the initialization value:

1. cannot be NULL,
2. must be numeric, and
3. may contain an optional leading sign (+ or -).

When &Vname is numeric and the initialization value is alphanumeric, the following Translator diagnostic will result:

N04 NON-NUMERIC DATA IN &SET OR &IF

When &Vname is a numeric or non-numeric variable and the initialization value does not contain quotes, only the first word is retained, regardless of the number of words in the initialization value.

When &Vname is a numeric or non-numeric variable and the initialization value contains quotes:

1. All words between the quotes are retained, but
2. The bounding quotes themselves are not retained.

When &Vname is a symbolic operand storage variable:

1. The initialization value must be a symbolic operand and
2. All words and any bounding quotes are retained.

All variables retain their current contents until altered by a &SET or &SETR.

Examples: In the following statements, &VNUMBER assumes the value of the Size attribute of symbolic operand &1.

```
&LOCAL &VNUMBER 9(7)
&SET &VNUMBER = &1'S
```

In the following, an ordinary concatenation is used and the result of &VALPHA will be AB.

```
&LOCAL &VALPHA X(2)
&SET &VALPHA = &( A B &)
```

In the following, a literal concatenation is used and the result of &VALPHA will be AB.

```
&LOCAL &VALPHA X(2)
&SET &VALPHA = &(Q A B &)
```

5.8.2 &SET - Format 2

The &SET directive initializes the left operand to an algebraically correct, arithmetic result for the two right operands. The right operand values must be numeric; leading signs (+ and -) are optional.

Format:

```
                {&Vname-2        }  {+}  {&Vname-3        }
                {&n-1            }  {-}  {&n-2            }
&SET &Vname-1 = {concatenation-1 }  {*}  {concatenation-2 }
                {literal-1       }  {/}  {literal-2       }
                {&n'attribute-1  }       {&n'attribute-2  }
```

&SET
is a directive that stores a value in a variable.

&Vname-1
is the name of an alphanumeric or numeric variable that will be initialized to the arithmetic result.

=
is the required separator.

&Vname-2
is the name of an alphanumeric or numeric variable.

&n-1
is an initialized symbolic operand &0 through &15.

concatenation-1
is a construction within a macro model which combines one or more words into a single word. Concatenation may be a string or literal concatenation.

literal-1
is a character string which may be an alphanumeric, hexadecimal, or numeric literal (see the Glossary in Appendix A).

&n'attribute-1
is a numeric symbolic operand attribute where "n" represents a numeric value of 0-15.

+, -, *, /
represent addition, subtraction, multiplication, and division, respectively.

&Vname-3
is the name of a numeric variable.

&n-2
is an initialized symbolic operand &0 through &15.

concatenation-2
is a numeric concatenation.

literal-2
is a numeric character string.

&n'attribute-2
is a numeric symbolic operand attribute.

Notes: &Vname is initialized right to left.

When the initialization value is longer than &Vname, the left-most excess characters are truncated.

Leading zeros are suppressed.

A positive result is unsigned.

Division yields an integer result.

The largest positive value is 99,999,999,999 (11 digits).

The largest negative value is -9,999,999,999 (10 digits and a sign).

The following Translator diagnostic will occur when either of the right operands is alphanumeric or NULL:

```
N04 NON-NUMERIC DATA IN &SET OR &IF
```

Examples: In the following statements, &VNUMBER assumes the value of the Size attribute of symbolic operand &3 minus 1.

```
&LOCAL &VNUMBER 9(7)
&SET &VNUMBER = &3'S - 1
```

Creating an arithmetic result larger than the maximum size of a variable will cause the left-most digit positions of the result to be truncated.

```
&LOCAL &VN 9
&SET &VN = 9
&SET &VN = &VN + 1
```

The result of &VN + 1 is 10; however, since &VN is only large enough to contain one character (integer), the left-most digit position is truncated, initializing &VN to 0.

5.8.3 &SET - Format 3

The &SET directive initializes the left operand to the value of the right operand. The beginning character position of the right operand is determined by the "character shift" operator %.

Format:

```
                    {&Vname-2        }     {&Vname-3        }
                    {&n-1            }     {&n-2            }
&SET &Vname-1 = {concatenation-1}  %  {concatenation-2}
                    {literal-1       }     {literal-2       }
                    {&n'attribute-1  }     {&n'attribute-2  }
                    {NULL            }
```

&SET
is a directive that stores a value in a variable.

&Vname-1
is the name of an alphanumeric or numeric variable that contains the contents of the right operand as a result of a move.

=
is the required separator.

&Vname-2
is the name of an alphanumeric or numeric variable.

&n-1
is an initialized symbolic operand &0 through &15.

concatenation-1
is a construction within a macro model which combines one or more words into a single word. Concatenation may be an ordinary or literal concatenation.

literal-1
is a character string which may be an alphanumeric, hexadecimal (see the Glossary in Appendix A), or numeric literal.

&n'attribute-1
is a symbolic operand attribute where "n" represents a numeric value of 0-15.

NULL
represents "no value" (i.e., &Vname is empty). Only valid when &Vname is an alphanumeric variable.

%
is the character shift operator.

&Vname-3
is a % operand that represents the name of a numeric variable.

&n-2
is a % operand that represents an initialized symbolic operand &0 through &15.

concatenation-2
is a % operand that represents a numeric concatenation.

literal-2
is a % operand that represents a numeric character string.

&n'attribute-2
is a % operand that represents a numeric symbolic operand attribute.

Notes: &Vname is initialized left to right.

When the initialization value is longer than &Vname, the right-most excess characters are truncated.

When &Vname is alphanumeric:

1. &Vname is initialized NULL when the % operand is beyond the length of the initialization value.

2. &Vname is initialized to a single space when the % operand is 0 or a negative value.

When &Vname is numeric:

1. The initialization value cannot be NULL and must be numeric.

2. A leading sign (+ or -) is optional.

3. The % operand must be greater than 0 and less than or equal to the length of the initialization value.

The following Translator diagnostic will occur when:

1. The % operand is alphanumeric or NULL.

2. &Vname is a numeric variable and the initialization value is alphanumeric or NULL.

3. &Vname is a numeric variable and the % operand is 0, negative, or greater than the length of the initialization value.

N04 NON-NUMERIC DATA IN &SET OR &IF

Examples: In the following statements, the character "C" is stored in &V2.

```
&LOCAL &V1 X(5)
&LOCAL &V2 X
&SET &V1 = 'ABC'
&SET &V2 = &V1 % 3
```

The following &SET expressions are logically equivalent.

```
&LOCAL &VX X
&SET &VX = &VY
&SET &VX = &VY % 1
```

5.8.4 &SET - Format 4

The &SET directive initializes the left operand to one word of the right operand. The "word shift" operator # uses the integer value to its right (must be greater than 0) as the determining word shift for the value to its left. The result is one word to which the left operand is initialized.

Format:

```
                                {&Vname-2     }
                                {&n-2         }
      &SET &Vname-1 = {&n-1}  #  {concatenation}
                      {NULL}     {literal     }
                                 {&n'attribute }
```

&SET
is a directive that stores a value in a variable.

&Vname-1
is the name of an alphanumeric or numeric variable that is initialized to the resulting value (one word) of the right operand.

=
is the required separator.

&n-1
is an initialized symbolic operand &0 through &15.

NULL
represents "no value" (i.e., &Vname is empty). Only valid when &Vname is an alphanumeric variable.

#
is the word shift operator.

&Vname-2
is a # operand that represents the name of a numeric variable.

&n-2
is a # operand that represents an initialized symbolic operand &0 through &15.

concatenation
is a # operand that represents a construction within a macro model which combines one or more words into a single word. Concatenation must be numeric and may be an ordinary or literal concatenation.

literal
is a # operand that represents a numeric character string.

&n'attribute
is a # operand that represents a symbolic operand attribute.

Notes: &Vname is initialized left to right.

When the initialization value is longer than &Vname, the right-most excess characters are truncated.

When &Vname is alphanumeric:

1. &Vname is initialized NULL when the # operand is beyond the last word of the initialization value.

2. &Vname is initialized to a single space when the # operand is 0 or a negative value.

When &Vname is numeric:

1. The initialization value cannot be NULL and must be numeric.

2. A leading sign (+ or -) is optional.

3. The # operand must be greater than 0 and less than or equal to the length of the initialization value.

The following Translator diagnostic will occur when:

1. The # operand is alphanumeric or NULL.

2. &Vname is a numeric variable and the initialization value is alphanumeric or NULL.

3. &Vname is a numeric variable and the # operand is 0, negative, or greater than the length of the initialization value.

```
N04 NON-NUMERIC DATA IN &SET OR &IF
```

Examples: In the following statements, assuming &1 contains A OF B, the word "OF" is stored in &VWORD.

```
&LOCAL &VWORD X(30)
&SET &VWORD = &1 # 2
```

Below, the following &SET expressions are logically equivalent.

```
&SET &VX = &1
&SET &VX = &1 # 1
```

5.8.5 &SET - Format 5

The &SET directive alters the truth value of a Boolean variable.

Format:

```
&SET &Bname = {TRUE }
              {FALSE}
```

&SET
is a directive used to modify the truth value of the subject Boolean variable.

&Bname
is the name of a Boolean variable.

= is the separator (required).

TRUE
FALSE
alters the Boolean variable to either the "true" or "false" state.

Note: Refer to Section 4.2.3 for further information on Boolean variables and Section 5.3.1 for Boolean conditions.

Example:

```
&LOCAL &B-FIRST-TIME = TRUE
       .
       .
       .
&IF &B-FIRST-TIME
   &SET &B-FIRST-TIME = FALSE
       .
       .
       .
&ENDIF
```

5.8.6 &EQU

The &EQU directive sets the current value of a symbolic operand (&n-1) to the value of item-2.

Format:

```
                {&Vname       }
                {&n-2         }
&EQU &n-1 =     {concatenation}
                {literal      }
                {&n'attribute }
                {NULL         }
```

&EQU
is a directive that sets the current value of a symbolic operand to value of item-2.

&n-1
is a symbolic operand &1 through &15.

&Vname
is the name of a non-numeric, numeric, or symbolic operand storage variable.

&n-2
is a symbolic operand &0 through &15.

concatenation
is a construction within a macro model which combines one or more words into a single word. It can also be the "&(E" form of concatenation described in Section 4.4.

literal
is a character string which may be a non-numeric, hexadecimal (see the Glossary in Appendix A), or numeric literal.

&n'attribute
is a symbolic operand attribute with "n" having a numeric value of 0-15.

NULL
represents "no value."

Notes: The &EQU directive is useful for establishing a current value in a symbolic operand, for modifying the contents of a symbolic operand, and for restoring the current value of a symbolic operand storage variable to a symbolic operand.

Examples: Assuming that &0 contains NAME OF MASTER, the following will place NAME OF MASTER in &V-OPERAND, &1 and &2, as well.

```
     .
     .
     .
&LOCAL &V-OPERAND S
&SET &V-OPERAND = &0
&EQU &1 &0
&EQU &2 &V-OPERAND
```

In the following,

```
&EQU &1 &(E NAME OF MASTER &)
```

places the qualified data-name NAME OF MASTER in a symbolic operand.

5.8.7 &SETR

The &SETR macro directive initializes a user-defined macro variable to the current value of a Translator Special Register.

The macro variable may be an &EXTERN, &GLOBAL, &LOCAL variable. Its definition must have a picture. The reference to the macro variable may be subscripted. Once initialized, the macro variable can be evaluated to determine the significance of the register setting.

Translator Special Register Names may only be used with the &SETR macro directive. The functions of the 20 Translator Special Registers are divided as follows:

- Nine registers return the value of the Translator options.
- Six provide information about the text being processed.
- Five provide the date, time, Translator return code, access to the Input Exit Interface area, and the address of the next available Attribute Table Entry.

Notation Example

```
&SETR &Vname = REGISTER-NAME
```

The 20 Translator Special Registers are listed below.

Special Register Name	Macro Picture	Description
ADDRESS	9(5)	Next available attribute table entry
COND	9(4)	Current Translator return code
COPY	9(1)	Translator OPTION COPY
DATE	X(8)	Current date
DDID	9(4)	Translator OPTION DDID
DEPTH	9(2)	Translator OPTION DEPTH minus 10
ID	X(8)	Identification Area Content
INVDEC	9(1)	Translator OPTION INVDEC and DECIMAL-POINT IS
IXIT	X(128)	Translator Input Exit return area
LINE	9(6)	Translator Input line number
NOTE	X(2)	Translator Note type
OPSYS	X(1)	Translator OPTION DIALECT Operating System
PGM	X(8)	PROGRAM-ID program name.
PSTAT	X(4)	Translator OPTION PSTAT
PVER	9(4)	Translator OPTION PVER
SEQ	X(6)	Sequence Number Area
XPGM	X(8)	Translator OPTION IXIT
STATUS	9(1)	Source or type of input record.
TIME	X(8)	Current time
VAR	X(30)	Translator OPTION VAR

The COPY, DDID, DEPTH, INVDEC, OPSYS, PSTAT, PVER, XPGM, and VAR Translator Special Registers all return a Translator Option value. The value is either the default established for the option or the value established by an option statement.

The ID, LINE, NOTE, PGM, SEQ, and STATUS Translator Special Registers all return information about the source text being processed. All these registers except PGM are dynamic: their values change constantly as a result of macro processing.

PGM reflects the name of each program when the source test consists of nested programs; however, once the name has been captured, it will remain unchanged until the end of the program.

Of the remaining Translator Special Registers, DATE, TIME, and COND are operating system-specific.

ADDRESS relates to use of the Attribute Table. IXIT is meaningful only when the &CALL directive is used to interface with an Input Exit subprogram.

Each special register is described below. Some descriptions include a table of return values. These return values, shown as nonnumeric literals, do not include the apostrophes as part of the code value.

All values returned by special registers are character values, but those returning exclusively numeric values have a numeric macro picture (9). The value is zero suppressed and left justified. The numeric macro pictures will always return a numeric value with potentially the maximum length shown in the picture.

A character macro picture (X) implies that the return value may be either numeric or character code values. If a numeric value is a possible return value, it will have leading zeros. The NOTE special register returns left justifed 1- and 2-character codes.

Special Register Name	**Macro Picture**	**Description**
ADDRESS	9(5)	Next Available Attribute Table Entry

returns an integer value for the next available entry in the Attribute Table. Address is useful in a $-LEVEL macro because, after macro processing completes, it is the address of the data item definition being processed. Address is also useful when you are using the Attribute Table to store information that is meaningful to the macro processing but will not necessarily become part of the generated program.

COND	9(4)	Current Translator Return Code

returns an integer for the current value of the Translator's return code. This value may be incremented by the &FLAG, &COND, and &NOTE macro directives and by Translator Advisory, Warning, and Error Messages. The Translator's return code is incremental. It always shows the highest value established by any of the macro directives or messages and can not be set to a lower value.

COPY 9(1) Translator OPTION COPY

returns an integer value for the COPY option setting, which may be established by the default setting or a setting established by an OPTION statement. Possible values are:

0 OPTION COPY=PASSIVE
1 OPTION COPY=ACTIVE
2 OPTION COPY=IGNORE

DATE X(8) Current Date

return a character value for the current date in the form MM/DD/YY. MM is the month; DD is the day, and YY is the year.

DDID 9(3) Translator OPTION DDID

returns an integer for the value specified by the DDID option (applicable to CA-DATADICTIONARY conventions).

DEPTH 9(2) Translator OPTION DEPTH (minus 10)

returns an integer for the value specified by the DEPTH option.

ID X(8) Identification Area Content

returns a character value for the contents of the Identification Area (positions 73-80) of the current input record, except when the current input record is a -INC statement. If the input record is a -INC statement, the value is retrieved from the previous input line.

INVDEC 9(1)Translator OPTION INVDEC and DECIMAL POINT IS

returns an integer value for the setting of the INVDEC option and the presence of 'DECIMAL POINT IS COMMA' in the source text. Possible values are:

0 NOINVDEC or DECIMAL POINT IS COMMA are not present

1 INVDEC or DECIMAL POINT IS COMMA are present

IXIT X(128) Translator Input Exit return area

returns a character value for the contents of the Input Exit return area after the completion of the most recent &CALL to the Input Exit.

LINE 9(6) Translator Input Line Number

returns an integer for the Translator Input Listing line number for the current input line, except when the current line is a -INC statement. If the current line is a -INC statement, the value is retrieved from the previous line.

NOTE X(2) Translator Note Type

is a character value that returns a 1- or 2-character code. This code identifies the Symbolic Operand Type Attribute of the word currently typed as a Note (i.e., &n'T = 'N'). Possible codes are listed below.

'A' &ACCT macro directive
'B' Blank lines
'C' CICS statements (OPTION PRESERVE=CICS must be specified)
'D' Debugging lines (cc7 = D)
'DP' &DSTOP macro directive
'DS' &DSTART macro directive
'E' Translator enabled comment (OPTION ENABLED must be specified)
'F' &B, &INDENT, &OUTDENT, &SETTAB, &GOTAB, and &MARGIN macro directives
'L macro Line output
'MK' &MARKER macro directive
'N'

- COBOL comments (cc7 = *)
- AUTHOR, INSTALLATION, DATE-WRITTEN,
- DATE-COMPILED, SECURITY, and REMARKS paragraphs
- COBOL NOTE statements
- paragraphs with the COMMENT and XCOM options specified
- *$MCT and *$NOMCT Translator Directing Statements
- any text between *$MCT and *$NOMCT Translator Directing
- Statements

'S' Spacing controls (EJECT, SKIP1, SKIP2, SKIP3) or -INC
'0' &END macro directive
'1' &ANTE macro directive
'2' &ENV macro directive
'3' &DATAF macro directive
'4 &DATAWS macro directive
'5' &DATAW macro directive
'6' &DATAWX macro directive
'7' &DATAL macro directive
'8' &DATAR macro directive
'9' &DATA macro directive
'10' &PROCS macro directive
'11' &PROC macro directive
'12' &PROCX macro directive
'13' &POST macro directive
'14' &AUX macro directive
'15' &POINT macro directive
'18' &DUMMY macro directive

OPSYS X(1) Translator OPTION DIALECT Operating System

returns a 1-character code that is the second character of the DIALECT option. Both characters of this option are available in the Reserved Variable &VDIALECT. Possible values for OPSYS are:

'D' for DOS (VSE) Operating System
'O' for OS (MVS) Operating System, PC-DOS, or MS-DOS

PGM X(8) PROGRAM-ID Program name

is a character value consisting of the first 8 characters of the comment entry following the PROGRAM-ID paragraph header in the IDENTIFICATION DIVISION.

PSTAT X(4) Translator OPTION PSTAT

returns a character value for the value specified in the PSTAT option (applicable to CA-DATADICTIONARY conventions).

PVER 9(4) Translator OPTION PVER

returns an integer value for the value specified in the PSTAT option (applicable to CA-DATADICTIONARY conventions).

SEQ X(6) Sequence Number Area

returns a character value for the contents of the Sequence Number Area (positions 1-6) of the current input record, except when the current input record is a -INC statement.

XPGM X(8) Translator OPTION IXIT

Returns an integer value for the IXIT option. The IXIT option is the name of the Input Exit subprogram.

STATUS 9(1) Source or Type of Input Record

is an integer value that identifies the source or type of the current input line. Values are:

0 Normal input
1 COPY or -INC statement
2 COPY or -INC text

TIME X(8) Current Time

is a character value for the current time. Time is given in the form HH:MM:SS. HH is the hour; MM is the minute, and SS is the second.

VAR X(30) Translator OPTION VAR

is a character value for the VAR option. The VAR option is a user-defined value 1-30 characters in length.

5.8.8 &PIC

The &PIC directive analyzes a picture-string and returns 3 words (type, display size, decimal point location) to a symbolic operand.

Format:

```
&PIC &n-1    {&n-2  }
             {&Vname}
```

&PIC
is a directive that is used for picture-string analysis. It can be used in the model of a Word, Prefix, or String macro for any division.

&n-1
is a symbolic operand &1 through &15 which contains the results of the analysis as 3 words:

1.
Type is the first word returned in &n-1 and is a non-numeric literal:

Type	Picture Clause
'E'	group data item
'R'	report item
'X'	non-numeric
'0'	floating point
'2'	signed numeric
'9'	unsigned numeric

2. Display size is the second word returned in &n-1, and is an 8-digit numeric literal representing the display size of the picture. Contains all leading zeroes, which are not suppressed.

3. Decimal point location is the third word returned in &n-1, and is a 5-digit signed numeric literal representing the decimal point location as denoted by the Point data attribute. Contains all leading zeroes, which are not suppressed.

&Vname
is the name of a non-numeric or numeric variable that contains the picture-string to be analyzed.

&n-2
is a symbolic operand &1 through &15 that contains the picture-string to be analyzed.

Notes: The USAGE, etc., of the data item containing the picture analyzed is not considered.

If an invalid picture is analyzed, illegal characters are ignored and the results are unpredictable. Under such circumstances, it is possible to have a size of 00000 returned as the second word.

Example: The following macro illustrates the use of the &PIC directive to analyze the parts of a PICTURE clause in the DATA DIVISION.

```
SD   PICTURE &1 :
         &GLOBAL &VTYPE X
         &GLOBAL &VSIZE 9(8)
         &GLOBAL &VPOINT 9(5)
         &DO &LP
         &GOBACK
     &LP
         &PIC &2 &1
         &SET &VTYPE = &2
         &SET &VSIZE = &2 # 2
         &SET &VPOINT = &2 # 3
         &EXIT
SD   PICTURE IS &1 :
         &DO &LP
```

If the clause "PICTURE S999V99" is analyzed by this PICTURE macro, &VTYPE contains '2', &VSIZE contains 00000005, and &VPOINT contains +0002.

To suppress leading zeros and the sign, place words 2 and 3 in variables and add zero, as shown below:

```
&SET &VSIZE = &VSIZE + 0
&SET &VPOINT = &VPOINT + 0
```

5.9 Input Directives

Input directives are used to analyze word sequences and to copy text into generated sources from String-type macro models.

5.9.1 &GET/&STORE/&STOW

The &GET, &STORE, and &STOW directives are used to acquire words following a String-type macro call.

Format:

```
                    [ &STORE ]
&GET &n    . .      [ &STOW  ]
```

&GET
is a directive that places a copy of the next word into a symbolic operand. It can be coded only in a String macro model.

&n
is a symbolic operand &1 through &15.

&STORE
is a directive that removes the single word last acquired via an &GET from the input stream.

&STOW
is a directive that removes the word acquired via an &GET from the input stream, including any associated qualifiers and/or subscripts/indices, and stores the entire item in a symbolic operand.

Notes: During an &STOW, if a COBOL comment is encountered while the Translator is searching for a potential qualifier, subscript, or index, the search ends and the Translator treats the symbolic operand as though no modifier were defined. (This may cause the next line to be read.)

Following an &GET, until an &STORE or &STOW is executed, only a copy of the input word is in &n; the word remains in the input stream.

A successive &GET may not be issued within the macro model until the current operand is &STOREd or &STOWed.

If no &STORE or &STOW is issued by the time of exit from the macro model, the word obtained by the last &GET remains in the input stream.

If the word matches a Word or Prefix prototype for the current division, the input word causes a nested macro call. The first substitution word from the nested macro is passed to the symbolic operand as though it were in the input stream.

If the word matches the name of the String-type prototype, the input word does not cause a call to the String macro, and the Type attribute of &n (&n'T) is set to 'S'.

If the item acquired by the &GET is not a qualified and/or subscripted name, there is no difference between the results of &STORE and &STOW.

A comma or semi-colon acquired by &GET...&STOW/&STORE is bypassed.

Supplementary Notes:

There are three conditions that cause the &GET directive to acquire words that do not appear in the source. The specific conditions under which these dummy words are acquired are itemized below.

1. Copied/Included Input

If the next word to be "acquired" by &GET begins a COBOL COPY statement or a LIBRARIAN -INC command, the word actually acquired by the &GET depends on the translate-time options specified.

- COBOL COPY

  If the COPY option is specified, &GET acquires the copied text as though it were in the input stream and does not acquire the COPY statement. If the Translator cannot retrieve the library text for a COPY in the DATA DIVISION, a period is returned.

  If the COPY option is not specified, &GET acquires only the COPY statement and not the associated text.

- LIBRARIAN -INC

  If the -INC option is specified and if the LIBRARIAN -INC command is subordinate to a *$LIBED statement from the primary input file, the &GET acquires the included text as though it were from the input stream and does not acquire the -INC command.

  If the -INC option is not specified and/or the -INC command is not subordinate to a *$LIBED statement from the primary input file, &GET acquires only the -INC command and not the associated text.

For example, given the following source library member named WKLIB:

```
01  WORK-AREA.
    02  WORK-KEY. . .
    .
    .
    .
    02  WORK-END PIC X.
```

Successive &GET...&STORE/&STOW of the input statements:

```
01  WORK-RECORD COPY WKLIB.
    .
    .
    .
```

With the COPY option specified, acquires the following current values, Type attributes, and &SETR...STATUS values:

Current Value	**Type**	**Status**
01	L	0
WORK-RECORD	blank	0
.	V (period after WORK-AREA)	2
02	L	2
WORK-KEY	blank	2
...		
02	L	2
WORK-END	blank	2
PIC	blank	2
X	blank	2
.	V	2
next input word	word type	0

With the COPY option not specified, the same input would result in:

Current Value	**Type**	**Status**
01	L	0
WORK-RECORD	blank	0
COPY	blank	1
WKLIB	blank	1
.	V	1
next input word	word type	0

Note: The preceding examples are based upon the 1968 ANSI COBOL standard for COPY statements. The 1974 ANSI COBOL COPY requires slightly different source and/or library syntax.

1. Area A

   An &GET for a word appearing in Area A first acquires a dummy word with Type attribute 'A'. After an &STORE/&STOW, a subsequent &GET is required to obtain the actual word in Area A.

   For instance, logic to successively &GET and &STORE/&STOW the following statements:

```
      .
      .
      .
     GO TO READ-INPUT.
SUBROUTINE-X.
      .
      .
      .
```

   Acquires the current values and Type attributes:

Current Value	Type
GO	V
TO	blank
READ-INPUT	blank
.	V
dummy	A (AREA A indicator)
SUBROUTINE-X	blank
.	V

2. Out-of-line from nested macro calls

   An &GET for a nested Word or Prefix macro which generates out-of- line code acquires a dummy word with Type attribute 'N', representing the out-of-line directive executed.

   For instance, given the following Prefix-type macro:

```
PP   SET= :
         MOVE '1' TO &
         &DATAWS
 77  & PIC X.
         &END
```

   logic to successively &GET and &STORE/&STOW the statements:

```
IF JOE EQUAL '0' SET=JOE GO TO ...
```

   acquires the current values and Type attributes:

Current Value	Type	&SETR = NOTE
IF	V	
JOE	blank	
EQUAL	glank	
'0'	L	
MOVE	V	
'1'	L	
TO	blank	
JOE	blank	
dummy	N (out-of-line &DATAWS)	4
dummy	A (AREA A indicator)	
77	L	
JOE	blank	
PIC	blank	
X	blank	
.	V	
dummy	N (out-of-line &END)	0
GO	V	
TO	blank	

Example:

```
SP   LIST &1(S) :
     &REPEAT
         .
         .
         .
       (Macro-code to process &1, either as received
       from the prototype or obtained by &GET.)
         .
         .
         .
         &GET &1
     &UNTIL &1'T = 'V'
         &OR &1'T = 'S'
         &OR &1'T = 'A'
         &STOW
     &ENDREP
```

In the above example, if a verb or period (Type attribute 'V'), a word that matches a String macro-name (Type attribute 'S'), or an Area A indicator (Type attribute 'A') is acquired by the &GET, the &STOW is not executed, the macro is terminated, and the word causing macro termination becomes the next word input to CA-MetaCOBOL+. In all other cases, the &STOW is executed. Any qualifiers and/or subscripts/indices following the word acquired by the &GET also are moved to &1. All words acquired by the &STOW are removed from the input stream. &ENDREP terminates &REPEAT.

Assume the following input stream:

```
LIST FIELD-A FIELD-B OF REC-A.
```

The words "LIST FIELD-A" match the prototype, and "FIELD-A" becomes the value of &1. After processing "FIELD-A", the &GET moves "FIELD-B" to &1. Since &1 has the value of a data-name, it fails the Type tests for verb/period, String name, and Area A indicator. The &STOW is executed, adding the qualifier "OF REC- A" to &1 and removing "FIELD-B OF REC-A" from the input stream. After processing "FIELD-B OF REC-A", the &GET acquires the word "." which remains in the input stream, matches the verb/period test, and the macro is ended.

5.9.2 &COPY

The &COPY directive copies text into the generated source program.

Format:

```
        {word          }
        {&Vname        }
&COPY   {&n            }
        {literal       }
        {concatenation}
```

&COPY
is a directive that is valid in the model of a String macro only and causes text from a COPY library to be output.

The operands described below must conform to the COBOL rules for member (MVS) or book (VSE) names, and name the library member or book to be copied. In MVS, SYSLIB and its concatenations are searched for the named member. In VSE, the COBOL Source Statement Library (public and private) are searched for the named book.

word
is a constant member or book name to be copied.

&Vname
is the name of a non-numeric variable containing the name of the member or book to be copied.

&n
is a symbolic operand &0 through &15. The first word of the current value is the member or book name.

literal
is a character string which may be a non-numeric literal containing the member or book name.

concatenation
is a construction within a macro model which combines one or more words into a single member or book name.

Note: After reaching the end of the member or book, macro processing resumes at the point immediately following the &COPY directive expression.

If the word following &COPY is enclosed in quotes, the quotes are stripped. Only the first eight characters of the resulting word are used as the member or book name.

The &COPY text is treated as program text. If &COPY text contains data definitions, attributes are stored. The only Translator-directing statement recognized within &COPY text is *$n.

Any outstanding &GET without an intervening &STORE/&STOW causes an error when an &COPY is executed.

Example: The following macro can be used to direct a copy member or book name to the Working-Storage Section of the PROCEDURE DIVISION.

```
SP   SAVE &1 :
          &DSTART
          &DATAW
          &COPY &1
          &END
          &DSTOP
```

5.10 Out-Of-Line Directives

Out-of-line directives are used to re-direct source or generated text to a location different from the point at which the macro call occured. This type of re-direction begins when an out-of-line directive is executed. The source text COBOL division is "saved" and the current COBOL division is updated to the division associated with the directive. In effect, when the divisions are different, the macros for the current division are de-activated and the macros in the division associated with the directive are activated. Text (words) output subsequent to the directive will match word and prefix macros that are active in the current COBOL division. Even though divisions may have changed, CA-MetaCOBOL+ remains sensitive to source text division headers.

Once a directive is executed, generated or source text is diverted to the out-of-line work file for processing during the "merge" phase. When more than one "text segment" is created for a single location, the merge result will depend on the order in which the text segments were created. The initial segment will merge first, followed by the next segment, and so on until there are no more segments to be merged, for a given location.

Normally, out-of-line processing ends when another out-of-line directive is executed or the macro terminates. Diversion of generated text to the out-of-line work file is discontinued and the source text COBOL division (and macros) is restored. However, if a &NOEND directive is executed prior to macro termination, out-of-line processing continues, and the source text COBOL division will not be restored until an &END directive is executed.

5.10.1 COBOL Out-Of-Line Directives

There are two directives that place text before or after the generated program. In addition, there are 11 directives that place text at pre-defined locations within the generated program; COBOL division headers must exist in the source text to support these locations. Text directed to any location that does not exist will appear on the LOST TEXT report, and cause CA-MetaCOBOL+ to terminate with a return code of 12.

Both types of directives are listed as follows in order by destination:

&ANTE — Places text ahead of the generated COBOL program. At least one COBOL division header must be present in the source text. The text is printed in the OUTPUT listing and included at the beginning of the PUNCHF file. Because this text will not be within the program, you may want to use "Line Output" to ensure that it is formatted correctly. The OUTPUT listing line will print a maximum of 110 characters; only the first 80 characters are included in the PUNCHF file. By placing asterisks (**) in the first two positions of your "Line Output", the line on the OUTPUT listing will print, but will be excluded from the PUNCHF file. By placing an asterisk and stroke (*/) in the first two positions, a page ejection of the OUTPUT listing will occur, and the line from the PUNCHF file will be excluded.

&ENV — Places text at the end of the ENVIRONMENT DIVISION. The ENVIRONMENT DIVISION header must be present in the source text. The text will appear following the final paragraph header of the division.

&DATAF Places text at the end of the FILE SECTION. The DATA DIVISION header must be present in the source text. The merge phase will insert a FILE SECTION header if it has been omitted.

&DATAWS Places text at the beginning of the WORKING-STORAGE SECTION. The DATA DIVISION header must be present in the source text. The merge phase will insert a WORKING-STORAGE SECTION header if it has been omitted.

&DATAW Places text at the end of the WORKING-STORAGE SECTION. The DATA DIVISION header must be present in the source text. The merge phase will insert a WORKING-STORAGE SECTION header if it has been omitted.

&DATAWX Places text after any text directed to &DATAW. If text was not directed to &DATAW, this directive places text at the end of the WORKING-STORAGE SECTION. The DATA DIVISION header must be present in the source text. The merge phase will insert a WORKING-STORAGE SECTION header if it has been omitted.

&DATAL Places text at the end of the LINKAGE SECTION. The DATA DIVISION header must be present in the source text. The merge phase will insert a LINKAGE SECTION header if it has been omitted.

&DATAR Places text at the end of the REPORT SECTION. The DATA DIVISION header must be present in the source text. The merge phase will insert a REPORT SECTION header if it has been omitted.

&DATA Places text at the end of the DATA DIVISION. The DATA DIVISION header must be present in the source text. The text will appear at the end of the last section of the division. The merge phase will not create any SECTION header.

&PROCS Places text at the end of the current procedure SECTION. The PROCEDURE DIVISION header must be present in the source text. The text will appear at the end of the SECTION within which the macro was called, or at the end of the first section of the PROCEDURE DIVISION if the macro was called within a division other than the PROCEDURE DIVISION.

&PROC Places text at the end of the PROCEDURE DIVISION. The PROCEDURE DIVISION header must be present in the source text. The text will appear at the end of the last or only SECTION of the division.

&PROCX Places text after any text directed to &PROCS or &PROC. If text was not directed to &PROCS or &PROC, this directive places text at the end of the PROCEDURE DIVISION. The PROCEDURE DIVISION header must be present in the source text. The text will appear after any text directed to &PROCS for the last section of the PROCEDURE DIVISION, or after any text directed to &PROC.

&POST Places text after the generated COBOL program. At least one COBOL division header must be present in the source program. The text is printed in the OUTPUT listing and included at the end of the PUNCHF file. Because this text will not be within the program, you may want to use "Line Output" to ensure that is formatted correctly. The OUTPUT listing line will print a maximum of 110 characters; only the first 80 characters are included in the PUNCHF file. By placing asterisks (**) in the first two positions of your "Line Output", the line on the OUTPUT listing will print, but will be excluded from the PUNCHF file. By placing an asterisk and stroke (*/) in the first two positions, a page ejection of the OUTPUT listing will occur, and the line from the PUNCHF file will be excluded.

5.10.2 AUXILIARY Out-Of-Line Directives

There are two directives that place text in the AUXILIARY File and/or Listing.

&AUX Places text in the AUXILIARY Listing and File. Because this text will not be within the program, you may want to use "Line Output" to ensure that it is formatted correctly. The AUXILIARY Listing line will print a maximum of 110 characters; only the first 80 characters are included in the AUXILIARY File. By placing asterisks (**) in the first two positions of your "Line Output", the line on the OUTPUT Listing will print, but will be excluded from the AUXILIARY File. By placing an asterisk and stroke (*/) in the first two positions, a page ejection of the OUTPUT Listing will occur, and the line will be excluded from the AUXILIARY File.

&AUXN Places text in the AUXILIARY File. Because this text will not be within the program, you may want to use "Line Output" to ensure that it is formatted correctly. Although "Line Output" has a maximum length of 110 characters, only the first 80 characters are included in the AUXILIARY File.

5.10.3 Excluding Directive

There is one directive which can be used to exclude source or generated text from the generated program.

Format:

```
&DUMMY
```

&DUMMY excludes text from the generated program and places it into a non-existent "Dummy" section of the DATA DIVISION. It changes the current division. It is most useful, when used together with the &DSTART directive to create entries in the TDS Attribute Table.

5.10.4 General Purpose Out-Of-Line Directives

There are two directives for "marking" unique locations within the generated program, the AUXILIARY Listing, and the AUXILIARY File, and "pointing" text to these "marked" locations.

&MARKER

The &MARKER directive marks a location where text can be pointed.

Format:

```
&MARKER {literal }
        {variable}
```

&MARKER
establishes a location within the generated text, the source text, or within text being directed to the AUXILIARY Listing or File, to which text may be pointed with the &POINT directive.

literal
is a numeric literal with a value of 0 - 4999.

variable
is the name of a variable with a value of 0 - 4999.

Note: The same &MARKER may be placed in two or more locations, which will cause the text directed by the matching &POINT to be duplicated in more than one location.

&POINT

The &POINT directive places text after a corresponding &MARKER.

```
&POINT {literal }
       {variable}
```

&POINT
places text after a &MARKER which has been assigned the same number.

literal
is a numeric literal with a value of 0 - 4999.

variable
is the name of a variable with a value of 0 - 4999.

Notes: &POINT does not change the current division. When using this directive to place code into a different division, it is advisable to precede it with an &ENV, &DATA, or &PROC directive, which will change the division and permit macros in the destination division to process the generated code. This may not be essential, but should be considered when using this directive to place data definitions in the DATA DIVISION. See &DSTART, which is also useful in this situation.

An &POINT can be issued before its matching &MARKER. Multiple &POINTs can be directed to the same &MARKER; text is placed in the output in the order issued.

When a &POINT directive refers to an undefined &MARKER, the text will print in the LOST TEXT report; it will not appear in the generated output.

Example: The following macros cause generation of "CLOSE file-name-1 file-name-2..." prior to the STOP RUN statement, where each file-name is defined in a SELECT statement in the ENVIRONMENT DIVISION.

```
SE   SELECT &1 :
         SELECT &1
         &PROC
         &POINT 999
         &1
         &END

SP   STOP RUN :
         CLOSE
         &MARKER 999
         STOP RUN
```

5.10.5 Ending Directive

There is one directive for explicitly terminating out-of-line text generation.

Format:

```
&END
```

&END
ends out-of-line text generation and restores the COBOL division if it was changed by the original out-of-line directive.

Example:

```
SE   SELECT &1 :
         SELECT &1
         &PROC
         &POINT 999
         &1
         &END

SP   STOP RUN :
         CLOSE
         &MARKER 999
         STOP RUN
```

5.10.6 Continuing Directive

There is one directive for continuing out-of-line text generation after macro termination.

Format:

```
&NOEND
```

&NOEND

continues out-of-line text generation after the macro terminates. This will cause source text following the original macro call to be directed to an out-of-line destination. If the COBOL division was changed by the out-of-line directive, it will remain changed when the macro terminates.

Example: CA-MetaCOBOL+'s structured programming macros permit relocation of PROCEDURE DIVISION code between START DATA and END DATA to the end of WORKING-STORAGE by macros similar to the following:

```
SP   START DATA :
         &DSTART
         &DATAW
         &NOEND

SD   END DATA :
         &END
         &DSTOP
```

Note that the END DATA macro is assigned to the DATA DIVISION.

5.11 Message Directive (&NOTE)

The message directive displays macro-defined messages on the INPUT & DIAGNOSTICS Listing and/or OUTPUT Listing. The &NOTE directive displays a macro-defined message.

Format:

```
        {&Vname       }
&NOTE   {&n           }
        {concatenation}
        {literal      }
```

&NOTE
is a directive that causes the operand to be printed as part of CA-MetaCOBOL+ message N99. The "text" is a maximum of 64 characters long, including bounding quotes. Additional characters are truncated. The &NOTE directive expression may appear in the model of a Word, Prefix, or String definition.

&Vname
is the name of a non-numeric or numeric variable.

&n
is a symbolic operand &0 through &15.

concatenation
is a construction within a macro model which combines one or more words into a single word.

literal
is a character string which may be a non-numeric literal, hexadecimal literal, or numeric literal (see the Glossary in Appendix A).

Notes: Only the first word of the current value of a symbolic operand is used. Thus, if a symbolic operand represents a qualified and/or subscripted name, only the first word of the name appears in this message.

A user diagnostic message can be output from a macro model by means of the &NOTE directive expression. This message appears on the INPUT Listing and/or the OUTPUT Listing depending on the NOTE option (see the CA-MetaCOBOL+ *User Guide*).

Execution of &NOTE sets the tentative condition code to 4.

The &NOTE directive output is not cancelled by the NOLISTIN and NOLISTOUT translate-time option or by the *$NOLIST Translator-directing statement.

When a NULL variable is output (&NOTE &V@VARIABLE), an apostrophe will print as the NOTE text, indicating no value.

Examples: The following prohibits the use of SORT by issuing an &NOTE:

```
WP    SORT : SORT
           &NOTE 'USE OF SORT NOT ALLOWED'
```

To introduce a variable word into a fixed text message, use the concatenation as follows:

```
           &NOTE &( 'FILE ' &VFILE ' DELETED' &)
```

5.12 Condition Code Directive (&COND)

The condition code directive is used to set a tentative condition code. The &COND directive defines the CA-MetaCOBOL+ Translator Condition Code.

Format:

```
&COND {&Vname }
      {literal}
```

&COND
is a directive that defines the tentative condition code to be passed from the Translator. The &COND directive executed with the highest value determines the ultimate completion code unless an &NOTE, &FLAG, or the Translator itself generates a higher code number.

Condition codes generated by the Translator include:

Code	Meaning
0	No diagnostics generated.
4	The highest level diagnostic is a macro-generated &FLAG or &NOTE.
8	The highest level diagnostic is CA-MetaCOBOL+ WARNING.
12	The highest level diagnostic is a CA-MetaCOBOL+ ERROR or Syntax Checker ERROR.
16	A CA-MetaCOBOL+ FATAL ERROR has occurred; the step is terminated prior to completion of the generation of an output source program.

Under VSE, the numeric value is stored in the Communication Section by the Translator as a binary number.

&Vname
is the name of a numeric variable in the range 0 - 255 (higher values are truncated).

literal
is a numeric literal (see the Glossary in Appendix A).

Example:

```
        .
        .
        .
&NOTE 'MNM01E - THIS IS BAD ERROR'
&COND 16
        .
        .
        .
```

5.13 Clock Directive (&CLOCK)

The clock directive is useful for determining the efficiency of macro code. The &CLOCK directive displays elapsed translate-time when used with the CLOCK option.

Format:

```
&CLOCK
```

&CLOCK
is a directive that, if specified with the CLOCK translate- time option when executing, displays a message on the INPUT & DIAGNOSTICS listing showing the elapsed translation time (see the CA-MetaCOBOL+ User Guide). By placing this directive in various macros, it is possible to find where, in the macro set, large amounts of time are being used and where a loop is being excessively repeated. If the CLOCK translate-time option is not specified, the &CLOCK directive is ignored.

Note: This directive is operative under MVS only. Under VSE, PC-DOS, and MS-DOS, it is ignored.

The &CLOCK directive is not affected by the NOLISTIN translate-time option or the *$NOLIST Translator-directing statement.

Example: The following "times" a macro loop.

```
        .
        .
        .
&REPEAT &UNTIL condition
    &CLOCK
        .
        .  (other processing)
        .
&ENDREP
```

5.14 Accounting Directive (&ACCT)

The accounting directive is useful for logging CA-MetaCOBOL+ macro use and for keeping track of application progress. The &ACCT directive creates an Accounting File record the first time this explicit directive is executed in a macro.

Format:

```
        {word          }
        {&Vname        }
&ACCT   {&n            }
        {concatenation}
        {literal       }
```

&ACCT
is a directive that creates an Accounting File record. Subsequent executions of the same &ACCT directive in the same execution of CA-MetaCOBOL+ are ignored.

The Accounting File is a separate CA-MetaCOBOL+ output data set consisting of 80-byte records in the format:

Position	Field
2-5	ACCT
7-44	first 38 characters of operand text excluding bounding quotes (additional characters are truncated)
46-53	current date
55-64	identification (from PROGRAM-ID; blank if no entry in program)
66-80	programmer (first word from AUTHOR; blank if no entry in program)

word
is a contiguous character string, not exceeding 30 characters.

&Vname
is the name of a non-numeric or numeric variable.

&n
is a symbolic operand &0 through &15.

concatenation
is a construction within a macro model which combines one or more words into a single word.

literal
is a character string which may be a non-numeric literal, hexadecimal literal, or numeric literal (see the Glossary in Appendix A).

Note: For more information, see the CA-MetaCOBOL+ *User Guide*.

Example: The following creates a record on the Accounting File to indicate the use of 'MACROSET', the name of the macro set including this macro.

```
SP   $PDE :
          &ACCT 'MACROSET'
```

5.15 Formatting Directives

Formatting directives permit the macro programmer to alter or enhance normal CA-MetaCOBOL+ output formatting.

5.15.1 &A

The &A directive forces a word output from the model of a macro to begin in Area A (column 8).

Format:

```
&A
```

&A
is a directive that is passed through the Translator as a "dummy" word. When the Source Generation segment of the Translator receives a dummy Area A word, it immediately outputs the previous line and prepares to align the subsequent output word at column 8 of a new line. The dummy word itself is not output. Further, if no macros are loaded that contain &INDENT/&OUTDENT directives, a period is forced at the end of the previous line.

Note: Consecutive Area A indicators passed to the output routine do not cause a series of blank lines. All Area A indicators after the first are ignored until a word is placed in the output line.

Example:

```
WD   -WS : &A WORKING-STORAGE SECTION.
```

Using &A allows greater model coding flexibility and is more compact than the following method. The following method begins the word in Area A of the macro model. This method is equivalent to &A but requires two records:

```
WD   -WS :
 WORKING-STORAGE SECTION.
```

5.15.2 &B

The &B directive cancels an Area A indicator and/or forces the next word to begin a new output record in Area B (columns 12-72).

Format:

```
&B
```

&B

is a directive that cancels an Area A indicator and/or forces the next word to begin a new output record in Area B (columns 12-72). If the &B directive is not immediately preceded by an Area A indicator, the current contents of the previous output line are output immediately and the &B directive prepares to place the subsequent output word in Area B of a new line. The directive &B itself is never output.

Note: Consecutive &B directives passed to the output routine do not cause a series of blank lines. All &B directives after the first are ignored until a word is placed in the output line.

Although the &B directive means that the next word output begins on a new line in Area B, it does not otherwise override the formatting rules in effect at translation time.

The example following illustrates the use of &B.

Example: As an example of &B usage, consider formatting the COBOL "GO TO...DEPENDING ON" statement so that each procedure-name in the list is placed on a separate line. The following GOSTACK macro uses &B to control such output formatting.

CA-MetaCOBOL+ Input

```
SP   GOSTACK &1 : GO TO
     &GET &2
     &REPEAT &UNTIL &2'T NE ' '
       &STOW
       &B &2
       &GET &2
     &ENDREP
     &B &( 'DEPENDING ON' &) &1
  .
  .
  .
 PROCEDURE DIVISION.
  .
  .
  .
     GOSTACK X1 PARA-A PARA-B PARA-C PARA-D.
  .
  .
  .
```

CA-MetaCOBOL+ Output

```
  .
  .
  .
 PROCEDURE DIVISON.
  .
  .
  .
     GO TO
      PARA-A
      PARA-B
      PARA-C
      PARA-D
      DEPENDING ON X1.
  .
  .
  .
```

5.15.3 &SETTAB

The &SETTAB directive defines a tab register as being equivalent to an absolute or relative column position.

Format:

```
&SETTAB treg = col
```

&SETTAB
is a directive for defining a tab register.

treg
is a tab register A0 through A15 for an absolute tab, or R0 through R15 for a relative tab.

=
is the separator (required).

col
is an integer or variable representing a column position 0 through 72.

Note: The default values of all tab registers are 0, causing no change in automatic formatting rules.

The &SETTAB directive can be "generated" out-of-line to control formatting of code based upon the longest name in a series, the MOVE statement with the longest sending operand, etc.

Example: Assume that all CALL operands are to be aligned vertically under the first, relative to the indentation of the CALL verb.

CA-MetaCOBOL+ Input

```
SP   CALL &1 USING &2(S) :
     CALL &1 USING &2
     &LOCAL &V-INDENT 9(2)
     &SET &V-INDENT = 12 + &1'N
     &SETTAB R0 = &V-INDENT
     &REPEAT
         &GET &2
     &UNTIL &2'T NE ' '
         &STOW
         &B
         &GOTAB R0
         &2
     &ENDREP
         .
         .
         .
     CALL SUBRTN USING A B C.
         .
         .
         .
```

CA-MetaCOBOL+ Output

```
         .
         .
         .
     CALL SUBRTN USING A
                       B
                       C.
         .
         .
         .
```

5.15.4 &GOTAB

The &GOTAB directive forces the following word to begin at an absolute or relative column position corresponding to the value of the "treg".

Format:

```
&GOTAB treg
```

&GOTAB
is a directive that applies only to the word following it.

treg
is a tab register A0 through A15 for an absolute tab, or R0 through R15 for a relative tab.

Notes: Relative column position is computed relative to the current left margin established by the automatic formatting or by the &INDENT/&OUTDENT commands.

If the &GOTAB cannot be satisfied on the current line, the current line is output and the &GOTAB attempted on the next line. The word may be left-shifted if it would cross into column 73.

The default values of all tab registers are 0. A value of 0 in the referenced tab register will cause &GOTAB to be ignored. A value for the column of less than 8 will cause &GOTAB to be ignored. A value for the column of more than 72 will cause the next word to be right-justified on a new line.

Example: Assume that all CALL operands are to be aligned vertically under the first, relative to the indentation of the CALL verb.

CA-MetaCOBOL+ Input

```
SP   CALL &1 USING &2(S) :
     CALL &1 USING &2
     &LOCAL &V-INDENT 9(2)
     &SET &V-INDENT = 12 + &1'N
     &SETTAB R0 = &V-INDENT
     &REPEAT
         &GET &2
     &UNTIL &2'T NE ' '
         &STOW
         &B
         &GOTAB R0
         &2
     &ENDREP
         .
         .
         .
     CALL SUBRTN USING A B C.
         .
         .
         .
```

CA-MetaCOBOL+ Output

```
         .
         .
         .
     CALL SUBRTN USING A
                       B
                       C.
         .
         .
         .
```

5.15.5 &MARGIN

The &MARGIN directive sets the value of the current left margin at an absolute or relative column position corresponding to the value of the "treg". It has no immediate effect as it controls the spacing of subsequent continuation lines.

Format:

```
&MARGIN treg
```

&MARGIN
is a directive that applies only to the new line following it. It is cancelled by any automatic or &INDENT/&OUTDENT format control.

treg
is a tab register A0 through A15 for an absolute tab, or R0 through R15 for a relative tab.

Notes: Relative column position is computed relative to the current left margin established by the automatic formatting or by the &INDENT/&OUTDENT commands.

The default values of all tab registers are 0. A value of 0 in the referenced tab register will cause &MARGIN to be ignored. A value for the column of less than 8 will cause &MARGIN to be ignored. A value for the column of more than 72 will cause the next word to be right-justified on a new line.

Example: Assume that all CALL operands are to be aligned vertically under the first, relative to the indentation of the CALL verb.

CA-MetaCOBOL+ Input

```
SP   CALL &1 USING &2(S) :
     CALL &1 USING &2
     &LOCAL &V-INDENT 9(2)
     &SET &V-INDENT = 12 + &1'N
     &SETTAB R0 = &V-INDENT
     &MARGIN R0
     &REPEAT
         &GET &2
     &UNTIL &2'T NE ' '
         &STOW
         &B
         &2
     &ENDREP
         .
         .
         .
     CALL SUBRTN USING A B C.
         .
         .
         .
```

CA-MetaCOBOL+ Output

```
         .
         .
         .
     CALL SUBRTN USING A
                       B
                       C.
         .
         .
         .
```

5.15.6 &INDENT/&OUTDENT

The &INDENT directive controls PROCEDURE DIVISION indentation and overrides all automatic PROCEDURE DIVISION formatting rules, and the &OUTDENT directive causes cancellation of the most recently issued &INDENT directive.

Format:

```
&INDENT
&OUTDENT
```

&INDENT
is a directive that can be issued only from String macros invoked during PROCEDURE DIVISION translation.

Each &INDENT directive causes the subsequent output line to be indented by a single level of indentation. If the indentation so specified exceeds the maximum level of indentation defined for PROCEDURE DIVISION formatting, indentation is to the maximum level defined until sufficient &OUTDENT directives are issued to bring indentation to the left of the maximum level defined.

&OUTDENT
is a directive that can be issued only from String macros invoked during PROCEDURE DIVISION translation.

If the indentation so specified is to the left of column 12, alignment to column 12 occurs until sufficient &INDENT directives are issued to cause indentation to the right of column 12.

Notes: The &INDENT and &OUTDENT directives are used with formatting to position the next Area B output line.

A word output in Area A resets indentation to column 12 until issuance of a subsequent &INDENT or &OUTDENT directive.

If an &INDENT and/or &OUTDENT directive is encountered during macro loading, all PROCEDURE DIVISION formatting must be under String-type macro control. &B directives are used to cause placement of the subsequent output word on a new line, and &INDENT and &OUTDENT directives are used to control indentation.

Example: The following is similar to structured programming formatting macros.

CA-MetaCOBOL+ Input

```
SP   IF :
     IF
     &INDENT

SP   ELSE :
     &OUTDENT
     &B
     ELSE
     &B
     &INDENT

SP   ENDIF :
     &OUTDENT
     &B
     &ENDIF

SP   $-VERB :
     &GET &1
     &IF &1 NOT = '.'
       &B
     &ENDIF
     .
     .
     .
IF A = B MOVE C TO D ELSE MOVE D TO C ENDIF
     .
     .
     .
```

CA-MetaCOBOL+ Output

```
     .
     .
     .
     IF A = B
          MOVE C TO D
     ELSE
          MOVE D TO C
     ENDIF
     .
     .
     .
```

5.16 Data Structure Analysis Directives

Data structure analysis directives are used to analyze the relationships between data items.

5.16.1 &SCAN

The &SCAN directive retrieves data-names in order of definition.

Format:

```
        { &Vname-1  }  { &Vname-2  }
&SCAN   { &n-1      }  { &n-2      }
        { section-1 }  { section-2 }
```

&SCAN
is a directive that retrieves the data-names, attributes, and structural relationships of items in the output source program DATA DIVISION.

&Vname-1, -2
is the name of a non-numeric variable.

&n-1, -2
is a symbolic operand &1 through &15.

&0 may not be specified in the &SCAN directive expression. &0 has a value only after execution on an &SCAN, &SCANC, &SCANF, or &SCANI.

section-1, -2
are the section-names File, Working-Storage, Linkage, Communication, or the pseudo-section PROCEDURE.

DATA DIVISION section headers, the pseudo-section PROCEDURE, FD's, SD's, and CD's are also returned by &SCAN. Such items have a Size attribute of 0 and a Level attribute of ' ' for section and pseudo-section headers, and a Level attribute of '00' for FD's, SD's, and CD's.

The Report Section and its subordinate items are not returned to &0 by the &SCAN.

Notes: The purpose of the &SCAN directive is to individually acquire data items within a contiguous range (the processing scope) in the DATA DIVISION while processing the PROCEDURE DIVISION. Item-1 is the "from" item, beginning the range of items to be acquired from the DATA DIVISION. Item-2 is the "thru" item, ending the range of items to be acquired from the DATA DIVISION. Both item-1 and item-2 may refer to the same data item.

The model must be coded as a processing loop. Each time that the same &SCAN is encountered in the loop, the next consecutive data-name in the DATA DIVISION becomes the current value of the special symbolic operand &0, which can be referenced in the model coding to access the result of the &SCAN execution. The &0 coding indicates what to do with the data item returned from the &SCAN execution. The list of items which are passed consecutively to &0 starts with item-1 and ends with the last elementary item of item-2, excluding all FILLER and level-88 items, but including level-66 items. After the specified &SCAN list is exhausted, the last execution of this &SCAN directive sets the ENDSCAN state condition to the true state and &0 is not available (the "at end" condition).

The specification of item-1 may contain a subscript which is returned in &0 on the first execution of the &SCAN (i.e., the first execution of the &SCAN directive expression will place item-1 in &0). If the specification of item-2 includes a subscript, the subscript is ignored by &SCAN. All values assigned to &0 via the &SCAN directive expression, except item-1, contain all qualifiers connected by the word "IN" if the data-name is not unique. If unique, no qualifiers are returned in &0.

If item-2 is defined sequentially before item-1 in the DATA DIVISION, execution of the macro model terminates and a diagnostic message is produced.

An &SCAN must not be initiated until the previous &SCAN, &SCANC, or &SCANI is "closed" by reaching the end of the &SCAN, &SCANC, or &SCANI list or executing an &SCANX directive. An &SCAN cannot be nested within the loop of another &SCAN, &SCANC, or &SCANI.

Items subordinate to a COPY or -INC are available to the &SCAN even if the COPY or -INC option is not specified, even though the subordinate items do not appear in the output program.

An &SCAN of a DATA DIVISION section does not return items generated out-of-line to the DATA DIVISION even if &DSTART is active; however, an &SCAN explicitly naming the pseudo-section PROCEDURE or an item generated out-of-line to the DATA DIVISION while &DSTART is active does return such an item. The pseudo-section PROCEDURE contains all items generated out-of-line to the DATA DIVISION while &DSTART is active, in the order of generation.

Example: The macro in the following example MOVEs SPACES to a group item and MOVEs ZEROS to each numeric elementary item of that group.

CA-MetaCOBOL+ Input

```
      .
      .
      .
SP    INIT &1(Q) :
      MOVE SPACES TO &1
      &REPEAT
          &SCAN &1 &1
      &UNTIL ENDSCAN
          &IF &0'U GE '0'
              MOVE ZEROS TO &0
          &ENDIF
      &ENDREP
      .
      .
      .
 DATA DIVISION.
      .
      .
      .
 01   CUSTOMER-RECORD.
      02  NAME           PICTURE X(30).
      02  ADDRESS        PICTURE X(30).
      02  CITY           PICTURE XX.
      02  ZIP            PICTURE 9(5).
      02  ACCOUNT-INFO   USAGE COMPUTATIONAL-3.
        03  PREVIOUS-BAL PICTURE S9(5)V99.
        03  CURRENT-BAL  PICTURE S9(5)V99.
      .
      .
      .
 PROCEDURE DIVISION.
      .
      .
      .
      INIT CUSTOMER-RECORD.
      .
      .
      .
```

CA-MetaCOBOL+ Output

```
      MOVE SPACES TO CUSTOMER-RECORD
      MOVE ZEROS TO ZIP
      MOVE ZEROS TO PREVIOUS-BAL
      MOVE ZEROS TO CURRENT-BAL.
```

5.16.2 &SCANA

The &SCANA directive retrieves data item names for a specified TDS relative address.

Format:

```
          {&Vname }
&SCANA    {&n     }
          {literal}
```

&SCANA
is a directive for retrieving a data item name for a specified TDS relative address.

&Vname
is the name of a non-numeric variable containing a TDS relative address.

&n
is a symbolic operand &1 through &15 containing a TDS relative address.

literal
is a character string which may be a non-numeric literal, hexadecimal literal, or numeric literal (see the Glossary in Appendix A) containing a TDS relative address.

Notes: Execution of this directive returns the data name, including all qualifiers, corresponding to the TDS relative address in &0.

When no data-name exists for the TDS relative address, the ENDSCAN state condition is set true and &0 is not available (the "at end" condition).

Execution of an &SCANX is not necessary as it is with other forms of &SCAN.

Example: To "scan" all of TDS:

```
&LOCAL &V99999 9(5)
&SET &V99999 = 0
&REPEAT
    &SET &V99999 = &V99999 + 1
    &SCANA &V99999
&UNTIL ENDSCAN
    &SELECT &0
    &WHEN . . .
        .
        .
        .
    &ENDSEL
&ENDREP
```

5.16.3 &SCANC

The &SCANC directive retrieves condition-name of &n and places the condition-name in &0.

Format:

```
&SCANC &n
```

&SCANC
is a directive that retrieves the condition-names associated with the referencing items in the output source program DATA DIVISION.

&n
is a symbolic operand &1 through &15.

Notes: This directive expression may be executed in a loop. The first and subsequent executions of this expression return the first and subsequent condition-names in &0 associated with the data item identifier in &n. This directive may be executed repeatedly. If the definition of the data item named in &n does not contain a subordinate level-88 item or the directive is executed more times than there are condition-names associated with the identified data-name, the ENDSCAN state condition is set true and &0 is not available (the "at end" condition).

Execution of &SCANX subsequent to the execution of &SCANC ends the processing of the current list of condition-names if it has not been terminated by the "at end" branch.

An &SCANC cannot be executed while an &SCANI is outstanding, but can be executed within the processing scope of an &SCAN.

Items subordinate to a COPY or -INC, which do not appear in the output source program only because the COPY or -INC translate-time option is not specified, are available to the &SCANC.

Example:

```
DATA DIVISION
01   TRANSACTION-TYPE           PIC X.
     88  ADD-TRANSACTION        VALUE 'A'.
     88  DELETE-TRANSACTION     VALUE 'D'.
     88  UPDATE-TRANSACTION     VALUE 'U'.
```

Given the previous data item, TRANSACTION-TYPE, the following could be used to retrieve the subordinate level-88 condition- names.

```
&EQU &1 'TRANSACTION-TYPE'
&REPEAT
    &SCANC &1
&UNTIL ENDSCAN
    &SELECT &0
    &WHEN 'ADD-TRANSACTION'
        .
        .
        .
    &WHEN 'DELETE-TRANSACTION'
        .
        .
        .
    &WHEN 'UPDATE-TRANSACTION'
        .
        .
        .
    &ENDSEL
&ENDREP
```

5.16.4 &SCANF

The &SCANF directive retrieves the next higher data-name in the data hierarchy from &n, and places the name in &0.

Format:

```
&SCANF &n
```

&SCANF
is a directive that retrieves the parent of a data item in the output source program DATA DIVISION.

&n
is a symbolic operand &0 through &15.

Notes: If &n is a subordinate data item, the name of the next higher group item containing it is returned in the special symbolic operand &0. If &n is an index-name, the name of the first data item defining the index is returned in &0. If &n is a condition-name, the name of the data item to which the condition-name is associated is returned in &0. If &n is not subordinate to another data item and is not an index- or condition-name, the ENDSCAN state condition is set true and &0 is not available (the "at end" condition).

Section names are not returned by &SCANF.

Items subordinate to a COPY or -INC, which do not appear in the output program only because the COPY or -INC option is not specified, are available to the &SCANF.

Example: To acquire successively higher levels in a data hierarchy, &0 must first be "primed" by an &SCAN-type directive where &n is specified as &1 through &15. Once primed, successive &SCANFs using &0 are used to loop through the hierarchy.

```
&REPEAT
    &SCANF &1
&UNTIL ENDSCAN &OR &0'L = '00'
    &EQU &1 &0
&ENDREP
```

5.16.5 &SCANI

The &SCANI directive retrieves indices of &n and places index-names in &0.

Format:

```
&SCANI &n
```

&SCANI
is a directive that retrieves the index-names declared for referencing items in the output source program DATA DIVISION.

&n
is a symbolic operand &1 through &15.

Notes: This directive expression may be executed in a loop. The first and subsequent executions of this expression return the first and subsequent indices in &0 associated with the data item named in &n. This directive may be executed repeatedly. If the definition of the data item named in &n does not contain an "INDEXED BY---" phrase or the directive is executed more times than there are indices named in the phrase, the ENDSCAN state condition is set true and &0 is not available (the "at end" condition).

Execution of &SCANX subsequent to the execution of &SCANI ends the processing of the current list of indices if it has not been terminated by the "at end" branch.

An &SCANI cannot be executed while an &SCANC is outstanding, but can be executed within the processing scope of an &SCAN.

Items subordinate to a COPY or -INC, which do not appear in the output source program only because the COPY or -INC translate-time option is not specified, are available to the &SCANI.

Example: The following macros "prime" and "increment" the index associated with a data item or a common subscript, as appropriate.

```
SP   $PRIME &1 :
         &SCANI &1
         &IF ENDSCAN
             MOVE 1 TO COMMON-SS
         &ELSE
             SET &0 TO 1
         &ENDIF


SP   $INCR &1 :
         &SCANI &1
         &IF ENDSCAN
             ADD 1 TO COMMON-SS
         &ELSE
             SET &0 UP BY 1
         &ENDIF
```

5.16.6 &SCANX

The &SCANX directive terminates &SCAN, &SCANC, and &SCANI directive expressions, and resets those expressions to be executed again.

Format:

```
&SCANX
```

&SCANX
is a directive for terminating &SCAN, &SCANC, or &SCANI. Once the &SCANX is issued, &0 is no longer available. If the last item in &0 is to be used after execution of the &SCANX, it should be stored in variables or equated to another symbolic operand.

If no &SCAN, &SCANC, or &SCANI is outstanding, execution of an &SCANX has no effect.

Example: The following logic can be used to obtain the first record-name associated with a file-name, but proceed no further down the data structure.

```
     .
     .
     .
&IF &1'L = '00'
    &REPEAT
        &SCAN &1 &1
    &UNTIL ENDSCAN
        &IF &0'L = '01'
           &EQU &2 &0
           &SCANX
           &ESCAPE
        &ENDIF
    &ENDREP
&ENDIF
     .
     .
     .
```

5.16.7 &DSTART/&DSTOP

The &DSTART directive defines the start of data attribute building for data item definitions directed out-of-line, and is terminated by &DSTOP.

Format:

```
{&DSTART}
{&DSTOP }
```

&DSTART
is a directive that defines the start of data attribute building for data item definition directed out-of-line.

&DSTOP
is a directive that terminates &DSTART.

Notes: &DSTART and &DSTOP are not, in themselves, out-of-line directives. They define the start and end of data attribute building for items directed out-of-line to the DATA DIVISION. Items generated out-of-line to the DATA DIVISION while &DSTART is active are available for attribute testing and can be acquired by the &SCAN-type directives. Executing &DSTOP cancels &DSTART.

Passing incomplete data hierarchies to the DATA DIVISION under &DSTART may result in invalid and/or misleading attributes and item relationships.

Note that, once initiated, &DSTART remains active until terminated by &DSTOP, even if a macro error occurs.

Execution of an &DSTOP with no outstanding &DSTART is ignored.

Example: CA-MetaCOBOL+'s structured programming macros permit relocation of data descriptions following the PROCEDURE DIVISION statement START DATA to the end of Working-Storage, and make the items subject to attribute analysis and scanning. The END DATA macro stops out-of-line generation and data attribute building.

```
SP   START DATA :
          &DSTART
          &DATAW
          &NOEND

SD   END DATA :
          &END
          &DSTOP
```

6. Input Exit Facility

CA-MetaCOBOL+'s Input Exit Facility allows:

- input to be passed to CA-MetaCOBOL+ from non-standard sources
- CA-MetaCOBOL+ input to be processed by subprograms

The following sections describe:

- how to invoke an input exit program
- the parameters that interface the input exit program and CA-MetaCOBOL+
- how to effect this interface in Assembler and COBOL
- how to use ADRXIXIT, an input exit loader that enables up to ten input exit programs to be invoked in a single translation

6.1 How to Invoke an Input Exit Program

Three steps are required to successfully execute input exit programs:

- specify the IXIT= translate-time option
- invoke the subprogram with an *$CALL translator-directing statement or an &CALL directive expression
- satisfy the MVS, VSE, VM/CMS, or PC operating system requirements

6.1.1 IXIT= Translate-time Option

The IXIT= translate-time option identifies the input exit program to the Translator. It has the following format:

Format:

```
IXIT=module-name
```

~~IXIT~~
can be abbreviated as IX. If the IXIT= translate-time option is not specified, no input exit program can be invoked.

module-name
is the 1-8 character name of an input exit load module.

6.1.2 *$CALL and &CALL

The *$CALL translator-directing statement invokes an input exit program. *$CALL can be used to invoke an input exit program that retrieves macros or source records that cannot be retrieved with *$COPY, *$LIBED, or *$LIBET.

The &CALL directive invokes an input exit program from within a String-type macro. A macro that invokes an input exit program with &CALL may continue or terminate a translation depending on the value of a parameter returned to it from the input exit program. The input exit program also may return data to the invoking macro without returning data to CA-MetaCOBOL+ as primary input.

*$CALL Statement

Format:

```
*$CALL word [commentary]
```

*****
must be placed in the continuation column (column 7).

word
is a character string of 1 to 56 characters that is passed to the input exit program. Specify the word either as a contiguous character string or as a non-numeric literal enclosed in apostrophes or quotation marks.

The contents of the character string are defined by the input exit program. Even if the input exit program requires no data, a character string must be specified.

commentary
can be any text normally placed within a comment record, but it must be separated from the word by at least one space. However, the *$CALL statement cannot be continued to the next record and may be specified in the Primary Input File only. If records passed to CA-MetaCOBOL+ with the *$CALL statement contain *$COPY, *$LIBED, *$LIBET, *$COL, or *$CALL translator-directing statements, they are treated as commentary; COBOL COPY statements, however, are processed.

Notes: Input source retrieved with *$CALL will match String, Word, and Prefix macros. However, &CALL cannot be used concurrently with *$CALL.

Example: Consider the following example where FETCH is an input exit program that verifies and retrieves the macro set DBMACROS identified in the *$CALL statement.

```
OPTION IXIT=FETCH
     .
     .
     .
*$CALL DBMACROS
IDENTIFICATION DIVISION.
     .
     .
     .
```

&CALL Directive

The &CALL directive invokes an input exit program from within the model of a String-type macro.

Format:

```
        {word          }
        {&Vname        }
 &CALL  {&n            }
        {literal       }
        {concatenation}
```

word
a character string of 30 characters or less.

&Vname
a numeric or non-numeric variable.

&n
any symbolic operand from &1 through &15, or &0. If &n contains multiple words, only the first operand is passed to the input exit program.

concatenation
a construction within a macro model that combines one or more words into a single word. It must specified in the "&(" form.

literal
a numeric or non-numeric literal. If the operand is defined as a non-numeric literal, delimiting quotation marks are not passed to the input exit program.

The &CALL parameters -- word, &Vname, &n, literal, or concatenation -- represent the data passed to the input exit program and must be 1-128 characters in length. The contents are defined by the input exit program. The &CALL operand is required even when the input exit program does not require any parameters.

Notes: An &GET must be resolved with an &STOW or &STORE directive before executing an &CALL.

The macro executing &CALL cannot process the primary input (if any) returned by the input exit. Word- and Prefix-type macro calls, however, will process this input.

Translator-directing statements encountered within input obtained with &CALL are treated as comments. Conventional COBOL COPY statements are ignored in input obtained via an &CALL.

&SETR Directive

On the completion of a call, the input exit program can return up to 128 characters of data to a CA-MetaCOBOL+ macro. This character-string is acquired by the following &SETR directive expression.

Format:

```
&SETR &Vname = IXIT
```

&Vname
is a non-numeric variable of up to 128 characters in length.

For example, a String macro using &CALL can pass data base access statements encountered in the CA-MetaCOBOL+ input to an input exit program which, in turn, verifies that any references to dataviews, records, or field names are defined in the data base. On the basis of a control value returned by the input exit program, the macro can respond by issuing a warning message, placing additional statements in the output, and/or setting the condition code.

Example: The following example illustrates a macro that checks for the presence of valid data base control information. The input exit program 'CHECKCB' is invoked with an &CALL to determine the presence of information known as 'DBCTLBLK'. If 'DBCTLBLK' is present, the input exit program returns 'OK' and the macro terminates. Otherwise, a warning message is issued and the condition code is set to 16.

```
OPTION IXIT=CHECKCB
         .
         .
         .
       SP  CHECK FOR &1 :
                 &LOCAL &VRETURN XX
                 &CALL &1
                 &SETR &VRETURN = IXIT
                 &IF &VRETURN = 'OK' &T$
                 &NOTE &( 'WARNING - ' &1 ' CONTROL MISSING' &)
                 &COND 16
         .
         .
         .
        PROCEDURE DIVISION.
         .
         .
         .
            CHECK FOR DBCTLBLK
         .
         .
         .
```

An alternative would be to have the input exit program issue the message and set the condition-code directly.

6.1.3 Operating System Considerations

Under MVS, the input exit program must be available in STEPLIB, JOBLIB, or LINKLIB. The module name specified in the IXIT= translate-time option is the module-name for loading the input exit program.

Under DOS/VSE, the input exit program must be available in the core image library. Under all VSE systems, the module name specified in the IXIT= translate-time option is the PHASE name. The input exit program must be self-relocating or the system loader must be a relocating loader. No partition space outside the input exit program phase boundary is available to the input exit program, with the exception of space acquired via the GETVIS function.

Under VM/CMS, the input exit program must be available as a TEXT file. If a GLOBAL TXTLIB is in effect, the input exit program also may be a member of the appropriate TXTLIB. The name specified in the IXIT translate-time option is the name used to INCLUDE the input exit program. A search of TEXT files and GLOBALed TXTLIBs to resolve any undefined references is automatically performed.

Under MS-DOS or PC-DOS, the input exit program must be available in the .DLL Library.

For additional information on using CA-MetaCOBOL+ under MVS, VSE, and CMS, refer to the CA-MetaCOBOL+ *User Guide*. For additional information on using CA-MetaCOBOL+/PC using MS-DOS or PC-DOS, refer to the CA-MetaCOBOL+/PC *User Guide*.

6.2 Input Exit Call Parameters

This section identifies and describes the data passed between CA-MetaCOBOL+ and the input exit program.

1) **Call Work area**: This area is used to communicate data between CA-MetaCOBOL+ and the input exit program. It consists of the following fields:

   a) A binary halfword (two bytes) containing the length of the parameter data.

   b) Up to 128 characters of the data specified in the *$CALL translator-directing statement or the &CALL directive. This field is left-aligned and variable in length. (The input exit program may also use this field to return data to a macro.)

2) **Card Image**: An 80-byte card image returned to CA-MetaCOBOL+ by the input exit program. This record may contain input to CA-MetaCOBOL+ or a diagnostic message. (The input exit program specifies the contents of this field with the interface status field, which is described below.)

3) **Input Exit Control Block**: This four-byte area consists of the following three fields.

   a) Input Exit Program Status: a one-byte field set by the input exit program to indicate whether CA-MetaCOBOL+ is to return control to the input exit program. This byte is checked by CA-MetaCOBOL+ and may contain the following values:

      Y indicates that the CA-MetaCOBOL+ Translator is to return control to the input exit program.

      N indicates that the input exit program has completed the services requested of it (i.e., the "at end" condition).

      If neither Y nor N is specified, N is assumed.

   b) Interface Status: A one-byte field that is set 1) by CA-MetaCOBOL+ when it calls the input exit program and 2) by the input exit program when it returns control to CA-MetaCOBOL+.

      CA-MetaCOBOL+ places the following values in this field prior to calling the input exit program:

      * indicates that the input exit program has been invoked by a *$CALL translator-directing statement.

      & indicates that the input exit program has been invoked by an &CALL directive.

Z indicates that CA-MetaCOBOL+ is terminating. The first two call parameters (i.e., 1 and 2 above) are unaddressable by the Input Exit And Facility and will not be processed by CA-MetaCOBOL+. The input exit program must complete final housekeeping, such as closing files that are used by the input exit program.

If the interface status is * or &, the input exit program must place one of the following values in the interface status field before returning control to CA-MetaCOBOL+. These values identify the contents of the card image containing the data returned by the input exit program:

P the card image contains an input record to be processed by CA-MetaCOBOL+ that is also to be printed on the Input listing.

R the card image contains an input record to be processed by CA-MetaCOBOL+ that is not to be printed unless the LISTALL translate-time option is specified.

D the card image contains the text of an advisory message defined by the input exit program in the first 64 bytes. The message is assigned number "H79".

F the card image contains the text of a FATAL ERROR message in the first 40 bytes. CA-MetaCOBOL+ will perform a final call to the input exit with the one-byte control character set to 'Z' (see explanation of 'Z' above). The message is assigned number "H78".

N the card image data contains no data.

c) Return Code: If the interface status field contains a D, this field may contain a binary halfword with a value of 0-255, which is assigned to the advisory message. This value becomes the condition code for a translation, unless the condition code is set to a higher value by CA-MetaCOBOL+ or by a macro.

6.3 COBOL and Assembler Interface Conventions

The interface between CA-MetaCOBOL+ and the input exit program uses the parameters described in Section 6.2. The interface conventions for Assembler and COBOL programs are outlined in this section.

6.3.1 COBOL Interface Conventions

CA-MetaCOBOL+ uses standard IBM COBOL linkage conventions to invoke an input exit program, requiring simply that parameters be defined in the Linkage Section and that the program contain an ENTRY or PROCEDURE DIVISION statement USING parameter. For example, the following represents the minimum parameters required to support the input exit interface in COBOL:

```
LINKAGE SECTION.
01 WORKAREA.
   02 WORKAREA-LENGTH             PIC S9(4) COMP SYNC.
   02 WORKAREA-BUFFER             PIC X(128).
01 CARD-IMAGE                     PIC X(80).
01 IXIT-CONTROL-BLOCK.
   02 IXIT-PROGRAM-STATUS         PIC X.
   02 IXIT-INTERFACE-STATUS       PIC X.
   02 IXIT-RETURN-CODE            PIC S9(4) COMP SYNC.
      .
      .
      .
PROCEDURE DIVISION USING          WORKAREA
                                  CARD-IMAGE
                                  IXIT-CONTROL-BLOCK.
      .
      .
      .
```

Refer to Section 6.2 for a description of the input exit call parameters.

When the input exit program is called, if IXIT-INTERFACE-STATUS contains either a '*' or '&', the input exit program can return a record or message to CA-MetaCOBOL+ by moving the contents of the record or message in CARD-IMAGE and setting the proper field in IXIT-INTERFACE-STATUS.

Before returning control to CA-MetaCOBOL+, the input exit program must indicate whether CA-MetaCOBOL+ is to return control to the input exit program. This is done by moving 'Y' or 'N' into IXIT-PROGRAM-STATUS.

When IXIT-INTERFACE-STATUS contains a 'Z', the input exit program must perform final housekeeping. The input exit program cannot ask that CA-MetaCOBOL+ return control, and it cannot return data in the CARD-IMAGE or WORKAREA-BUFFER fields.

Input Exit Example

The following program is an input exit program written using CA-MetaCOBOL+ Structured Programming Language. The purpose of this program is to surround CICS macros encountered in the source input with the *$NOMCT and *$MCT translator-directing statements. In this way, CA-MetaCOBOL+ will ignore any CICS macros in the input source program, because *$NOMCT and *$MCT, respectively, cause CA-MetaCOBOL+ to ignore and process input source statements.

This input exit program is invoked with *$CALL, and it does not require any parameters. The CALL-CONTROL module checks the interface status parameter IXIT-CB-INTERFACE-STATUS to verify that the input exit has been invoked with *$CALL only. If not, the call is terminated; i.e., 'N' is moved to the status parameter IXIT-CB-PROGRAM-STATUS, which specifies that CA-MetaCOBOL+ is not to return control to the input exit program. If the input exit program is invoked with &CALL, a diagnostic message is returned to CA-MetaCOBOL+ and the condition code for the translation is set to 16.

The PROCESS-VALID-CALL module performs end-of-file processing. The PROCESS-RECORD module determines which of the low-level modules -- LOOK-FOR-DFH, PROCESS-DFH, or PROCESS-MCT -- are performed, based on the value of STATUS-OF-INPUT.

In the module LOOK-FOR-DFH, the program searches each record in the file CA-METACOBOL+-INPUT for the character string "DFH". If a match is found, a record containing *$NOMCT is moved to column 7 of the IXIT-CARD-IMAGE field.

PROCESS-DFH passes macro-level CICS statements that are continued to the next line as input to CA-MetaCOBOL+ through the IXIT-CARD-IMAGE field. This paragraph continues processing until the last line of the DFH macro is detected.

Control is passed to PROCESS-MCT after PROCESS-DFH has processed the last DFH record. This module moves *$MCT to column 7 of the IXIT-CARD-IMAGE, which is passed to CA-MetaCOBOL+. PROCESS-MCT then resets the STATUS-OF-INPUT to "PROCESSING COBOL" so that processing will continue with the LOOK-FOR-DFH module when control is returned to the input exit program.

After one of the low-level modules is performed, control returns to CA-MetaCOBOL+ so that it can retrieve the record moved into the IXIT-CARD-IMAGE field. If 'Y' has been moved into IXIT-CB-PROGRAM-STATUS, the input exit program is reentered in the top-most module (CALL-CONTROL), and if an end-of-file condition does not exist, control returns to the module referenced by STATUS-OF-INPUT.

```
      *$LIBED SPP
       IDENTIFICATION DIVISION.
       PROGRAM-ID. IXITCICS.
       ENVIRONMENT DIVISION.
       CONFIGURATION SECTION.
       SOURCE-COMPUTER. IBM-370.
       OBJECT-COMPUTER. IBM-370.
       INPUT-OUTPUT SECTION.
       FILE-CONTROL.
           SELECT CA-METACOBOL-INPUT ASSIGN TO UT-S-SYSIXIT.
       DATA DIVISION.
       FILE SECTION.
       FD  CA-METACOBOL-INPUT     RECORDING MODE F
                                  BLOCK CONTAINS 0 RECORDS
                                  LABEL RECORD IS STANDARD
                                  DATA RECORD IS
CA-METACOBOL-INPUT-RECORD.
       01  CA-METACOBOL-INPUT-RECORD.
           02 FILLER              PIC X(71).
           02 DFH-CONTINUATION    PIC X.
           02 FILLER              PIC X(8).
       WORKING-STORAGE SECTION.
       01  FLAG                   RECORD-AVAILABLE IS FALSE.
       01  FLAG                   FIRST-CALL IS TRUE.
       01  STATUS-OF-INPUT        PIC 9 VALUE 1.
           88 PROCESSING-COBOL    VALUE 1.
           88 PROCESSING-DFH      VALUE 2.
           88 PROCESSING-MCT      VALUE 3.
       LINKAGE SECTION.
       01  IXIT-WORKAREA.
           02 IXIT-WORKAREA-LENGTH     PIC S9(4) COMP SYNC.
           02 IXIT-WORKAREA-BUFFER.
              03 IXIT-WORKAREA-BYTE    OCCURS 128 PIC X.
       01  IXIT-CARD-IMAGE             PIC X(80).
       01  IXIT-CB.
           02 IXIT-CB-PROGRAM-STATUS   PIC X.
           02 IXIT-CB-INTERFACE-STATUS PIC X.
           02 IXIT-CB-RETURN-CODE      PIC S9(4) COMP SYNC.
      /
       PROCEDURE DIVISION USING   IXIT-WORKAREA
                                  IXIT-CARD-IMAGE
                                  IXIT-CB
```

```
      *
       CALL-CONTROL.
           SELECT FOR IXIT-CB-INTERFACE-STATUS
           WHEN '*'
              DO PROCESS-VALID-CALL
           WHEN '&'
              MOVE 'N' TO IXIT-CB-PROGRAM-STATUS
              MOVE 'D' TO IXIT-CB-INTERFACE-STATUS
              MOVE '&CALL NOT SUPPORTED BY IXITCICS' TO
IXIT-CARD-IMAGE
              MOVE 16 TO IXIT-CB-RETURN-CODE
           WHEN NONE ARE SELECTED
              MOVE 'N' TO IXIT-CB-PROGRAM-STATUS
              MOVE 'N' TO IXIT-CB-INTERFACE-STATUS
           ENDSELECT
           GOBACK
      /
       PROCESS-VALID-CALL.
           IF RECORD-AVAILABLE
              DO PROCESS-RECORD
           ELSE
              IF FIRST-CALL
                 OPEN INPUT CA-METACOBOL-INPUT
                 SET-FALSE FIRST-CALL
              ENDIF
              READ CA-METACOBOL-INPUT
              IF END
                 CLOSE CA-METACOBOL-INPUT
                 SET-FALSE RECORD-AVAILABLE
                 SET-TRUE FIRST-CALL
                 MOVE 'N' TO IXIT-CB-PROGRAM-STATUS
                 IF PROCESSING-COBOL
                    MOVE 'N' TO IXIT-CB-INTERFACE-STATUS
                 ELSE
                    MOVE 1 TO STATUS-OF-INPUT
                    MOVE 'P' TO IXIT-CB-INTERFACE-STATUS
                    MOVE '      *$MCT' TO IXIT-CARD-IMAGE
                 ENDIF
              ELSE
                 SET-TRUE RECORD-AVAILABLE
                 DO PROCESS-RECORD
              ENDIF
           ENDIF
      /
```

```
       PROCESS-RECORD.
           SELECT FIRST ACTION
           WHEN PROCESSING-COBOL
              DO LOOK-FOR-DFH
           WHEN PROCESSING-DFH
              DO PROCESS-DFH
           WHEN PROCESSING-MCT
              DO PROCESS-MCT
           ENDSELECT
      /
       LOOK-FOR-DFH.
           START DATA
           01 SEARCH-BLOCK.
              02 SEARCH-BYTE      PIC X(1) OCCURS 22 INDEXED BY
SX.
           01 KEYWORD-BLOCK.
              02 KEYWORD-AREA.
                 03 KEYWORD-DFH   PIC X(3).
                 03 FILLER        PIC X(69).
              02 KEYWORD-TABLE    REDEFINES KEYWORD-AREA.
                 03 KEYWORD-BYTE  PIC X(1) OCCURS 22 INDEXED BY
KX.
           END DATA
           MOVE CA-METACOBOL-INPUT-RECORD TO SEARCH-BLOCK
           MOVE SPACES TO KEYWORD-BLOCK
           SET SX TO 1
           SET KX TO 1
           LOOP VARYING SX FROM 1 BY 1 THRU 22
              IF SEARCH-BYTE (SX) NOT NUMERIC AND
                 SEARCH-BYTE (SX) NOT = SPACE
              OR SEARCH-BYTE (SX) = SPACE AND
                 KEYWORD-DFH  = 'DFH'
                 MOVE SEARCH-BYTE (SX) TO KEYWORD-BYTE (KX)
                 SET KX UP BY 1
              ENDIF
           UNTIL KEYWORD-DFH = 'DFH' AND KX > 4
           ENDLOOP
           IF KEYWORD-DFH = 'DFH' AND
              SX < 22
              MOVE 2 TO STATUS-OF-INPUT
              MOVE '      *$NOMCT' TO IXIT-CARD-IMAGE
              SET-TRUE RECORD-AVAILABLE
           ELSE
              MOVE CA-METACOBOL-INPUT-RECORD TO IXIT-CARD-IMAGE
              SET-FALSE RECORD-AVAILABLE
           ENDIF
           MOVE 'Y' TO IXIT-CB-PROGRAM-STATUS
           MOVE 'P' TO IXIT-CB-INTERFACE-STATUS
      /
```

```
        PROCESS-DFH.
            MOVE 'Y' TO IXIT-CB-PROGRAM-STATUS
            MOVE 'P' TO IXIT-CB-INTERFACE-STATUS
            MOVE CA-METACOBOL-INPUT-RECORD TO IXIT-CARD-IMAGE
            IF DFH-CONTINUATION = SPACE
               MOVE 3 TO STATUS-OF-INPUT
            ENDIF
            SET-FALSE RECORD-AVAILABLE
       /
        PROCESS-MCT.
            MOVE 'Y' TO IXIT-CB-PROGRAM-STATUS
            MOVE 'P' TO IXIT-CB-INTERFACE-STATUS
            MOVE '      *$MCT' TO IXIT-CARD-IMAGE
            MOVE 1 TO STATUS-OF-INPUT
            SET-TRUE RECORD-AVAILABLE
```

Mainframe COBOL Considerations

The CA-MetaCOBOL+ Translator is not written in COBOL. A COBOL input exit called by the translator will initiate a COBOL run unit marking the input exit as the main module. In order to prevent the translator from marking the input exit as the main module, an ASSEMBLER routine must be written that calls one of the COBOL library subroutines prior to passing control to the COBOL program.

The name of the COBOL library subroutine to be named in the SRADDR external address is ILBOSET0 for MVS and ILBDSET0 for VSE. This ASSEMBLER module and the COBOL program can be link-edited into one executable module. This module should be named CSECT and should be placed on the ENTRY link-edit control statement.

The following is suggested as a model routine.

```
xxx      CSECT              CSECT NAME OF USER CHOICE
         USING *,15
         TM    FLAG,FIRST
         BO    ENTER        BR NOT FIRST ENTRY
         OI    FLAG,FIRST   CLOSE GATE
         STM   14,12,12(13) SAVE ENTRY (CA-METACOBOL+) REGISTERS
         LA    14,SAVE      TEMPORARY REGISTER SAVE AREA
         ST    13,8(13)     CHAIN
         ST    14,4(14)       ... IT
         LR    13,14
         SR    1,1          NO PARAMETERS ON CALL
         L     15,SRADDR
         BALR  14,15        LINK TO COBOL UTILITY SUBPROGRAM
         L     13,4(13)
         LM    14,12,12(13) RESTORE CA-METACOBOL+ REGISTERS
ENTER    L     15,PGMADDR
         BR    15           BRANCH TO COBOL PROGRAM
PGMADDR  DC    V(zzz)       VCON OF COBOL PROGRAM ENTRY
```

```
SRADDR   DC    V (yyyyyyyy) VCON OF COBOL SR TO SET 'CALLED' SWITCH
SAVE     DS    18F FLAG
         DC    X'00'
FIRST    EQU   X'01'
         END   xxx          CSECT NAME OF USER CHOICE
```

PC COBOL Considerations

CA-Realia COBOL is supplied with CA-MetaCOBOL+/PC. You must use the compiler supplied with CA-Realia COBOL to compile your program.

Follow the steps below for link-editing.

1. When you link-edit the input exit program, you must direct the executable module to a .DLL library.
2. You must include the CA-supplied component METARCL as the first library component in the linkage-edit library specifications.
3. You must specify a definition file that contains these two statements:

```
LIBRARY
EXPORTS modulename
```

6.3.2 Assembler Interface Conventions

The CA-MetaCOBOL+ Translator uses standard IBM register and linkage conventions to call an input exit program. The input exit program must save general-purpose registers on entry and restore them on exit.

On entry to the input exit program, the following rules apply:

Register 1 contains the address of the list of fullword parameter addresses,

Register 13 contains the address of an 18-fullword register save area,

Register 14 contains the address of the return point in CA-MetaCOBOL+, and

Register 15 contains the address of the entry point of the input exit program.

The three interface parameters described in Section 6.2 are addressed in the following manner: the address contained in register 1 points to three fullword addresses, which are respectively the address of the 130-byte call workarea, the 80-byte card image returned to CA-MetaCOBOL+, and the four-byte input exit control block. Note that the high-order bit of the third fullword is set 'on' to designate the end of the parameter address list.

When the input exit program is called, if the interface status byte (the second byte of the input exit control block) contains a '*' or '&', the input exit program can return a record or message to CA-MetaCOBOL+ by placing the contents of the record or message in the card image and returning the proper control character in the interface status byte. The input exit program must also indicate whether CA-MetaCOBOL+ is to return control to the input exit program by placing a 'Y' or 'N' in the subprogram status byte; i.e., the first byte of the input exit control block.

6.4 ADRXIXIT

If you must call more than one input exit program in a single translation, specify IXIT=ADRXIXIT. ADRXIXIT is a CA-MetaCOBOL+ input exit program that manages up to ten input exit programs.

ADRXIXIT performs a load and branch to a "destination" input exit program on the first call from CA-MetaCOBOL+. It branches to the same input exit program on subsequent calls.

Note: ADRXIXIT is required when calling COBOL input exit programs on the PC.

The name of the destination input exit is specified in the first eight bytes of either &CALL or *$CALL. For example, to load the input exit program ADRPSB, specify:

```
&CALL &('ADRPSB ' destination-exit-parameters &)
```

If you require more than ten input exit programs for a single translation, specify '--CANCEL' immediately before the name of the input exit program you wish to delete. For example, to delete the program 'ADRPSB', specify:

```
&CALL &('--CANCELADRPSB ' &)
```

Appendix A. Glossary

The following terms and definitions are considered to be standard within CA-MetaCOBOL+.

Active Division

There are three definitions for this term.

First, the four 'Active Division' codes placed in Area A of a macro definition are:

I IDENTIFICATION
E ENVIRONMENT
D DATA
P PROCEDURE

Second, the Active Division is determined by the COBOL source text division being processed. A COBOL division header or an abbreviation must be present in the source text to establish it as active. The abbreviations are $ID, $ED, $DD, $PD.

Third, when an 'Out of Line' macro directive is used, the Active Division may change if the directive specifies a division other than the current Active Division. This adjustment remains in effect until one of the events listed below occurs. If one of the following events occurs, the original COBOL source text Active Division is restored:

- another directive specifying a different division is executed
- the macro terminates
- the &END macro directive is executed

If the &NOEND directive is used, an Active Division established by a macro directive remains in effect after the macro is terminated. The Active Division established in this manner remains in effect until another 'Out of Line' macro directive or the &END directive is executed.

Alphabetic character

Alphabetic characters are the sets 'A' through 'Z' and 'a' through 'z'.

Area A

COBOL defines columns 8-11 as Area A. Only the first word beginning in Area A is considered an Area A word. Another word on the same line as the Area A word is considered an Area B word.

Area A indicator

During translation, an 'Area A indicator' is placed in front of a source text word or macro model word constant found in Area A. This indicator causes the Area A word to be placed in Area A of the output.

Area B

COBOL defines columns 12-72 as Area B. Any word within these columns is considered an Area B word. Any word after the first word in Area A is also considered an Area B word.

Character

A letter, digit, or symbol is a character. Major classifications of characters are:

Alphabetic: A through Z and a through z.

Numeric: the digits 0 through 9.

Punctuation: comma, semicolon, period, and colon.

Sign: plus (+) and minus (-) signs.

Special characters: these are neither alphabetic nor numeric, i.e., *, %, etc.

Comment lines

Any line with an asterisk or slash in the Indicator Area is a comment. These lines may contain special codes or commands to control CA-MetaCOBOL+, such as Translator directing statements.

Constructs (control structures)

are combinations of macro directives that define the logic and scope of selection and repetition.

Continuation column

(see Indicator Area)

Current value

is the actual value of a macro variable or symbolic operand. This value may have been taken from the input or assigned through processing.

Data item definition

is any sentence in the DATA DIVISION beginning with a level number and ending with a period. A definition may include the name of the item and all attribute clauses needed to describe it.

Data-name

is a word that defines a data item in the DATA DIVISION. The word must have at least one alphabetic character.

Directive

is a special word, coded in a macro model, that starts a Translator action. This word does not appear in the output. A directive functions like a COBOL verb that defines an action to be taken by a COBOL program. A directive always begins with an ampersand (&).

Directive expression

is a series of words, beginning with a directive, that defines explicit Translator action. A directive expression functions like a COBOL statement.

Division header

COBOL division headers begin in Area A. The following words, if they begin in Area A, are recognized as Division Headers:

DATA DIVISION	IDENTIFICATION DIVISION
ENVIRONMENT DIVISION	PROCEDURE DIVISION
ID DIVISION	PROCEDURE DIVISION USING...
$DD	$ID
$ED	$PD

Event

An event is a task that occurs during translation and is designated by a macro name. When an event occurs, a predefined "Event Macro Name" causes a macro-guided task to be performed. An event is inherent to the source text. There is no matching text. The seven Event Macro Names and the events they cause are described below.

$-LEVEL

is in the DATA DIVISION before a level number preceded by a sentence or header ending with a period. Level numbers 01 and 77 coded in Area A are also preceded by an Area A indicator.

$DDE occurs at the end of the DATA DIVISION. DATA and PROCEDURE DIVISION headers must be present in the source text.

$DDX has the same conditions as the $DDE event but only occurs after any $DDE macro has ended.

$PDE occurs at the end of the source text or the end of a nested program. A division header for the next program must exist.

$PDX has the same conditions as the $PDE event but only occurs after any $PDE macro has ended.

$-PROC

occurs in the PROCEDURE DIVISION and is prompted by any word coded in Area A.

$-VERB

occurs in the PROCEDURE DIVISION and is initiated by any verb or period ending a header or sentence. If a verb is coded in Area A, the $-PROC macro takes precedence over $-VERB.

Execute time
The run time of a COBOL program after translation, compilation, and link-editing.

Figurative constants
consist of the user-defined names for symbolic characters, as defined within COBOL, and the reserved names for manifest constants. The reserved names are:

HIGH-VALUE	SPACE
HIGH-VALUES	SPACES
LOW-VALUE	ZERO
LOW-VALUES	ZEROES
QUOTE	ZEROS
QUOTES	

Floating point literal
is a literal with a mantissa and an exponent, both of which may have a leading sign. The mantissa may be 1-16 digits in length and must include a decimal point. An E followed by an optional sign and one or two digits represents the exponent.

Generated program text
is the program text produced by a translation. It is usually a complete COBOL program.

Hexadecimal literal
in a macro model, is a string of 1-128 pairs of hexadecimal digits bounded by pairs of single or double quotation marks, e.g., ""C1"" or ''C1''. Each digit pair represents one EBCDIC character.

The digits may be 0-9 or A-F inclusive. The first quote must be preceded by a space or margin, and the last quote must be followed by a space, margin, or terminating punctuation character. Macro assembly converts the literal to a nonnumeric literal. A hexadecimal literal is valid anywhere a nonnumeric literal is valid. Continuation rules are the same for hexadecimal and nonnumeric literals.

Within a COBOL program, a hexadecimal literal may be defined as a string of 1-160 pairs of hexadecimal digits. The digits may be 0-9, A-F, or a-f inclusive. Single or double quotation marks preceded by an X bind the literal, e.g., X"C1" or X'C1'. This literal is valid anywhere a nonnumeric literal is valid. Continuation rules are the same for hexadecimal and nonnumeric literals. The initial delimiter (X" or X') may not be split across lines.

Identification Area
COBOL defines columns 73-80 as the 'Identification Area'. It is used for brief comments or consistent identification throughout the source program.

Identifier
is a unique data-name. The data-name can be made unique by using qualifiers, subscripts, or both.

Indicator Area

is column 7. It specifies:

- the continuation of words or nonnumeric literals from the previous line to the current line.
- treatment of the text as documentation (comments). An asterisk indicates a comment.
- page breaks. A forward slash (/) indicates a page break.
- debugging lines.

Keyword

is a constant that can be used in a string macro prototype after the macro-name. The following compiler directing words can not be specified as keywords in string macro prototypes:

*CBL	REMARKS
CBL	SKIP1
*CONTROL	SKIP2
CONTROL	SKIP3
EJECT	TITLE
NOTE	

A keyword can be defined as a literal.

Label

is a logical destination defined within a macro model. It takes the form of "&Lname" and may be coded in Area A or Area B of the model. A label must be unique within the macro region in which it is defined. It can be referenced by macro directives in other models.

Level indicator

The words FD, SD, RD, and CD are level indicators. They identify a specific type of file or a position in the data hierarchy.

Level number

is an integer in the DATA DIVISION immediately following either a period or a period and an Area A indicator.

Literal

(See Figurative constant, Hexadecimal literal, Nonnumeric literal, and Numeric literal)

Logical destination

are special words in a macro model that precede directive expressions and replacement words and declare a place. If specified within a directive expression, the logical destination is referenced, and it names the location to which a transfer of control may be made.

(See also Label and Tag)

Macro
is the equation of a macro prototype to a macro model. A macro may generate output statements through direct substitution or through conditional, logical analysis.

Macro call
is the execution of a macro model. The execution can be initiated by:

- a match of the macro prototype from the input stream
- a word substituted from another macro model
- an event

A completed macro call removes any matching word or words from the text.

Macro definition
A complete macro definition is formed by the explicit definition of a macro type, active division, macro prototype, and macro model.

Macro loading
is the reading and assembling of macros in preparation for translation.

Macro model
is the specific translation action executed upon recognition of a source word, syntax pattern, word prefix, or event that matches the corresponding macro prototype. When the macro model is omitted, any text matching the prototype is removed.

Macro name
is the first or only word of a macro prototype. The following rules must be observed when creating macro names:

- A macro name can not exceed 30 characters.
- A prefix macro name can not exceed 29 characters.
- The arithmetic operators asterisk (*) and forward slash (/) may only be specified as 1-character prefix macro names.
- A literal can not be a macro name.
- The words NOTE and REMARKS are only available to macros when the COMMENT option is not specified.
- The word COPY is available only when the COPY option is PASSIVE or IGNORE.
- The following compiler directing words are <u>never</u> available to macros:

```
*CBL        EJECT
CBL         SKIP1
*CONTROL    SKIP2
CONTROL     SKIP3
EJECT
```

Macro prototype
is completed by the explicit definition of a single word, a partial word, a series of words, or an event. The macro's active division codes, the Macro prototype, words and events in the input stream, and words output by other macros determine when the macro model will execute.

Macro set
A group of macros designed to process a COBOL program. Macro sets can be independent or interdependent.

Macro type

The five macro types and their codes are:

P Prefix macro
S String macro
U Unverb macro
V Verb macro
W Word macro

An unverb macro allows you to exclude one or more verbs from the list of words that your compiler will recognize as verbs.

Code the macro type in the Indicator Area (column 7).

Margin A
(See Area A)

Margin B
(See Area B)

Model
(See Macro model)

Nest level
The level at which a macro executes in a series of "nested macro calls" is its "nest level." A nested macro call is a macro definition invoked by a macro call embedded in another macro. Nine nested macro calls are supported. A tenth attempt to invoke a macro causes an error. Only one prefix macro can be present in a series of nested macro calls.

Nonnumeric literal
is a constant of 1-160 characters and spaces bound by apostrophes or quotation marks. Nonnumeric literals bound by quotation marks (") or single apostrophes (') are equally acceptable in the same source text. The QUOTE and APOST options define how literals will be bound in the generated text.

Numeric character
is a character from 0-9.

Numeric literal
is a constant 1-18 digits in length. A numeric literal may contain a sign character as the left-most character and a decimal point.

Procedure header
PROCEDURE DIVISION, DECLARATIVES, and END DECLARATIVES are are procedure headers.

Procedure name
is a paragraph name or section name in the PROCEDURE DIVISION. When referenced, a procedure name must be unique. It can be made unique by adding a section name qualifier.

Prototype
(see Macro prototype)

Punctuation character
COBOL specifies the following punctuation characters: colon, comma, equals sign, parenthesis, period, semicolon, and space.

Qualifier
is a unique name hierarchically superior to a nonunique data-name or paragraph name. To reference a nonunique name, follow the name with either IN or OF and the qualifying name. Paragraph names can have only one qualifier, the section name in which the paragraph is defined. Data-names may have more than one qualifer.

Region
An intial macro region is implied. The arbitrary subdivision of multiple macro sets by this *$REGION Translator directing statement creates an additional macro region. Ten regions are supported. Regions prevent identical variable names and label names in different macro sets from conflicting when the combined macro sets execute in different regions of the same translation.

Reserved variables
There are 11 predefined, external, nonnumeric variables. These reserved variables are: &VUPSI1 through &VUPSI8, &VSOURCE, &VTARGET, and &VDIALECT. These variables cannot be specified within &EXTERN, &GLOBAL, or &LOCAL directives or as the receiving item in an &SET or &SETR directive.

Sequence Number Area
is located in columns 1-6 in a COBOL program. Use this area to label a source statement line. Any character may be coded in this area.

Sign character
is a plus (+) or minus (-) sign.

Space
is one or more contiguous blanks (hexadecimal 40).

Source program text
is the program text to be processed by translation. It is usually a complete COBOL program.

Special character
is any character other than an alphabetic or numeric character or a space.

Subscript

Acceptable macro variable subscripts are: numeric literals, character variables with a numeric value, numeric variables, and a reserved variables with a numeric value.

Subscripted variables can be used anywhere a variable is allowed except in the &PIC and &SETTAB directives and line output.

Substitution word

in a macro model, is a symbolic or constant word substituted directly or indirectly into the generated output.

Symbolic operand

is the macro notation & or &n, where "n" is 0-15. This notation represents the current value determined by words from the input stream, source item acquisition (&GET), or initialization (&EQU).

Tag

is a logical destination defined within a macro model. It takes the form of "&Tname" and may be coded in Area A or Area B of the model. A tag must be unique within the macro model in which it is defined. It can not be referenced by macro directives in other macro models.

TDS relative address

is the address of an entry in the TDS Table (Attribute Table). Each entry in the TDS table has a serial number. This number is the address or position of the entry in the table. Each data-name in the generated program can have an entry in the table. One method of accessing the data in the table is using the TDS relative address.

Terminating punctuation character

is a punctuation character followed by a space, margin, or right parenthesis.

Translation

is the modification, expansion, and analysis of standard COBOL or other input to produce COBOL output as determined by macros, options, and translator directing statements.

Translate-time

is the time during which translation is taking place.

Variable

is a named storage area available to macros.

Word

is a string of characters fitting one of the following descriptions:

- Integer
- Numeric literal.
- Nonnumeric literal.
- Terminating punctuation character.
- parenthesis
- picture character string
- colon
- Contiguous character string less than 30 characters bound by a space, margin, or terminating punctuation character. The string may not contain an embedded space, quotation mark, or punctuation character. Such words can be a verb, reserved word, data-name, or procedure name.

Appendix B. Procedure Documentor

The CA-MetaCOBOL+ Procedure Documentor (MPD) is a COBOL program that aids in macro set debugging and maintenance. The input to the MPD may be any macro set or series of macros (procedure set). The output from the MPD is a series of reports. The first two reports are always produced, and are entitled RUN-TIME OPTIONS and MACRO SET LISTING.

Additional reports can be activated or suppressed by means of execute-time parameters. The parameters can be specified in any sequence, and activate the optional reports shown in the following list:

Parameter	**Optional Report Name**
VXREF	EXTERNAL/GLOBAL VARIABLE CROSS-REFERENCE
LXREF	GLOBAL LABEL CROSS-REFERENCE
LOCAL=	LOCAL VARIABLE/TAG CROSS-REFERENCE
DLIST	NOTES & FLAGS
INDEX	MACRO INDEX
SWD	SPECIAL WORD DIRECTORY

An additional parameter, ID=, provides macro set identity to be displayed in all report headings.
The remainder of this appendix describes the various reports and parameters, provides MPD installation and operating instructions, and explains execute-time diagnostics, causes, and corrective action.
All report examples are shown in the sequence in which displayed by the MPD.

B.1 MPD Execute-time Parameters and Reports

The CA-MetaCOBOL+ Procedure Documentor reads one or more macro sets as input and produces two standard reports. Additional cross- referencing and indexing reports are optional, and are controlled by execute-time parameters.

The various MPD execute-time parameters and reports are described in subsequent paragraphs. The example macro set documented is relatively short for ease of explanation. The value of the MPD is most obvious, of course, when documenting a procedure set that is larger.

B.1.1 ID= Parameter

The ID= parameter provides the MPD with macro set identification that is to be printed at the top of each report page. The parameter is:

Format:

```
ID=procedure-name
```

ID=
is the keyword. If the ID= parameter is not specified, no macro set name is included in the reports.

procedure-name
is the macro set name, and can be specified as from 1 to 8 alphabetic and numeric characters.

B.1.2 Standard Reports

The two standard reports that are always produced are the RUN- TIME OPTIONS and MACRO SET LISTING reports.

The RUN-TIME OPTIONS report lists all options in effect during an MPD execution. A sample report follows:

```
CA-METACOBOL+ PROCEDURE DOCUMENTOR V9.R2 - RUN-TIME OPTIONS
INDEX   DLIST   LXREF   VXREF   SWD  ID=MPDDEMO  LOCAL=ALL
**SPECIAL WORD DIRECTORY INVOKED FOR:
&A
&EXIT
&DO
```

The various options listed are explained in subsequent paragraphs. The SPECIAL WORD DIRECTORY lists user-defined words for which the MPD is to search and cross-reference.

The MACRO SET LISTING is the listing of all input. Each input line is assigned a unique, sequential line number for cross- referencing and indexing purposes. The page heading lines include the identity of the macro sets, the date and time of run, and a page number. A sample report follows:

```
CA-METACOBOL+ PROCEDURE DOCUMENTOR - MACRO SET LISTING    MPDDEMO
09/18/91   100847        PAGE 1

      LINE     CARD IMAGE

     000001          SI   $$EXTERN :
     000002                   &EXTERN &B@FILE-SECTION-FOUND    = FALSE
     000003                   &EXTERN &B@WORKING-STORAGE-FOUND = FALSE
     000004                   &GLOBAL &V@FILENAME(10) X(30)
     000005               &L@FS
     000006                   &IF&IF &B@FILE-SECTION-FOUND
     000007                      &NOTE &( 'FILE SECTION PREVIOUSLY GENERATED' &)
     000008                   &ELSE
     000009                      &SET &B@FILE-SECTION-FOUNG = TRUE
     000010                      &A FILE SECTION.
     000011                   &ENDIF
     000012                   &EXIT
     000013               &L@WS
     000014                   &DO &L@FS
     000015                   &IF &B@WORKING-STORAGE-FOUND
     000016                      &NOTE &( 'WORKING-STORAGE PREVIOUSLY GENERATED' &)
     000017                   &ELSE
     000018                      &SET &B@WORKING-STORAGE-FOUND = TRUE
     000019                      &A WORKING-STORAGE SECTION.
     000020                   &ENDIF
     000021                   &EXIT
     000022               &L@LS
     000023                   &LOCAL &BLINKAGE-SECTION-FOUND = FALSE
     000024                   &DO &L@WS
     000025                   &IF NOT &BLINKAGE-SECTION-FOUND
     000026                      &SET &BLINKAGE-SECTION-FOUND = TRUE
     000027                      &A LINKAGE SECTION.
     000028                   &ENDIF
     000029                   &EXIT
     000030          SD   -FS:                        &DO &L@FS
     000031          SD   FILE SECTION. :             &DO &L@FS
     000032          SD   -WS :                       &DO &L@WS
     000033          SD   WORKING-STORAGE SECTION. :  &DO &L@WS
     000034          SD   -LS :                       &DO &L@LS
     000035          SD   LINKAGE SECTION. :          &DO &L@LS
     000036          SD   $DDX :                      &DO &L@WS
     000037          SE   SELECT &1 : SELECT &1
     000038                   &LOCAL  &VINDEX 99
     000039                   &SET &VINDEX = &VINDEX + 1
     000040                   &IF &VINDEX > 10
     000041                      &DO &TOUTPUT-NOTE
     000042                   &ELSE
     000043                      &SET &V@FILENAME(&VINDEX) = &1
     000044                   &ENDIF
     000045                   &GOBACK
     000046               &TOUTPUT-NOTE
     000047                   &NOTE &) 'FILE NAME ' &1 ' NOT INCLUDED' &)
     000048                   &EXIT
```

B.1.3 VXREF PARAMETER and EXTERNAL/GLOBAL VARIABLE CROSS-REFERENCE Report

The VXREF parameter activates the EXTERNAL/GLOBAL VARIABLE CROSS-REFERENCE report. The parameters are:

Format:

```
{VXREF   }
{NOVXREF}
```

VXREF activates the report. This is the default.

NOVXREF suppresses the report.

Example:

```
CA-METACOBOL+ PROCEDURE DOCUMENTOR - PROCEDURE ANALYSIS  MPDDEMO        09/18/91
100847    PAGE 1
EXTERNAL/GLOBAL VARIABLE CROSS-REFERENCE

  VARIABLE                                          LINE   *-----REFERENCES------*
  NAME                          PIC  OCCUR          DEF.   *---------------------*

E &B@FILE-SECTION-FOUND           X   1          F      000002  000006 000009
E &B@WORKING-STORAGE FOUND        X   1          F      000003  000015 000018
G &V@FILENAME                     X  30   10            000004  000043
E &VDIALECT                       X   2                 000000
E &VSOURCE                        X   1                 000000
E &VTARGET                        X   1                 000000
E &VUPSI1                         X   1                 000000
E &VUPSI2                         X   1                 000000
E &VUPSI3                         X   1                 000000
E &VUPSI4                         X   1                 000000
E &VUPSI5                         X   1                 000000
E &VUPSI6                         X   1                 000000
E &VUPSI7                         X   1                 000000
E &VUPSI8                         X   1                 000000
```

The EXTERNAL/GLOBAL VARIABLE CROSS-REFERENCE report includes, from left to right:

- the character "E" (for External) or "G" (for Global);
- the variable name;
- the variable picture;
- the number of occurrences, if specified;
- the characters "V" (for variable with initial value specified), "T" (for Boolean variable set true), or "F" (for Boolean variable set false);
- the line number on which the variable is defined;
- the line numbers of all references to the variable.

All reserved external variable names (&VDIALECT, &VSOURCE, etc.) are cross-referenced on the EXTERNAL/GLOBAL VARIABLE CROSS- REFERENCE report.

References to all variable names that are undefined are listed on the EXTERNAL/GLOBAL VARIABLE CROSS-REFERENCE report followed by the word "*****UNRESOLVED*****".

If the input macro set includes one or more *$REGION Translator-directing statements, a separate EXTERNAL/GLOBAL VARIABLE CROSS- REFERENCE report is produced for each region. The region number is included in the heading.

B.1.4 LXREF Parameter and GLOBAL LABEL CROSS-REFERENCE Report

The LXREF parameter activates the GLOBAL LABEL CROSS-REFERENCE report. The parameters are:

Format:

```
{LXREF  }
{NOLXREF}
```

LXREF
activates the report. This is the default.

NOLXREF
suppresses the report.

Example:

```
CA-METACOBOL+ PROCEDURE DOCUMENTOR - PROCEDURE ANALYSIS    MPDDEMO       09/18/91
100847   PAGE 1
GLOBAL LABEL CROSS-REFERENCE

LABEL                            LINE    *----------------REFERENCES---------------*
NAME                             DEF.    *----------------------------------------*

&L@FS                                     000005  000014 000030 000031
&L@LS                                     000022  000034 000035
&L@WS                                     000013  000034 000032 000033 000036
```

The GLOBAL LABEL CROSS-REFERENCE report includes, from left to right:

the label name;
the line number on which the label is defined;
the line numbers of all references to the label.

References to all label names that are undefined are listed on the GLOBAL LABEL CROSS-REFERENCE report followed by the word "*****UNRESOLVED*****".

If the input macro set includes one or more *$REGION Translator- directing statements, a separate GLOBAL LABEL CROSS-REFERENCE report is produced for each region. The region number is included in the heading.

B.1.5 LOCAL Parameter, LOCAL VARIABLE CROSS-REFERENCE Reports, and LOCAL TAG CROSS-REFERENCE Reports

The LOCAL= parameter activates the LOCAL VARIABLE CROSS-REFERENCE and LOCAL TAG CROSS-REFERENCE reports. The parameter is:

Format:

```
LOCAL = {ALL        }
        {macro-name}
```

ALL
activates the reports for all macros.

macro-name
activates the reports for the macro named. This is the default.

If the LOCAL= parameter is specified, the NOVXREF and NOLXREF parameters must not be specified.

Examples:

```
CA-METACOBOL+ PROCEDURE DOCUMENTOR - PROCEDURE ANALYSIS    MPDDEMO        09/18/91   100847   PAGE 1
LOCAL VARIABLE CROSS-REFERENCE - 000001 $$EXTERN

VARIABLE                                        LINE   *----------------REFERENCES----------------*
NAME                            PIC  OCCUR     DEF.   *-----------------------------------------*

&BLINKAGE-SECTION-FOUND          X   1          F      000023  000025 000026


CA-METACOBOL+ PROCEDURE DOCUMENTOR - PROCEDURE ANALYSIS    MPDDEMO       09/18/91   100847    PAGE 1
LOCAL VARIABLE CROSS-REFERENCE - 000037 SELECT

VARIABLE                                          LINE   *---------------REFERENCES---------------*
NAME                            PIC  OCCUR       DEF.   *-----------------------------------------*

&VINDEX                          9   2                 000038  000039 000039 000040 000043
```

The LOCAL VARIABLE CROSS-REFERENCE report heading includes the macro-name and line on which the macro is defined. In the body, the report includes from left to right:

the variable name;
the variable picture;
the variable number of occurrences, if specified;
the character "V" (for variable with initial value specified), "T" (for Boolean variable set true), and "F" (for Boolean variable set false);
the line number on which the variable is defined;
the line numbers of all references to the variable.

If the input macro set includes one or more *$REGION Translator-directing statements, the region number is included in the heading.

Example:

```
CA-METACOBOL+ PROCEDURE DOCUMENTOR - PROCEDURE ANALYSIS    MPDDEMO        09/18/91
100847   PAGE 1
LOCAL TAG CROSS-REFERENCE -      000037 SELECT

VARIABLE                    LINE     *------------------REFERENCES-----------------*
NAME                        DEF.     *--------------------------------------------*

&TOUTPUT-NOTE                        000046  000041
```

The LOCAL TAG CROSS-REFERENCE report heading includes the macro- name, and line number on which the macro is defined. In the body, the report includes, from left to right:

- the variable name;
- the line number on which the tag is defined;
- the line number of all references to the tag.

References to all tag names that are undefined are listed on the LOCAL TAG CROSS-REFERENCE report, followed by the word "*****UNRESOLVED*****".

If the input macro set includes one or more *$REGION Translator-directing statements, the region number is included in the heading.

B.1.6 DLIST Parameter and NOTES & FLAGS Report

The DLIST parameter activates the NOTES & FLAGS report. The parameters are:

Format:

```
{DLIST  }
{NODLIST}
```

DLIST
activates the report. This is the default.

NODLIST
suppresses the report.

Example:

```
CA-METACOBOL+ PROCEDURE DOCUMENTOR - PROCEDURE ANALYSIS    MPDDEMO       09/18/91
100847   PAGE 1
NOTES & FLAGS

          NOTE TEXT                                                    LINE

&( 'FILE SECTION PREVIOUSLY GENERATED' &)                             000007
&( 'WORKING-STORAGE PREVIOUSLY GENERATED' &)                          000016
&( 'FILE NAME ' &1 ' NOT INCLUDED' &)                                 000047
```

The NOTES & FLAGS report includes, from left to right:

the diagnostic message -- the contents of the diagnostic word (&NOTE) or the macro-name (&FLAG);

the line number on which the &NOTE or &FLAG is defined.

B.1.7 INDEX Parameter and MACRO INDEX Report

The INDEX parameter activates the MACRO INDEX report. The parameters are:

Format:

```
{INDEX  }
{NOINDEX}
```

INDEX
activates the report. This is the default.

NOINDEX
suppresses the report.

Example:

```
CA-METACOBOL+ PROCEDURE DOCUMENTOR - PROCEDURE ANALYSIS    MPDDEMO         09/18/91
100847  PAGE 1
MACRO INDEX

TYPE DIV           NAME                              LINE

 S   I     $$EXTERN                                 000001
 S   D     $DDX                                     000036
 S   D     -FS                                      000030
 S   D     -LS                                      000034
 S   D     -WS                                      000032
 S   D     FILE                                     000031
 S   D     LINKAGE                                  000035
 S   E     SELECT                                   000037
 S   D     WORKING-STORAGE                          000033
```

The MACRO INDEX report includes, from left to right:

- the macro type code;
- the division codes;
- the macro-name (in alphanumeric sequence);
- the region number in which defined, if regionalized;
- the line number on which the macro is defined.

B.1.8 SWD Parameter and SPECIAL WORD DIRECTORY Report

The SWD parameter activates interpretation of a special word list and the SPECIAL WORD DIRECTORY report. The parameter is:

Format:

```
SWD                      (for MVS and PC)

SWD=(word1,word2,...)  (for VSE)
```

SWD activates the special word function. If not specified, no special word processing is done. A maximum of 25 special words can be cross-referenced during a single MPD execution. No word may exceed 30 characters in length.

Example:

```
CA-METACOBOL+ PROCEDURE DOCUMENTOR - PROCEDURE ANALYSIS    MPDDEMO        09/18/91
100847   PAGE 1
SPECIAL WORD DIRECTORY

WORD

&A                           000010  000019  000027
&DO                          000014  000024  000030  000031  000032  000033  000034
000035  000036  000041
&EXIT                        000012  000021  000029  000048
```

The SPECIAL WORD DIRECTORY report includes, from left to right:

the user-defined special words (in alphabetic sequence);
the line numbers of all references to each word.

If no reference to a special word is found, the message "***NO OCCURRENCES FOUND" is displayed in the reference column.

The mode of parameter and special word specification differs between VSE and MVS operating environments, and is outlined in Sections B.2 and B.3 that describe MPD, VSE, and MVS operating instructions.

B.2 MVS MPD Operating Procedures

The MVS CA-MetaCOBOL+ Procedure Documentor operating procedures are explained below. The MPD program is available on the CA-MetaCOBOL+ installation tape in the target load library.

B.2.1 JCL for MVS MPD Execution

The JCL required to execute the MPD under MVS is shown below.

```
  //stepname EXEC PGM=MPDname[,PARM='parameters']
  //STEPLIB  DD   DSN=user.loadlib,DISP=SHR
  //SYSPRINT DD   SYSOUT=A,DCB=BLKSIZE=121
  //SYSOUT   DD   SYSOUT=A
  //SORTWK01 DD   UNIT=SYSDA,SPACE=((TRK,40),,CONTIG)
  //SORTWK02 DD   UNIT=(SYSDA,,,SEP=SORTWK01),
  //              SPACE=((TRK,40),,CONTIG)
  //SORTWK03 DD   UNIT=(SYSDA,,,SEP=(SORTWK01,SORTWK02)),
  //              SPACE=((TRK,40),,CONTIG)
  //SORTLIB  DD   DSN=SYS1.SORTLIB,DISP=SHR
[ //SYSWORD  DD   *[,DCB=BLKSIZE=80]     ]
[   parameters                       ]    ]
  //SYSIN    DD   *[,DCB=BLKSIZE=80]

  (--Macro Set Input--)
  /*
```

SYSIN input can be read directly from card input through the normal system READER, from a sequential tape or from a sequential or partitioned source library. The BLKSIZE must be specified in multiples of that shown for the SYSPRINT data set, and for the SYSIN and SYSWORD data sets if input from an unlabelled file. The MPD accepts parameters via the PARM= field of the EXEC statement.

The SWD parameter requires specification of the SYSWORD data set for input of the special word list. The SYSWORD data set consists of the special words desired punched in columns 1-72 and separated by commas. Any word followed by a space terminates special word loading. For example, the parameters used in the sample reports could be specified and activated by a default as follows:

```
//stepname EXEC PGM=MPDname,PARM='ID=MPDDEMO,LOCAL=ALL,SWD'
.
.
.
//SYSWORD DD *
&A,&EXIT,
&DO
//SYSIN. . .
```

B.3 VSE MPD Operating Procedures

The VSE CA-MetaCOBOL+ Procedure Documentor operating procedures are explained below.

The MPD program is available on the CA-MetaCOBOL+ installation tape in the target load library. Refer to the CA-MetaCOBOL+ VSE *Installation Guide*.

B.3.1 JCL for VSE MPD Execution

The Job Control Statements required to execute the MPD under VSE are shown below.

```
// JOB DOCUMENT MACROS
// ASSGN SYS010,X'cuu'          Macro Input
// ASSGN SYS011,X'cuu'          Printer
// ASSGN SYS001,X'cuu'       }
// ASSGN SYS002,X'cuu'       }  Sort Work Files (Tape or Disk)
// ASSGN SYS003,X'cuu'       }
// DLBL SORTWK1,,yy/ddd,SD
// EXTENT SYS001,xxxxxxx,1,0,20,100
// DLBL SORTWK2,,yy/ddd,SD
// EXTENT SYS002,xxxxxxx,1,0,120,100
// DLBL SORTWK3,,yy/ddd,SD
// EXTENT SYS003,xxxxxxx,1,0,1000,
// EXEC MPDname
[OPTION parameters]
(--Macro Set Input--)
/*
/&
```

SYS010 input must be directly from card input.

The VSE version of the MPD accepts parameters via OPTION cards read as the first cards of the input stream. The word OPTION must appear in columns 2-7 followed by a single space. The optional parameters must begin in column 9, and must be separated by commas with no intervening spaces. If the OPTION is not found as the first card, the default values for MPD parameters are assumed.

For example, the parameters used in the sample reports could be specified and activated by default as any of the following:

```
OPTION SWD=(&A,&EXIT,&DO),ID=MPDDEMO,LOCAL=ALL

OPTION ID=MPDDEMO,SWD=(&A,&EXIT,&DO),LOCAL=ALL

OPTION INDEX
OPTION DLIST
OPTION VXREF,LXREF
OPTION ID=MPDDEMO
OPTION LOCAL=ALL
OPTION SWD=(&A,&EXIT,&DO)
```

B.4 PC Operating Procedures

The CA-MetaCOBOL+/PC Procedure Documentor operating procedures for PC-DOS and MS-DOS are explained below.

The MPD program is available on the CA-MetaCOBOL+/PC installation diskettes.

The SET Statements for executing the MPD on the PC are shown below.

```
SET SYSIN = filename [U]
SET SYSPRINT = filename [U]
SET SYSSORT = filename [U]

[ SET SYSWORD = filename [U] ]
```

The SET SYSWORD Statement is optional; however, the MPD parameter requires specification of the SYSWORD data set for input of the special word list. The SYSWORD data set consists of the special words desired punched in columns 1-72 and separated by commas. Any word followed by a space terminates special word loading.

The SYSWORD file, if used, must contain the words that were requested by the user to be included in the SPECIAL WORD DIRECTORY Report. In the example below, the words &A, &EXIT, and &DO are to be included in the report.

```
SYSWORD filename
&A,&EXIT,
&DO
```

See Section B.1.8 for more information about the SPECIAL WORD DIRECTORY Report.

Parameters
The parameters used in the sample reports could be specified and activated by a default as follows:

```
MPD SWD,LOCAL=ALL

.
.
.
```

B.5 MPD Execute-time Diagnostics

Unlike CA-MetaCOBOL+, the MPD does a minimum of macro syntax checking. The following is a list of possible diagnostics from MPD. Diagnostics issued during processing of an input line will cause the parsing of that line to terminate. Processing continues with the next line.

MPD00E-****INVALID RECORD TYPE****

A record has been returned from the sort with an invalid type code. The text of the record is printed under this message and MPD processing is terminated.

MPD01E-CONTINUED LITERAL NOT FOUND

A new macro or Line output directive was encountered while looking for continuation of a literal. Review code preceding the appearance of this error message for the exact location of problem.

MPD02W-COLON NOT FOUND

A colon could not be found in the first record of a macro prototype. Normal processing continues from the error message line.

MPD03E-INCOMPLETE CONCATENATION

A new macro or Line output statement was encountered while looking for the "&)" end of a concatenation. Review the code preceding the appearance of this message for the exact line causing the error.

MPD04E-WORD FOLLOWING EXTERN/GLOBAL/LOCAL MUST BEGIN WITH &V

The word following an &EXTERN, &GLOBAL or &LOCAL did not begin with the characters &V. Processing continues.

MPD05E-MISSING RIGHT PAREN

While processing a variable definition with an index, the right parenthesis was not found. Processing continues.

MPD06E-CONTINUED CONCATENATION OR LITERAL NOT FOUND

A macro definition was found before the termination of a concatenation or literal. Review the lines preceding the error message for the exact location of the syntax error. Normal processing continues from the error message line.

MPD07E-LOCAL VARIABLE TABLE FULL

More than 100 local variable definitions have been encountered in the macro being processed. Processing continues, but subsequent local variables are not processed. (To increase the table size, review the Working-Storage entries for LOC-VARIABLE-AREA and LOC-VARIABLE-TABLE.)

MPD08E-MORE THAN 9 *$REGIONS FOUND

More than 9 *$REGION Translator-directing statements were found in the procedure set. Processing continues, with all code following the first occurrence of this message attributed to REGION 9.

MPD11W-MORE THAN 25 WORDS SPECIFIED

In processing the SWD list, a 26th word is encountered. Processing continues with first 25 words specified only.

MPD12W-WORD EXCEEDS 30 BYTES

When processing the SWD list in the SYSWORD or OPTION card, a word of more than 30 bytes is found. Processing of words is terminated and the MPD continues with any words loaded before this message.

Appendix C. Macro Debugging

The testing and debugging of CA-MetaCOBOL+ macros and macro sets is very similar to testing and debugging of problem programs written in any language. Macro set debugging parallels integrated systems testing, where individual macros or programs must interrelate within the entire macro region or system.

This appendix discusses many of the techniques and facilities for macro debugging, including interpretation of test results, development of special-purpose debugging aids, and use of the CA-MetaCOBOL+ macro debugging aid feature.

Normal processing disciplines which apply to macro writing, such as careful definitions of the problem to be solved, pseudo-coding and desk checking, are beyond the scope of this appendix. Also not covered is interpretation of CA-MetaCOBOL+ Translator diagnostics, since a thorough description of cause and corrective action for each diagnostic is detailed in the CA-MetaCOBOL+ *User Guide*.

Following each test of a CA-MetaCOBOL+ macro or macro set, the results must be analyzed for CA-MetaCOBOL+ diagnostics and logical errors. The STATISTICS listing indicates the presence or absence of Translator ERRORs and WARNINGs. All errors should be corrected, if possible, before further logical analysis of test results.

Following correction of Translator errors, CA-MetaCOBOL+ input and output must be examined to ensure correct translation. Many common errors can be corrected by a thorough analysis of the Listing File. Others may require additional testing under control of user-defined or CA-MetaCOBOL+ debugging aids.

C.1 Interpretation of Test Results

The following sections provide checklists for analysis of many common logical errors.

C.1.1 No Apparent Call to a Macro

Failure to match source to a macro prototype is generally caused by:

- Failure to load the macro.
- P, S, or W type code not in column 7.
- Improper division codes in columns 8-11.
- Improper macro-name, such as NOTE.
- Type code inconsistent with macro prototype.
- Improper spelling of prototype keywords or source words.
- Incorrect symbolic operand qualification in a String-type macro prototype.
- Specification of keywords OF, IN, or parentheses after a qualified symbolic operand in a String-type macro prototype.
- An identical macro prototype is defined elsewhere in the macro set.
- A Prefix-type macro alters a Word- or String-type macro-name or a String macro keyword.
- A Word-type macro invalidates a String-type macro-name or alters a keyword. For example, the following String-type macro is never called because a Word-type macro removes the word CONTAINS:

```
WD   CONTAINS :
SD   BLOCK CONTAINS &1 ...
```

- The appropriate COBOL division header does not precede the source words.
- A CA-MetaCOBOL+ reserved word has been improperly specified as a qualified symbolic operand. For example, the following MOVE macro cannot be called because KEY is a COBOL reserved word, and therefore, cannot be an identifier or literal:

```
SP   MOVE &1(S,L) TO &2(S) :
  .
  .
  .
     MOVE KEY TO REF-CODE ...
```

- Improper specification of the source words within a COBOL NOTE, comment, or literal.

C.1.2 Improper Area Alignment

Improper Area A word alignment is generally caused by:

- A source or model substitution word inadvertently beginning in column 8 through 11.
- An &A directive being inadvertently executed in a macro model.

Improper formatting of an Area A word within Area B is generally caused by:

- Source and model substitution words which do not force Area A alignment.
- Improper execution of an &B directive.
- Inadvertent &STORE or &STOW of an Area A indicator (i.e., &n'T = 'A').

C.1.3 Incomplete Substitution

Incomplete word substitution via a macro model is generally caused by:

- Failure to include constant or symbolic substitution words in the macro model.
- Improper transfer to control around model substitution words.
- Substitution to the wrong output location, indicating a logical error in out-of-line substitution.
- Out-of-line substitution to a non-existent location, resulting in generation of the LOST OUT-OF-LINE listing.
- Incomplete translation of the "logical" macro prototype (i.e., the String-type macro prototype and all words to be acquired via &GET/&STORE/&STOW logic). Normally, a portion of the "logical" prototype remains in the generated COBOL. This situation is usually caused by improper and premature determination of the logical terminating word.

C.1.4 Generated COBOL Out of Order

Whenever major portions of the generated COBOL program are out of order, the cause generally is:

- Out-of-line substitution remains active following macro model execution due to an executed &NOEND directive. No subsequent macro call has reset the &NOEND.

C.1.5 Improper Terminating Period Substitution

Substitution of an unnecessary and unwanted period in the COBOL output is generally caused by:

- An unnecessary period following a source statement.
- An inadvertent period following a constant or symbolic substitution word.
- An inadvertent period following a directive or directive expression. For example, the following &SET directive expression does not include the period, which is translated as a substitution word:

```
&SET &VX = &VX + 1.
```

C.1.6 Consecutive Period Substitution

Consecutive terminating periods in the generated COBOL output are generally caused by:

- Improper substitution of a concatentation construction or variable containing a terminating period, resulting in loss or identity of the period as a separate word.

C.1.7 Garbled Word Subsitution

Incomprehensive or garbled substitution words in the generated output are generally caused by:

- An improperly specified concatenation construction.
- Improperly initialized symbolic substitution words referenced in a concatenation construction or substituted directly.
- Inadvertent reference to the wrong variable name in a Line output record.

C.1.8 Improper Out-of-line Format

Improper out-of-line format is generally caused by:

- Failure to specify &ENV, &DATA, or &PROC where out-of-line substitution crosses division boundaries via an &POINT directive expression (i.e., &DATA &POINT...etc.).
- Substitution via &ANTE or &POST by other than Line output records.

C.1.9 Lost Out-of-line Words

Out-of-line substitution words appearing in the LOST OUT-OF-LINE listing are generally caused by:

- A missing division header in the source.
- A division header substituted out-of-line.
- Failure to drop an &MARKER.

C.1.10 CA-MetaCOBOL+ Translator in a Loop

Logical errors can cause the CA-MetaCOBOL+ Translator to loop continuously. This condition is generally caused by:

- A closed loop in a macro model, with no provisions for exit.
- A closed loop in a macro model, with improper testing of a counting variable. For example, the following loop in logic never exits because &VX overflows from 9 to 1 and, therefore, never is greater than 9.

```
     .
     .
     .
&LOCAL &VX 9
&SET &VX = 1
&REPEAT &UNTIL &VX > 9
     .
     .
     .
     &SET &VX = &VX + 1
&ENDREP
     .
     .
     .
```

- An "&SCANF &n" loop, where "&n" is never updated to reflect the previous parent in the data structure.

C.1.11 Data-name Undefined

Can be caused by missing period on item previously defined in the DATA DIVISION.

C.2 User-defined Macro Debugging Aids

Following logical analysis of test results, it is sometimes necessary to verify macro calls, current values of symbolic words, and other conditions which are difficult to debug at the source level. &NOTE directives and Line output records are essential tools which the macro writer can use to define the necessary debugging aids.

C.2.1 Macro Call Verification

&NOTE directives are the simplest means for verifying macro calls. For example, a $-LEVEL macro does not appear to be processing certain data item definitions. The results displayed on the INPUT & DIAGNOSTICS listing indicate that the event which calls the $-LEVEL macro -- a period preceding a numeric literal -- is disabled following a group item definition. This indicates that the third word, if a period, is removed from the source, as illustrated below:

CA-MetaCOBOL+ Input Listing:

```
          SD   $-LEVEL :
                    &GET &1
                    &STOW
                    &1
                    &NOTE &( 'LEVEL ' &1 ' CALLED' &)
               .
               .
               .
           DATA DIVISION.
               .
               .
               .
           WORKING-STORAGE SECTION.
           01  RECORD-A.
****NOTE   N99 LEVEL 01 CALLED
             02 ITEM-02.
               03 ELEM-03 PIC X.
****NOTE   N99 LEVEL 03 CALLED
               .
               .
               .
```

C.2.2 Current Value Verification

The &NOTE directive and Line output records can be used to display the current values of symbolic words for logical verification. For example, a Word-type macro can be defined to dump the current contents of global variables when that word is used as a substitution word, as follows:

CA-MetaCOBOL+ Input Listing:

```
          WI   $$GLOBAL :
                    &GLOBAL &VA X(30)
                    &GLOBAL &VB 9(2)

          W   @DEBUG :
                    &NOTE &( '&VA=' &VA &)
                    &NOTE &( '&VB=' &VB &)

          W   DIVISION :
                    DIVISION
                    @DEBUG

          S   $PDX :
                    @DEBUG
            .
            .
            .
           IDENTIFICATION DIVISION.
****NOTE N99 &VA =
****NOTE N99 &VB = 0
            .
            .
            .
           ENVIRONMENT DIVISION.
****NOTE N99 &VA = XYZ
****NOTE N99 &VB = 21
            .
            .
            .
****NOTE N99 &VA = ABC
****NOTE N99 &VB = 40
```

When developing complex macro sets involving tables of interrelated variables, the development of such debugging aids should be pre-planned. Original test results thus provide debugging information to enhance logical analysis.

C.3 CA-MetaCOBOL+ Debugging Aid Facility

A macro debugging aid facility is an integral part of the CA-MetaCOBOL+ Translator and can be used to trace logical macro errors. In order to activate the debugging aid, the macro writer must define both the extent and level of logical tracing.

C.3.1 Definition of Macros to be Traced

Two Translator-directing statements must be inserted in the macro set to define the macros to be traced.

Format:

```
{*$TRACE   }
{*$NOTRACE}
```

***$TRACE**
is a command in columns 7-13 that specifies all subsequent macros are to be traced until detection of an *$NOTRACE command in columns 7-15.

***$NOTRACE**
is a command that terminates *$TRACE.

Note: These statements are ignored if the translate-time option TRACE= is not specified.

C.3.2 TRACE= (the Translate-time Option)

Macro debugging can be invoked for all macros defined between *$TRACE and *$NOTRACE commands by specifying the TRACE= translate-time option.

Format:

TRACE=[D] [G] [S] [T]

TRACE=D
specifies that all "Nnn" class Translator diagnostics (as defined in the CA-MetaCOBOL+ *User Guide*), including NOTE and FLAG diagnostics, display an additional message indicating the macro line number which caused the diagnostic, provided the macro being executed is subject to tracing. The format of the message is:

```
#### TRACE- DIAGNOSTIC IN MACRO LINE #nnnn
```

TRACE=G
specifies that all source words acquired via an &GET directive expression display a message indicating the macro line number containing the &GET, the Type attribute, and the word acquired, provided the macro being executed is subject to tracing. The format of the message is:

```
#### TRACE- &GET IN MACRO LINE #nnnn 't' @word@
```

TRACE=S
specifies that all data items acquired via a &SCAN-type directive expression display a message indicating the macro line number containing the directive and the name of the data item acquired, provided the macro being executed is subject to tracing. The format of the message is:

```
                {&SCAN }
#### TRACE- {&SCANC}
                {&SCANF}  IN MACRO LINE #nnnn @word@
                {&SCANI}
```

TRACE=T
specifies that each executed transfer of control displays a message indicating that macro line number containing the transfer, provided the macro being executed is subject to tracing. The format of the message is:

```
#### TRACE- TRANSFER IN MACRO LINE #nnnn
```

Note: The operands of the TRACE= option can be specified in combination and in any order (e.g., TRACE=TGDS).

C.3.3 Diagnostic Tracing

CA-MetaCOBOL+ ERRORs, WARNINGs, NOTEs, and FLAGs are numbered diagnostics, as listed in the CA-MetaCOBOL+ *User Guide*. For example, all ERRORs and WARNINGs representing macro syntax errors and displayed during macro loading are encoded "Cnn", where nn is the number of the diagnostic. Other classes of diagnostics are displayed during the option analysis, memory management, input parsing, and macro translation executions.

Diagnostics encoded "Nnn", including NOTEs and FLAGs, are displayed during macro execution and represent conditions in macro logic. Since this category of diagnostics is displayed following a source statement, but actually represents a condition in a macro, it is often difficult to determine which macro statement caused the diagnostic during translation. The TRACE=D option is included to provide the necessary correlation between N-class diagnostics and the macro statements which cause them.

As an example of TRACE=D debugging, assume that a complex $-LEVEL macro produces the diagnostic "N04 NON-NUMERIC DATA IN &SET OR &IF" in the source DATA DIVISION. Logical analysis of the $-LEVEL macro cannot isolate the &SET or &IF directive expression which has caused the error. The following example uses the TRACE=D option to determine the macro statement number:

CA-MetaCOBOL+ Input Listing:

```
0050        *$TRACE
0051        SD   $-LEVEL :
              .
              .
              .
0090                  &SET &VX = &VY + 1
              .
              .
              .
0120        *$NOTRACE
              .
              .
              .
0400         DATA DIVISION.
0401         WORKING-STORAGE SECTION.
0402         77 R-COUNT  PIC S9(5)V99 COMP-3 VALUE +0.
**** ERROR N04 NON-NUMERIC DATA IN &SET OR &IF
#### TRACE- DIAGNOSTIC IN MACRO LINE #0090
              .
              .
              .
```

The preceding example indicates that the error is caused by execution of line 0090; either &VY contains a non-numeric current value or &VX is defined as non-numeric.

C.3.4 &GET Tracing

Continued source acquisition via &GET and &STORE/&STOW directives in String-type macro models is often difficult to debug. The primary reason for this difficulty is because the words passed to macro translation from input parsing are not readily apparent from an examination of the source statement. Remember that Word- and Prefix-type macros can be expanded when acquired via the &GET, and that certain dummy words, such as the Area A indicator, can be detected. Improper termination testing can result in premature exit from the macro or a runaway &GET/&STOW loop which continues through end-of-input and produces the "H91 READ PAST END OF INPUT" diagnostic as a FATAL ERROR. The TRACE=G option is a convenient method for examining the actual words obtained via the &GET directive expression. The TRACE=G option can be used to isolate the cause of improper &GET and &STORE/&STOW termination, inability to call a String-type macro, or a runaway source acquisition loop.

As an example, assume that a BLOCK CONTAINS String-type macro is not called in a complex macro set. An FD macro can be written to determine the cause and to ignore all other String-type macro calls, as follows:

CA-MetaCOBOL+ Input Listing:

```
0010      WD   CONTAINS :
             .
             .
             .
0400      SD   BLOCK CONTAINS &1 RECORDS :
             .
             .
             .
0600      *$TRACE
0601      SD   FD :
0602           FD  &REPEAT
0603                   &GET &1
0604                   &STOW
0605                   &1
0606               &UNTIL &1'T = 'V' &AND &1 = '.'
0607               &ENDREP
0608      *$NOTRACE
             .
             .
             .
0900       FD  MASTER-FILE
#### TRACE-      &GET IN MACRO LINE #0603 ' ' @MASTER-FILE@
0901           BLOCK CONTAINS 10 RECORDS
#### TRACE-      &GET IN MACRO LINE #0603 'S' @BLOCK@
#### TRACE-      &GET IN MACRO LINE #0603 'L' @10@
#### TRACE-      &GET IN MACRO LINE #0603 ' ' @RECORDS@
```

Notice in the preceding example that the Type attribute of the word BLOCK is 'S' for a String-type macro-name, instead of 'V' for a DATA DIVISION verb, and that the word CONTAINS is not acquired, since the Word-type macro on line 0010 removes it. The BLOCK CONTAINS macro, therefore, cannot be called.

C.3.5 &SCAN-type Directive Tracing

Data structure analysis via &SCAN-type directives is often difficult to check if code is generated conditionally for only some items. When no code is generated, it may not be apparent just what data items were acquired.

As an example of TRACE=S debugging, assume that a $PDE macro is supposed to generate code to clear table items in Working-Storage but is not generating any code. The following example uses the TRACE=S option to determine if all of Working-Storage is being acquired:

CA-MetaCOBOL+ Input:

```
           .
           .
           .
0499        *$TRACE
0500        SP $PDE :
0501           &REPEAT
0502               &SCAN WORKING-STORAGE WORKING-STORAGE
0503           &UNTIL ENDSCAN
           .
           .
           .
0600           &ENDREP
           .
           .
           .
0900         WORKING-STORAGE SECTION.
0901         01 COUNTERS USAGE COMP-3.
0902            02 COUNTER-1 PIC S9(5) VALUE +0.
           .
           .
           .
#### TRACE- &SCAN IN MACRO LINE #502 @WORKING-STORAGE@
#### TRACE- &SCAN IN MACRO LINE #502 @COUNTERS@
```

Since no items beyond COUNTERS appear in the TRACE listing, the &SCAN must be terminating prematurely. To do this, there must be a branch out of the &SCAN loop in the $PDE macro.

C.3.6 Transfer-of-Control Tracing

Logical transfers to control within macro models are often difficult to analyze and debug, especially if the CA-MetaCOBOL+ Translator is looping continuously in an undetermined macro model. The TRACE=T option is the most convenient technique for determining the location of such logical errors.

As an example, assume that CA-MetaCOBOL+ is looping continuously during translation of the DATA DIVISION. The following example indicates that the closed loop is in the &REPEAT construct in the $-LEVEL macro. The variable &VX is not large enough.

CA-MetaCOBOL+ Input Listing:

```
0300       *$TRACE
             .
             .
             .
0400       SD   $-LEVEL :
0401            &LOCAL &VX 9
             .
             .
             .
0459            &REPEAT
0460                &SET &VX = &VX + 1
0461            &UNTIL &VX > 9
0462            &ENDREP
0463            &SET &VX = 0
             .
             .
             .
0600       *$NOTRACE
             .
             .
             .
0700        WORKING-STORAGE SECTION.
0701        77  ITEM-A  PIC X.
#### TRACE-       TRANSFER IN MACRO LINE #0410
#### TRACE-       TRANSFER IN MACRO LINE #0417
#### TRACE-       TRANSFER IN MACRO LINE #0440
#### TRACE-       TRANSFER IN MACRO LINE #0463
#### TRACE-       TRANSFER IN MACRO LINE #0463
#### TRACE-       TRANSFER IN MACRO LINE #0463
                  (previous line repeats until job is cancelled)
```

When control is transferred during tracing, the TRACE directive displays the line number of the first macro code line that will not be executed as a result of the transfer. The TRACE directive does not display the line number of the macro code to which control has been transferred.

The following shows an example of this processing in a case situation. Note that lines 00007, 00015, 00020, 00023, and 00029 were not the lines displayed by the TRACE directive, even though they were the lines to which control was transferred.

CA-MetaCOBOL+ Input Listing:

```
00001     *$TRACE
00002     SP   TEST-TRACE :
00003          EQU &9 'B'
00004          &SELECT
00005            &WHEN &9 EQ 'A'
00006              &NOTE &( 'At the WHEN "A" condition' & <-- * see below
00007            &WHEN &9 EQ 'B'
00008              &NOTE &( 'At the WHEN "B" condition' &)
00009            &WHEN &9 EQ 'C'      <---------------------- * see below
00010              &NOTE &( 'At the WHEN "C" condition' &)
00011            &WHEN &9 EQ 'D'
00012              &NOTE &( 'At the WHEN "D" condition' &)
00013            &WHEN &9 EQ 'E'
00014              &NOTE &( 'At the WHEN "E" condition' &)
00015            &WHEN ANY
00016              &NOTE &( 'At the WHEN ANY condition' &)
00017            &WHEN OTHER          <---------------------- * see below
00018              &NOTE &( 'At the WHEN OTHER condition' &)
00019          &ENDSEL
00020          &IF &9 EQ 'A'
00021            &NOTE &( 'At IF &9 EQ "A"' &)  <------------ * see below
00022          &ENDIF
00023          &IF &9 EQ 'B'
00024            &NOTE &( 'At IF &9 EQ "B"' &)
00025          &ENDIF
00026          &IF &9 EQ 'C'
00027            &NOTE &( 'At IF &9 EQ "C"' &)  <------------ * see below
00028          &ENDIF
00029          &GOBACK
00030          PROCEDURE DIVISION.
00031              TEST-TRACE.
#######                 TRACE-      TRANSFER IN MACRO LINE # 00006
******* NOTE      N99  At the WHEN "B" condition
#######                 TRACE-      TRANSFER IN MACRO LINE # 00009
******* NOTE      N99  At the WHEN ANY condition
#######                 TRACE-      TRANSFER IN MACRO LINE # 00017
#######                 TRACE-      TRANSFER IN MACRO LINE # 00021
******* NOTE      N99  At IF &9 EQ "B"
#######                 TRACE-      TRANSFER IN MACRO LINE # 00027
00032               GOBACK.
```

***** The marked lines are not executed because a transfer of control takes place on the lines immediately preceding them. For example, line 6 is not executed because a transfer of control takes place when line 5 is executed. Similarly, line 9 is not executed because a transfer of control takes place when line 8 is executed.

When a transfer of control takes place, more than one line may not be executed. Line 5 transfers control to line 7; therefore, line 6 is not executed. However, line 8 transfers control to line 15, so lines 9-14 are not executed.

Appendix D. Directives Retained for Compatibility

CA-MetaCOBOL+'s macro language has evolved and improved over time. Often, improvements represent a better way to accomplish a logical task, providing a "preferred method." In a few instances, the old form remains in the language for compatibility purposes, but is no longer recommended for the development of modern macros.

This appendix contains descriptions of the "un-preferred" old forms.

D.1 &FLAG

The &FLAG directive displays a macro-name.

Format:

```
&FLAG
```

&FLAG
is a directive which allows the user to output a MCT diagnostic message from a macro model. This message appears on the INPUT & DIAGNOSTICS listing and/or the OUTPUT listing depending on the NOTE option (see the CA-MetaCOBOL+ *User Guide)*.

Notes: The &FLAG directive output is not cancelled by the NOLISTIN and NOLISTOUT translate-time options or by the *$NOLIST Translator-directing statement.

Execution of this directive sets the tentative condition code to 4.

Example: The &FLAG causes the name of the current macro to be printed as part of CA-MetaCOBOL+ message N98, as follows:

```
WP   EXHIBIT :
     &FLAG
     EXHIBIT
     .
     .
     .
 PROCEDURE DIVISION.
     .
     .
     .
     EXHIBIT FIELD-A
*****  FLAG N98 EXHIBIT
```

In this Word-type definition, the &FLAG causes the first word of the prototype (EXHIBIT) to appear within a diagnostic message. The remainder of the model (EXHIBIT) replaces each occurrence of the word EXHIBIT in the source program PROCEDURE DIVISION.

D.2 &IF

The &IF directive does comparisons and conditional branching.

Format:

```
                  {&Tname}
 &IF condition    {&Lname}
                  {&T$   }
```

&IF
is a directive which is similar in function to the COBOL "IF" statement.

condition
is a simple or combined condition. (Refer to Section 5.3.)

&Tname
is a tag known only to the macro model in which it appears.

&Lname
is a label known to all models of all macros within a region.

&T$
is a predefined tag that represents the end of the macro in which it appears, and is used to branch out of a macro, ending the call to that macro. It cannot be defined by the user.

Note: If the condition is true, macro processing continues at the specified destination name or the macro is terminated (&T$). If the condition is false, the macro call continues consecutively, even if the specified destination is undefined.

Example:
```
&SET &VWORK = &1'D + &1'S
&IF &VWORK > 80 &TGO
```

In the preceding example, if the sum of the displacement and size of &1 exceeds 80, a branch to &TGO is made and the macro call continues from that point, otherwise, the macro call continues at the next model word.

D.3 &JUMP/&JEND

The &JUMP directive does conditional transfer of control with the destination list based on the value in &Vname, it is terminated by &JEND.

Format:

```
                  {&Tname-1}  {&Tname-2}
 &JUMP &Vname     {&Lname-1}  {&Lname-2} ... &JEND
                  {&T$     }  {&T$     }
```

&JUMP
is a directive that is similar in function to the COBOL "GO TO...DEPENDING ON..." statement.

&Vname
is the name of a numeric variable.

&Tname
is a tag known only to the macro model in which it appears.

&Lname
is a label known to all models of all macros within a region.

&T$
is a predefined tag that represents the end of the macro in which it appears, and is used to branch out of a macro, ending the call to that macro. It cannot be defined by the user.

&JEND
is a directive that ends the destination list.

Notes: A list of logical destinations is included in the &JUMP directive expression. When executed, the macro call continues at the nth destination in the list, where n is the current value of &Vname.

If the value of the variable is less than 1 or greater than the number of logical destinations in the list, a diagnostic is issued and processing of the current macro is terminated. Therefore, a valid value must be placed in the variable &V*name* before the &JUMP is executed.

Example: In the following, action to be taken in the $-LEVEL macro depends on the current DATA DIVISION section.

```
WD   FILE :
         &GLOBAL &VSECTION 9
         &SET &VSECTION = 1
     .
     .
     .
WD   WORKING-STORAGE :
         &SET &VSECTION = 2
     .
     .
     .
WD   LINKAGE :
         &SET &VSECTION = 3
     .
     .
     .
SD   $-LEVEL :
     .
     .
     .
         &JUMP &VSECTION
             &T$
             &TA
             &TB
         &JEND
     .
     .
     .
```

D.4 &SCAN

The &SCAN directive retrieves data-names in order of definition and places data-names in &0.

Format:

```
        {&Vname-1 }  {&Vname-2  }  {&Tname}
&SCAN   {&n-1     }  {&n-2      }  {&Lname}
        {section-1}  {section-2 }  {&T$   }
```

Refer to the preferred form of the &SCAN directive in Section 5.16.

&Tname
is a tag known only to the macro model in which it appears.

&Lname
is a label known to all models of all macros within a region.

&T$
is a predefined tag that represents the end of the macro in which it appears, and is used to branch out of a macro, ending the call to that macro. It cannot be defined by the user.

Note: This form of the &SCAN automatically branches to the logical destination specified when no subordinate items are available in the defined range of data items (the "at end" condition).

D.5 &SCANC

The &SCANC directive retrieves the condition-name associated with &n and places the condition-name in &0.

Format:

```
            {&Tname}
&SCANC &n   {&Lname}
            {&T$   }
```

Refer to the preferred form of the &SCANC directive in Section 5.16.

&Tname
is a tag known only to the macro model in which it appears.

&Lname
is a label known to all models of all macros within a region.

&T$
is a predefined tag that represents the end of the macro in which it appears, and is used to branch out of a macro, ending the call to that macro. It cannot be defined by the user.

Note: This form of the &SCANC automatically branches to the logical destination specified when no more condition-names are associated with &n (the "at end" condition).

D.6 &SCANF

The &SCANF directive retrieves the next higher data-name or the data-name defining an index or the data-name associated with a condition-name, and places the name in &0.

Format:

```
                {&Tname}
&SCANC &n       {&Lname}
                {&T$   }
```

Refer to the preferred form of the &SCANC directive in Section 5.16.

&Tname
is a tag known only to the macro model in which it appears.

&Lname
is a label known to all models of all macros within a region.

&T$
is a predefined tag that represents the end of the macro in which it appears, and is used to branch out of a macro, ending the call to that macro. It cannot be defined by the user.

Note: This form of the &SCANF automatically branches to the logical destination specified when there are no more parents in the data hierarchy (the "at end" condition).

D.7 &SCANI

The &SCANI directive retrieves index-names associated with &n and places index-names in &0.

Format:

```
                {&Tname}
&SCANI  &n      {&Lname}
                {&T$   }
```

Refer to the preferred form of the &SCANC directive in Section 5.16.

&Tname
is a tag known only to the macro model in which it appears.

&Lname
is a label known to all models of all macros within a region.

&T$
is a predefined tag that represents the end of the macro in which it appears, and is used to branch out of a macro, ending the call to that macro. It cannot be defined by the user.

Note: This form of the &SCANI automatically branches to the logical destination specified when no more index-names are associated with &n (the "at end" condition).

Index

*

@

A

B

C

D

E

F

G

I

L

M

N

O

P

R

S

T

U

V

W

X

Z