CA-MetaCOBOL™ +

User Guide

Release 1.1

I101M+11IGP

-- PROPRIETARY AND CONFIDENTIAL INFORMATION --

Release 1.1, January, 1992
Updated March, 1992, May, 1992

Contents

Figures

1. About This Manual

CA-MetaCOBOL+ solves a broad range of COBOL program development and maintenance problems:

Structured Programming:	CA-MetaCOBOL+ provides constructs and utilities that enable you to code COBOL programs that are truly structured.
Accessing CA DATACOM/DB:	CA-MetaCOBOL+ provides data manipulation verbs for accessing CA DATACOM/DB databases. CA-MetaCOBOL+ also has an interface to CA DATADICTIONARY, which enables programs to extract necessary data and definitions.
Quality Assurance:	CA-MetaCOBOL+ audits and corrects COBOL programs to assure consistent formatting, documentation, and correct usage of COBOL syntax.
Work Bench:	CA-MetaCOBOL+ provides an online workstation through the CA-MetaCOBOL+ dialogs. The dialogs also extend the functionality of your ROSCOE, VOLLIE, and ISPF/PDF programming editor.
Macro Facility:	CA-MetaCOBOL+ can be customized with its macro language to solve a broad range of specific problems. This flexibility makes CA-MetaCOBOL+ unique among COBOL pre-processors, which traditionally have been limited to specialized tasks.

Refer to the publication *Introduction to CA-MetaCOBOL+* for a general description of the uses and organization of CA-MetaCOBOL+, and a list of prerequisite software.

1.1 Purpose

This manual describes how to customize, get started, and use CA-MetaCOBOL+. It includes information on keyword expansion, the CA-MetaCOBOL+ translator, and CA macro sets and programs.

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

1.2 Organization

The *User Guide* 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 how to access CA-MetaCOBOL+ under CA ROSCOE, CA VOLLIE, or ISPF/PDF.
3	Gives a brief explanation of the CA-MetaCOBOL+ facilities, including program shell generation and procedure code generation.
4	Explains how to use keyword expansion for Data Division keywords and Procedure Division keywords.
5	Describes how to customize CA-MetaCOBOL+ to meet your programming needs.
6	Discusses the features of the CA-MetaCOBOL+ Translator.
7	Gives a directory of the standard programs and macro sets provided with CA-MetaCOBOL+.
Appendix A	Glossary. defines common terms in CA-MetaCOBOL+.
Appendix B	Defines CA-MetaCOBOL+ reserved words.
Appendix C	Provides instruction for controlling the output format of your programs.
Appendix D	Lists and explains the messages that may appear while you are using CA-MetaCOBOL+.
Appendix E	Describes how to use the Account Management Review, which can be used to provide automatic status reports and to evaluate the use of CA-MetaCOBOL+ by the programming staff.

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 Supplement - MVS	Explains how to install CA MetaCOBOL+ in the 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.
Structured Programming Guide	Introduces the Structured Programming Facility. Includes information on creating, testing, and maintaining structured programs.
Macro Facility Tutorial	Introduces the Macro Facility. Includes information on writing basic macros, model programming, macro writing techniques, and debugging.
Macro Facility Reference	Includes detailed information on the program flow of the CA MetaCOBOL+ macro translator, macro format, definition of comments, macro nesting, macro prototypes, symbolic words, and model programming.
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 DATACOB/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.

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 Supplement - MVS	Explains how to install CA MetaCOBOL+ in the 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.
Online Programming Language Guide	Contains further instructions for using the Online Programming Language.
PC User Guide	Explains how to customize, get started, and use CA MetaCOBOL+/PC. Includes information on keyword expansion, the CA MetaCOBOL+/PC translator, and CA macro sets and programs.
Program Development Guide CA DATACOM/PC	Describes how to use CA MetaCOBOL+/PC with CA DATACOM/PC.
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	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 or represent 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

Information in Section 6.3, "How to Run a Translation," has been revised.

In Appendix C, "Output Format Control," information about the Procedure Division (Section C.1.4) has been updated.

Messages X15-X20 have been added to Appendix D, "Diagnostics." A few messages have been removed.

The Glossary has been updated.

Chapter 7 has been updated to include new information about macro sets. Section 7.2.3 describes the new String Manipulation Language (SML) macro set.

2. How to Get Started

This chapter tells you how to sign on to CA-MetaCOBOL+. It also describes PF keys and considerations for using CA-MetaCOBOL+ with your programming editor.

2.1 Under CA ROSCOE

Sign on to CA ROSCOE. From the command line, enter:

```
xxx.META
```

where *xxx* is the CA-MetaCOBOL+ prefix determined at installation.

CA-MetaCOBOL+ under CA ROSCOE enables you to make use of most of CA ROSCOE's features and functions. However, you must remember the following:

- PF3/15 (MAIN MENU) and PF5/17 (RETURN) update the member containing your program when you are editing the program under CA-MetaCOBOL+. To bypass the update, enter PF1/13 (EXIT).

- Do not assign the CA ROSCOE SCROLL and INPUT functions to PF1 through PF3, PF9, PF13 through P15, or PF21.

- Do not execute an RPF while using CA-MetaCOBOL+.

- You cannot use the following CA ROSCOE commands while using CA-MetaCOBOL+:

DELETE	EXIT	LET	OFFON
FETCH	EXEC	OFF	RUN

2.2 Under CA VOLLIE

Sign on to CA VOLLIE. From the command line, enter:

```
DEF OLL=filename
%META
```

where *filename* is the name of the CA-MetaCOBOL+ library.

CA-MetaCOBOL+ under CA VOLLIE enables you to make use of most of CA VOLLIE's features and functions. However, you must remember the following:

- Under CA VOLLIE use only the PF keys defined by CA-MetaCOBOL+. These include the SCROLL BACK, SCROLL FORWARD, TOP, BOTTOM and INSERT keys.

- Do not execute a VOICE procedure from CA-MetaCOBOL+.

- You can only issue the following CA VOLLIE commands from CA-MetaCOBOL+:

ADD	
ATTRIBUTE	(except SHA= or VOI= for the current member)
COPY	
DEFINE	(except DEF OLL=)
DELETE	(except the current member)
DIRECTORY	
DISPLAY	
DUPLICATE	
EDIT	
FILL	
HCF	
LIBRARY	
LIST	(except LIST member)
LMINDEX	
OPERATOR	
OUTPUT	
POSITION	
PAGE	(only when in output mode)
PRINT	
QUEUE	
RELEASE	
RENAME	(except when renaming current member)
SAVE	
SCAN	
STATUS	
SUBMIT	
TRANSFER	

2.3 Under ISPF/PDF

Select CA-MetaCOBOL+ from the ISPF/PDF Option Menu. If CA-MetaCOBOL+ libraries are allocated during signon, you can sign on by entering

```
ISPSTART PGM(META)
```

2.3.1 PF Keys

The first time you sign on, CA-MetaCOBOL+ makes copies of your ISPF function key assignments. Thereafter any changes you make to your PF keys under ISPF will not be applied to your CA-MetaCOBOL+ PF keys.

Because CA-MetaCOBOL+ maintains function keys separately from ISPF function keys, you must set CA-MetaCOBOL+ function keys separately from ISPF function keys. Change your CA-MetaCOBOL+ function key settings by issuing the KEYS command from the CA-MetaCOBOL+ EDIT panel.

CA-MetaCOBOL+ automatically assigns EXPAND, the keyword expansion command, to the lowest of PF keys 13 through 24 that is not bound to the END or HELP command. For example, if PF13 is bound to END and PF14 is bound to HELP, then PF15 will be automatically set to EXPAND.

You can use the PFSHOW TAILOR command to format your CA-MetaCOBOL+ PF key display. As with ISPF, PF key assignments are not displayed when NOSHOW is in effect.

2.3.2 CA-MetaCOBOL+ Edit Profile

The following edit profile is in effect throughout your CA-MetaCOBOL+ session. You can change an edit mode, but it will not be saved when you exit from CA-MetaCOBOL+:

```
ADRDL (FIXED 80)
AUTOLIST OFF
AUTONUM OFF
AUTOSAVE ON
CAPS ON
HEX OFF
IMACRO NONE
NOTE ON
NULLS ON ALL
NUMBER OFF
PACK OFF
PROFILE LOCK
RECOVERY OFF
STATS ON
TABS OFF
```

In addition, the following edit modes cannot be set from CA-MetaCOBOL+: AUTOSAVE OFF, RENUM, and NUMBER ON COBOL. Any edit mode with ON COBOL specified is disabled.

2.3.3 Member Naming Conventions

Like ISPF, CA-MetaCOBOL+ lets you specify members that are in ISPF or non-ISPF libraries. So if the member containing your COBOL program is in an ISPF library, enter the name in the PROJECT.GROUP.TYPE fields when CA-MetaCOBOL+ prompts for its name.

If the member is not in an ISPF library, enter the name in the OTHER DATA SET field. You can enter any fully qualified data set name by enclosing it in apostrophes. If you omit the apostrophes, your TSO prefix is appended to the left of the data set name.

3. CA-MetaCOBOL+ Facilities

The CA-MetaCOBOL+ Work Bench provides an online system to develop and maintain COBOL programs. It supplies menu-driven program generation and utility services for application programmers.

The Work Bench combines the features of your programming editor with CA-MetaCOBOL+ facilities to form an effective COBOL workstation. You can perform CA-MetaCOBOL+ tasks, and review listings, error messages, and test results in a single on-line session.

The first part of the Work Bench is the Dialog Facility. The CA-MetaCOBOL+ Dialog Main Menu groups services under the tasks listed on the menu:

```
||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||
||||||  |||  ||||||||||||||||||||||||'    '||'    '||     '||'    '||  |||||||
|||||   '   ||||||||||| ||||||||||||  ||  ||  ||  ||  ||' ||  ||  ||  |||  |||
||||  | |  ||' .. ||     ||'  ' |||  ||||||  ||  ||     <||  ||  ||  ||      |
|||  |||  || ....|||| |||| ||| |||  ||  ||  ||  ||  ||' ||  ||  ||  |||||  |||
||  |||  ||.    ||||.  ||.   . ||.    .||.    .||     .||.    .||     ||||||||
||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||
COPYRIGHT (C) 1989 COMPUTER ASSOCIATES INTERNATIONAL, INC.        Version 1.1

User:  name                                     Date: mm/dd/yy  Time: hh.mm.ss

Enter a selection code:  __

     PS - Program Shell Generation         QA - Quality Assurance
     FD - File Definition                  FM - Program Formatting
     DD - Data Definition                  PM - Profile Maintenance
     CG - Procedure Code Generation        JM - JCL Procedure Maintenance
     PC - Program Compilation              PD - Panel Definition



PF1/13: Exit    PF2/14: Help
```

Figure 1. Dialog Main Menu

Program Shell Generation
Select this option to build COBOL program shells.

File Definition
Select this option to specify files to be accessed by the program.

Data Definition
Select this option to build data descriptions for conventional COBOL or CA-MetaCOBOL+ programs.

Procedure Code Generation
Select this option to build procedural statements for conventional COBOL or CA-MetaCOBOL+ programs.

Program Compilation
Select this option to generate jobstreams to translate, compile, and link-edit your program, including precompiles.

Quality Assurance
Select this option to ensure that your site standards are adhered to in your COBOL programs.

Program Formatting
Select this option to format a program according to your site-defined standards.

Profile Maintenance
Select this option to specify or modify site-dependent global parameters that are used to generate JCL through the link-edit step.

JCL Procedure Maintenance
Select this option to maintain the JCL procedures used by the Program Compilation services.

Panel Definition
Select this option to access the Panel Definition Facility.

CA-MetaCOBOL+ guides you through each facility. It edits and verifies user responses to minimize errors that may occur in subsequent steps. There is a context-sensitive help facility. Following sections describe each facility in greater detail.

Note: Depending on the options installed, you may not be able to use all of the facilities listed on the main menu. If you have questions, ask your CA-MetaCOBOL+ administrator which facilities are available.

3.1 Program Shell Generation (PS)

Program Shell Generation creates the basic elements of a COBOL program based on your specification of program ID, program type (batch or online), coding convention, and type of data the program will access. The program shell is stored as a member in your library or on an optionally specified dataset.

For a newly created program, use the Program Shell Generation facility before you use any other facility on the main menu. First, specify PS, fill in the requisite fields, then press the Enter key. If you choose, you can check the program shell immediately without exiting CA-MetaCOBOL+.

As an example, below is a sample program shell for a program that will access CA DATACOM/DB. The lowercase italic words are program-specific information you supply.

```
 IDENTIFICATION DIVISION.
 PROGRAM-ID. PAYROLL.
*  THE FOLLOWING COMMENTS MUST NOT BE DELETED; THEY
*   ARE USED BY CA-MetaCOBOL+ AND MUST REMAIN INTACT
* <+ATTRIBUTES+>: TYP=B CDE=C LVL=M DBM=D
 AUTHOR. name.
 ENVIRONMENT DIVISION.
 CONFIGURATION SECTION.
 SOURCE-COMPUTER. IBM-370.
 OBJECT-COMPUTER. IBM-370.
 DATACOM SECTION.
     ID-AREA IS id-area-identifier
     PRINT NOGEN.
 INPUT-OUTPUT SECTION.
 FILE-CONTROL.
     SELECT file-name ASSIGN TO system-name.
 DATA DIVISION.
 FILE SECTION.
 FD  file-name     BLOCK CONTAINS integer RECORDS
                   RECORD CONTAINS integer CHARACTERS
                   LABEL RECORD IS STANDARD
                   DATA RECORD IS record-name.
 01  record-name.
 WORKING-STORAGE SECTION.
 01  id-area-identifier VALUE 'PAYROLL' PIC X(32).
 LINKAGE SECTION.
 PROCEDURE DIVISION.
 main-line-module.
     ENTER-DATACOM-DB
     process
     GOBACK
```

3.2 File Definition (FD)

The File Definition service enables you to name and describe the files your program will use. It then creates the required SELECT statements and file definitions for:

- Physical sequential files
- VSAM sequential files
- VSAM indexed files
- VSAM relative files
- SORT/MERGE files

From the main menu, specify FD, fill in the requisite fields, then press the Enter key.

The File Definition service generates FD and SD entries and corresponding 01-level record descriptions for each file, in the program's File Section. The format of each entry depends on the file type.

The service also generates appropriate SELECT statements in the Input-Output Section. The format of each entry depends on the file type. The order of file definition determines the order of SELECT statements.

3.3 Data Definition (DD)

The Data Definition service generates data descriptions for the Working-Storage Section and Linkage Section. This service enables you to use keyword expansion and your editor's power to build data descriptions. Keyword expansion is discussed in the following chapter.

From the main menu, specify DD, fill in the requisite fields, then press the Enter key. If you've selected a program, the service asks you to confirm your choice. This service also provides a subpanel to define CA DATACOM/DB dataviews.

3.4 Procedure Code Generation (CG)

The Procedure Code Generation service builds statements for the Procedure Division. This service enables you to use keyword expansion and your editor's power to create constructs in the Procedure Division. Keyword expansion is discussed in the following chapter.

From the main menu, specify CG, fill in the requisite fields, then press the Enter key. If you've selected a program, the service asks you to confirm your choice.

3.5 Program Compilation (PC)

The Program Compilation service generates JCL to translate, compile, and link-edit the program. This service automatically creates a jobstream using the selected program's attributes and information you entered in the CA-MetaCOBOL+ Profile. You can review and edit the jobstream before you submit it, and you can save the jobstream in a library member.

You can compile a program contained in a member from an outside library (ROSCOE), from a shareable library (VOLLIE), or, if this feature is not restricted at your site, from a partitioned data set (ISPF/PDF).

From the main menu, specify PC, fill in the requisite fields, then press the Enter key. If you've selected a program, the service asks you to confirm your choice, then prompts you to fill in JOBCARD information. From there, you can either submit the code for compilation and link-editing, review the jobstream before submission, or save the program in a library member.

If you generate the program using the CA-MetaCOBOL+ Structured Programming Facility, you'll be asked to specify translation options. For more information on these options, refer to the CA-MetaCOBOL+ *Structured Programming Guide*.

3.6 Quality Assurance (QA)

The Quality Assurance (QA) Facility ensures that all of your site standards and requirements are implemented in your COBOL programs. This facility:

- **Enables Customization of Environment** to establish your organization's requirements for application development.
- **Enforces the Standards You Create** to ensure that requirements are always met.
- **Promotes Good Programming Practices** by flagging inefficient code and making programmers aware of structured programming techniques.
- **Controls Language Use** by checking for non-standard or nearly obsolete language elements.
- **Improves Readability** with automatic formatting services.
- **Documents Your Program** by producing reports.

From the main menu, specify QA, fill in the requisite fields on all subpanels, then press the Enter key. For more information, refer to the CA-MetaCOBOL+ *Quality Assurance Guide*.

3.7 Program Formatting (FM)

The Program Formatting service enables you to determine and provide consistent program formatting to help you read and debug COBOL programs. For example, you can specify automatic indents for continued sentences and conditional statements, and you can specify that each verb begins on a new line.

This service can format batch COBOL programs, programs that contain command level CICS statements, and programs that contain CA-MetaCOBOL+ high-level language extensions.

From the main menu, specify FM, fill in the requisite fields on all subpanels, then press the Enter key. The following example illustrates how CA-MetaCOBOL+ formatting can clearly express program logic:

Before:

```
CHECK-BATCH-CODE.
    IF BATCH-CODE = 'B' PERFORM BATCH-HEADER
    ELSE IF BATCH-CODE = 'A' OR 'C' OR 'D'
    PERFORM BATCH-DETAIL ELSE
    PERFORM BATCH-ERROR ENDIF ENDIF
```

After:

```
CHECK-BATCH-CODE.
    IF BATCH-CODE = 'B'
       PERFORM BATCH-HEADER
    ELSE
       IF BATCH-CODE = 'A' OR 'C' OR 'D'
          PERFORM BATCH-DETAIL
       ELSE
          PERFORM BATCH-ERROR
       ENDIF
    ENDIF
```

3.8 Profile Maintenance (PM)

The Profile Maintenance services help you create and maintain a profile of CA-MetaCOBOL+ session defaults for Job Control, CA-MetaCOBOL+ development, and CA DATACOM/DB facilities.

From the main menu, specify PM, fill in the requisite fields on all sub-menus, then press the Enter key. For example, to change your compiler option, select Job Control Information on the sub-menu, then change the compiler specification. The next time you use the Program Compilation services to compile a program, the JCL will include the new compiler option.

For more information on the CA-MetaCOBOL+ Profile, refer to Chapter 5, "Customizing CA-MetaCOBOL+."

3.9 JCL Procedure Maintenance (JM)

The JCL Procedure Maintenance services enable you to modify and update the JCL procedures used by the Program Compilation services. You can save these procedures in a library.

From the main menu, specify JM, fill in the requisite fields on all subpanels, then press the Enter key.

3.10 Panel Definition (PD)

The CA-MetaCOBOL+ Panel Definition Facility is a CICS panel painter and BMS map generator. With a panel definition as input, PDF/CICS can generate CICS BMS map source to be stored in a PDF/CICS member. Panel definitions can be displayed, edited, printed, duplicated, renamed, and deleted with PDF/CICS.

From the main menu, specify PD, fill in the requisite fields on all subpanels, then press the Enter key.

4. How to Use Keyword Expansion

The Data Definition and Procedure Code Generation services provide keyword expansion, which you can use to generate data items and procedural statements in your program.

Keyword expansion generates a template of a data description or a statement. In this case, a keyword is a short mnemonic for the statement it generates. You can generate the template by entering the keyword at the beginning of a line and pressing a PF key. After the template is generated, you can complete it by typing the appropriate information.

The generated template is syntactically correct and logically indented, making it less likely that an essential clause will be left out or misspelled. Multiple statement options are separated by slashes (/), and optional clauses are delimited by the less than (<) or the greater than (>) symbol.

Keyword expansion is available for all conventional COBOL, CA-MetaCOBOL+ syntax, command-level CICS, and SQL for CA DATACOM/DB. As an example, the keyword .for generates the syntax for the CA DATACOM/DB FOR statement:

Input:

```
.for
```

Output:

```
FOR EACH/FIRST/ANY <identifier-1/literal>
      data-view-name RECORDS
   <WHERE (condition)>
   <HOLD RECORDS>
   <COUNT IN identifier-2>
   <ORDER <UNIQUE> RECORDS ON
         <ASCENDING/DESCENDING> data-view-field-1
        <<ASCENDING/DESCENDING> data-view-field-2...>>
       <imperative statements>
   <WHEN NONE
        imperative statements>
   <WHEN END
        imperative statements>
   <WHEN ERROR
        imperative statements>
ENDFOR
```

4.1 The Keyword Expansion Process

Keyword expansion generates a model of a Procedure Division statement and Data Division data description. Because multiple statement options are separated by slashes (/) and optional clauses are delimited by the less than (<) or the greater than (>) symbol, you must delete these characters from the statement model. Available keywords are listed in Sections 4.2 and 4.3, and in the CA-MetaCOBOL+ Help facility.

Keyword Syntax

Each keyword must be preceded by a period (.), which is placed in Column 1, and can optionally be followed by a period (.), a duplication factor, and at least one blank character. The duplication factor is a number that specifies how many times you want the keyword expanded. The duplication factor must be greater than or equal to 1 and less than or equal to 20. The default is 1.

For example, if you specify .IF.2, then press the keyword expansion PF key, the following statements will be generated in the Procedure Division of a conventional COBOL program:

```
IF condition
    process
ELSE
    process

IF condition
    process
ELSE
    process
```

Indent Control

If a keyword is typed onto a line that begins with IF, ELSE, LOOP, UNTIL, WHEN, MODULE, or *module*, the expanded code is indented four more spaces than the current level of indentation; in other cases, the expanded code is indented to the same tab position as the current line, up to a maximum tab position of 36.

If a keyword is inserted on a new line, it is expanded from Column 8 or Column 12, depending on the keyword. If the keyword is typed on the same line as PROCESS, or *process*, the expanded code replaces the contents of the line with the same tab position.

For example, to insert an ADD and MOVE statement into a previously generated IF construct, enter .ADD and .MOVE starting in Column 1:

```
.ADD   IF condition
.MOVE      process
       ELSE
           process
```

After you press the keyword expansion PF key, the following construct is generated:

```
       IF condition
           ADD identifier/literal-1
               <identifier/literal-2>...
               TO/GIVING identifier-m <ROUNDED>
                         <identifier-n <ROUNDED>>...
           MOVE identifier-1/literal-1 TO identifier-2
                                          <identifier-3>...
       ELSE
           process
```

Generating Level Numbers

In expanded statement models resulting from Data Division keywords, the code lvl represents the COBOL level number. However, if you include a two-digit level number in the keyword, the data definition will contain the corresponding level in the generated data description. The specified level number must be greater than 0 and less than 50, and it must be placed before the keyword following the initial period.

For example, to replace lvl, start in Column 1 and code the following:

```
.05ALPHAMERIC.
```

After you press the keyword expansion PF key, the following data description is generated:

`05` *`data-name`* `PIC X(`*`nnnn`*`)`

If you specify an invalid level number, the data description is generated with the code lvl. For example, if you specify .54ALPHAMERIC., the result is:

`lvl data-name` `PIC X(`*`nnnn`*`)`

If you specify a level number for a keyword that does not require a keyword, the keyword is ignored.

4.2 Data Division Keywords That Can Be Expanded

Following are Data Definition keywords for conventional COBOL, SP COBOL, CA DATACOM/DB Facility, CA DATACOM/DB 8.0 or higher SQL, DB2 SQL, IDMS, IDMS-DC, and OPL. If a short form or alias exists, it is indented beneath the unabbreviated form of the keyword:

4.2.1 Conventional and SP COBOL Keywords

For All Programs

The following are keywords for conventional COBOL statement templates available for all programs, including Batch and VS COBOL II.

- .ALPHABETIC
  - .ALB
- .ALPHAMERIC
  - .ALM
- .BINARY
  - .BIN
  - .COMP
- .CONDITION
  - .COND
- .FILLER
  - .FILL
  - .F
- .GROUP
  - .GRP
- .GROUPRDF
  - .GRPRRDF
- .INDEX
- .NUMERIC
  - .NUM
  - .N
- .OCCURS
- .PACKED
  - .COMP-3
- .RECORD
  - .REC
- .USAGE
- .VALUE
  - .VAL

For Batch Programs Only

The following are keywords for conventional COBOL statement templates available only for programs with a program type attribute of BATCH.

.FD .SELSEQ .SELVSAM

For VS COBOL II Programs Only

The following are keywords for VS COBOL II statement templates only.

.DISPLAY-1
    .GRAPHIC

4.2.2 CA DATACOM/DB Facility Keywords

For DLM or MIXED

The following are keywords for CA-MetaCOBOL+ statement templates for programming with a Database Type Attribute of DATACOM and a Database Access Language attribute of either DLM or MIXED.

.DATAVIEW
    .DVW

For SQL or MIXED

The following are keywords for SQL statement templates for programs with a Database Type Attribute of DATACOM and a Database Access Language attribute of either SQL or MIXED.

.SQLDECLARE
    .QDC

4.2.3 DB2 Keywords

The following are keywords for SQL statement templates for programs with a Database Type Attribute of DB2.

.SQLCA
    .QCA
.SQLDECLARE
    .QDC

4.2.4 IDMS Keywords

The following are keywords for IDMS statement templates for all programs with a Database Type Attribute of IDMS and IDMS.

The following are keywords for CA IDMS statement templates for programs with a Database Type attribute of IDMS:

For All IDMS

.IDMSCONTROL
.PROTOCOL
.SCHEMA
.COPYIDMSLR
.COPYIDMSREC
.COPYIDMSSUB
.SSCTRL

For SQL Only

The following are keywords for SQL statement templates for programs with a Database Type attribute of IDMS:

.SQLCA,
.QCA
.SQLDECLARE
.QDC

For Batch Only

The following are keywords for CA IDMS statement templates for programs with a Program Type attribute of BATCH:

.FILE

4.2.5 IDMS-DC Keywords

The following are keywords for CA IDMS-DC statement templates for programs with an Online Monitor attribute of IDMS-DC:

.COPYIDMSMAP
.MAP

4.2.6 OPL Keywords

The following are keywords for CA-MetaCOBOL+ Online Programming Language (OPL) statement templates available for programs with the Online Monitor attrbute of CICS and designated as using the Online Programming Language.

.DISPLAY
.DIS

.MAPDEF
    .MAP
    .MD

.RESPONSEDEF
    .RESPDEF
    .RD

4.3 Procedure Division Keywords That Can Be Expanded

Following are Procedure Division keywords for SPF COBOL, conventional COBOL, command-level CICS, CA DATACOM/DB Facility, CA DATACOM/DB 8.0 SQL, and DB2 SQL. If a short form or alias exists, it is indented beneath the unabbreviated form of the keyword.

4.3.1 Conventional COBOL Keywords

The following are keywords for conventional COBOL statement templates available for programs with a Coding Convention Attribute of COBOL.

.ADD
.ADDSIZE
.COMPUTE
.COMPUTESIZE
    .COMPSIZE
.COPY
.COPYREP
    .COPYR
.DIVIDE
    .DIV
.DIVIDESIZE
.DIVSIZE

.IF
.INSPECT
.MODULE
    .MOD
.MOVE
.MULTIPLY
    .MULT
.MULTIPLYSIZE
    .MULTSIZE
.PERFORM
.PERFORMVARY
    .PERFVARY

.SEARCH
.SEARCHALL
.SET
.SORT
.STRING
.SUBTRACT
    .SUB
.SUBTRACTSIZE
    .SUBSIZE
.UNSTRING

For Batch Only

The following are keywords for conventional COBOL statement templates available for programs with a Coding Convention Attribute of COBOL and a Program Type Attribute of BATCH.

.CANCEL
.CLOSEFILE
    .CLOSE
.DELETEFILE
    .DELFILE
    .DELETE
    .DEL
.DELETEKEY
    .DELKEY
.DISPLAY
.MERGE

.OPENFILE
    .OPEN
.READFILE
    .READ
.READATEND
    .READSEQ
.READKEY
.RETURNFILE
    .RETFILE
.REWRITEFILE
    .REWRITE
.REWRITEKEY

.START
.STARTKEY
.USEBEFORE
    .USEREPORT
    .USERPT
.USEERROR
.USELABEL
.WRITEADV
.WRITEFILE
    .WRITE
.WRITEEOP
.WRITEKEY

For VS COBOL II Only

The following are keywords for conventional VS COBOL II statement templates available for programs with a Coding Convention Attribute of COBOL and a Compiler Dialect Attribute of COBOL2.

.EVALUATE
    .EVAL
.PERFORMI
    .PERFI
.PERFORMO
    .PERFO

4.3.2 SP COBOL Keywords

For All SP Keywords

The following are keywords for CA-MetaCOBOL+ Structured Programming (SP) statement templates available for programs with a Coding Convention Attribute of SP.

- .ADD
- .ADDSIZE
- .CALL
- .COMPUTE
- .COMPUTESIZE
  - .COMPSIZE
- .COPY
- .COPYREP
  - .COPYR
- .DIVIDE
  - .DIV
- .DIVIDESIZE
  - .DIVSIZE
- .DO
- .ESCAPE
- .EXIT
- .FLAG
- .IF
- .INSPECT
- .LOOP
- .LOOPTIMES
- .LOOPVAR
- .MODULE
  - .MOD
- .MOVE
- .MULTIPLY
  - .MULT
- .MULTIPLYSIZE
  - .MULTSIZE
- .PERFORM
- .PERFORMVARY
  - .PERFVARY
- .SEARCH
- .SEARCHALL
- .SELECT
- .SELECTEVERY
  - .SELEVERY
- .SELECTFOR
- .SELECTLEADING
  - .SELECTLEAD
  - .SELLEAD
- .SET
- .SETFALSE
- .SETTRUE
- .SORT
- .STARTDATA
- .STRING
- .SUBTRACT
  - .SUB
- .SUBTRACTSIZE
  - .SUBSIZE
- .UNSTRING

For Batch Only

The following are keywords for CA-MetaCOBOL+ Structured Programming (SP) statement templates available for programs with a Coding Convention Attribute of SP and a Program Type Attribute of BATCH.

- .CANCEL
- .CLOSEFILE
  - .CLOSE
- .DELETEFILE
  - .DELFILE
  - .DELETE
  - .DEL
- .DELETEKEY
  - .DELKEY
- .DISPLAY
- .MERGE
- .OPENFILE
  - .OPEN
- .READFILE
  - .READ
- .READATEND
  - .READSEQ
- .READKEY
- .RETURNFILE
  - .RETFILE
- .REWRITEFILE
  - .REWRITE
- .REWRITEKEY
- .START
- .STARTKEY
- .USEBEFORE
  - .USEREPORT
  - .USERPT
- .USEERROR
- .USELABEL
- .WRITEADV
- .WRITEFILE
  - .WRITE
- .WRITEEOP
- .WRITEKEY

4.3.3 CICS Command-Level Keywords

The following are keywords for all command-level CICS statement templates available for programs with an online monitor attribute of CICS.

.CONVERSE
.DELETECICS
    .DELCICS
.DELETEQTD
.DELETEQTS
.ENDBR
.FREEMAIN
.GETMAIN
.HANDLEAID
    .HANDAID
.HANDLECONDITION
    .HANDCOND
.LINK
.LOAD
.READCICS
.READNEXTCICS
    .READNCICS
.READPREV
.READQTD
.READQTS
.RECEIVE
.RECEIVEMAP
    .RECVMAP
    .RECMAP
.RELEASE
.RESETBR
.RETURN
    .CICSRETURN
.REWRITECICS
    .REWRCICS
.SEND
.SENDMAP
.SENDTEXT
.STARTBR
.UNLOCK
.WRITECICS
.WRITEQTD
.WRITEQTS
.XCTL

4.3.4 CA DATACOM/DB Facility Keywords

For DLM or Mixed

The following are keywords for command-level CICS statement templates available for programming with CA DATACOM available for programs with a Database Type Attribute of DATACOM and a Database Access Language attribute of either DLM or MIXED.

.LOCATENEXTRANGE
    .LOCNEXTWR
    .LNWR
.ABEND
.BACKOUT
.CHECKPOINT
    .CHKPOINT
    .CHKPNT
.CLOSE
.DELETE
    .DEL
.ENTERDB
.FOR
.FREEALL
.FREELAST
.FREESET
.INSERT
.LOCATE
    .LOC
.LOCATEAT
    .LOCAT
.LOCATENEXT
    .LOCNEXT
    .LOCNXT
.LOCATEPREV
    .LOCPREV
    .LOCPRE
.LOCATERANGE
    .LOCRANGE
    .LOCWR
    .LWR
.OBTAIN
.OBTAINNEXT
    .OBTNEXT
.OBTAINNEXTRANGE
    .OBTNEXTWR
    .ONWR
.OBTAINPREV
    .OBTPRE
.OBTAINRANGE
    .OBTWR
    .OWR
.READ
.READLOG
.READNEXT
.READNEXTRANGE
    .READNEXTWR
    .RNWR
.READPRE
.READRANGE
    .READWR
    .RWR
.REWRITE
.SETTEST
.UPDATE
.WRITE
.WRITELOG

For SQL or Mixed

The following are keywords for command-level CICS statement templates available for programming with CA DATACOM available for programs with a Database Type Attribute of DATACOM and a Database Access Language attribute of either SQL or MIXED.

The SQL keywords listed below can be used with either CA DATACOM/DB 8.0 or higher, or with DB2.

.SQLCLOSE
    .QCC
.SQLCOMMIT
    .QCM
.SQLDELETE
    .QDL
.SQLFETCH
    .QFC
.SQLINSERT
    .QIN
.SQLLOCK
    .QLK
.SQLOPEN
    .QOC
.SQLROLLBACK
    .QRL
.SQLSELECT
    .QES
.SQLSUBQUERY
    .QSQ
.SQLUNION
    .QUN
.SQLUPDATE
    .QUP
.SQLWHENEVER
    .QWH

For Batch Only

The following are keywords for command-level CICS statement templates available for programming with CA DATACOM available for programs with a Database Type Attribute of DATACOM, a Database Access Language attribute of either DLM or MIXED, and a Program Type Attribute of BATCH.

.LOCATEPHYS
    .LOCPHYS
.LOCATESEQ
    .LOCSEQ
.OBTAINPHYS
.OBTAINSEQ
.OPEN
.READPHYS
.READSEQ

4.3.5 DB2 Keywords

The following are keywords for SQL statement templates for programs with a Database Type Attribute of DB2.

.SQLCLOSE
    .QCC
.SQLCOMMIT
    .QCM
.SQLDELETE
    .QDL
.SQLEXECUTE
    .QXI
.SQLFETCH
    .QFC
.SQLID
    .QID
.SQLINSERT
    .QIN
.SQLLOCK
    .QLK
.SQLOPEN
    .QOC
.SQLROLLBACK
    .QRL
.SQLSELECT
    .QES
.SQLSUBQUERY
    .QSQ
.SQLUNION
    .QUN
.SQLUPDATE
    .QUP
.SQLWHENEVER
    .QWH

4.3.6 IDMS Keywords

For All IDMS

The following are keywords for IDMS statement templates available for programs with a Database Type attribute of IDMS:

- .ACCEPTBINDADD
  - .ACCBIND
  - .ACPTBND
- .ACCEPTDBKEY
- .ACCEPTDBSTATS
  - .ACCEPTSTATS
  - .ACCSTATS
- .ACCEPTPROCEDURE
  - .ACCEPTPROC
  - .ACCPROC
- .BINDPROCEDURE
  - .BINDPROC
- .BINDRECORD
  - .BINDREC
- .BINDRUNUNIT
  - .BINDRUN
- .COMMIT
- .CONNECT
  - .CONN
- .COPYIDMSMOD
  - .IDMSMOD
  - .MODULE
- .DELETEQUE
  - .DELQUE
  - .DELQ
- .DISCONNECT
  - .DISCONN
  - .DISCO
- .ERASE
- .ERASELRF
- .FINDCALC
- .FINDCURRENT
  - .FINDCUR
- .FINDDBKEY
  - .FINDDBK
- .FINDOWNER
  - .FINDOWN
- .FINDDUPLICATE
  - .FINDDUP
- .FINDFIRST
  - .FINDFIR
- .FINDLAST
- .FINDNNEXT
- .FINDPRIOR
  - .FINDPRI
- .FINDNBR
- .FINDSEQ
- .FINDSORT
- .FINDUSINGSORT
  - .FINDUSING
- .FINISH
- .GET
- .GETQUEUE
  - .GETQ
- .IFEMPTY
  - .IFNOTEMPTY
- .IFMEMBER
  - .IFNOTMEMBER
- .KEEP
- .MOD
- .MODIFY
- .MODIFYLRF
  - .MODLRF
- .OBTAINCALC
  - .OBTCALC
- .OBTAINCURRENT
  - .OBTCURRENT
  - .OBTAINCUR
  - .OBTCUR
- .OBTAINDBKEY
  - .OBTAINDBK
  - .OBTDBKEY
  - .OBTDBK
- .OBTAINDUPLICATE
  - .OBTAINDUP
  - .OBTDUP
- .OBTAINFIRST
  - .OBTAINFIR
  - .OBTFIRST
  - .OBTFIR
- .OBTAINLAST
  - .OBTLAST
- .OBTAINLRF
  - .OBTLRF
- .OBTAINNEXT
  - .OBTNEXT
- .OBTAINOWNER
  - .OBTAINOWN
  - .OBTOWNER
  - .OBTOWN
- .OBTAINPRIOR
  - .OBTAINPRI
  - .OBTPRIOR
  - .OBTPRI
- .OBTAINSEQ
  - .OBTSEQ
  - .OBTNBR
- .OBTAINUSINGSORT
  - .OBTAINSORT
  - .OBTSORT
  - .OBTUSING
- .PUTQUEUE
  - .PUTQ
- .READY
- .RETURN
- .ROLLBACK
- .STORE
- .STORELRF
- .SUBSCHEMABINDS
- .WHERE
- .WRITEPRINT
  - .WRITEPRINTER

For SQL Only

The following are keywords for IDMS statement templates for SQL statement templates available for programs with the Database Type attribute of IDMS:

- .SQLCLOSE
  - .QCC
- .SQLCOMMIT
  - .QCM
- .SQLDELETE
  - .QDL
- .SQLEXECUTE
  - .QXI
- .SQLFETCH
- .QFC
- .SQLID
  - .QID
- .SQLINSERT
  - .QIN
- .SQLLOCK
  - .QLK
- .SQLOPEN
  - .QOC
- .SQLROLLBACK
  - .QRL
- .SQLSELECT
  - .QES
- .SQLSUBQUERY
  - .QSQ
- .SQLUNION
  - .QUN
- .SQLUPDATE
  - .QUP
- .SQLWHENEVER
  - .QWH

For Batch Only

The following are keywords for IDMS statement templates available for programs with a Database Type attribute of IDMS and a Program Type attribute of BATCH:

.BINDTASK

4.3.7 IDMS-DC Keywords

The following are keywords for IDMS-DC statement templates available for programs with an Online Monitor attribute of IDMS-DC:

.INQUIREMAPCURSOR
    .INQMAPCURSOR
.ABEND
.ACCEPT
.ACCEPTTRANSSTATS
    .ACCTRANSTAT
    .ACCTRAN
.ATTACH
.BINDMAP
.BINDTRANSACTION
    .BINDTRANS
.CHANGEPRIORITY
    .CHAP
.CHECKTERMINAL
    .CHKTERM
.COMMIT
.DCRETURN
.DELETEQUE
    .DELQUE
    .DELQ
.DELETESCRATCH
    .DELSCR
.DELETETABLE
    .DELTAB
.DEQUE
.ENDLINETERMINAL
    .ENDLINE
.ENDPAGE
.ENDTRANSSTATS
.ENQUEUE
.FINISH
.FREESTORAGE
    .FREESTO
    .FREESTG
.GETQUEUE
    .GETQ
.GETSCRATCH
    .GETSCR
.GETSTORAGE
    .GETSTO
    .GETSTG
.GETTIME
.INQUIREMAP
    .INQMAP
.INQUIREMAPDATA
    .INQMAPDATA
.INQUIREMAPINPUT
    .INQMAPINPUT
.INQUIREMAPMOVE
    .INQMAPMOVE
.KEEPLONG
    .KEEPL
.LOADTABLE
    .LOADTAB
.MAPBINDS
.MAPIN
.MAPOUT
.MAPOUTIN
.MODIFYMAP
    .MODMAP
.POST
.PUTQUEUE
    .PUTQ
.PUTSCRATCH
    .PUTSCR
.READLINE
.READTERMINAL
    .READTERM
    .READT
.ROLLBACK
.SENDMESSAGE
    .SENDMSG
    .SEND
.SETABEND
    .SETEXIT
    .STAE
.SETTIMER
.SNAP
.STARTPAGE
.TRANSFER
.WAIT
.WHERE
.WRITEJOURNAL
    .WRITEJ
.WRITELINE
.WRITELOG
.WRITEPRINTER
    .WRITEPRINT
.WRITEREADTERM
    .WRITEREAD
.WRITETERMINAL
    .WRITETERM

4.3.8 OPL Keywords

The following are keywords for command-level CICS statement templates available for programs with an online monitor attribute of CICS and designated as using the Online Programming Language.

.DELETEGLOBAL
    .DELETEGBL
.DISPLAY
    .DIS
.SETALARM
.SETERASE
.SETMAP
.SETATTRIBUTES
    .SETATTRS
    .SETATTR
.SETCOLOR
.SETHIGHLIGHT
    .SETHIGH
    .SETHI
.SETCURSOR
    .SETCSR
    .SETCUR
.TRANSFER
.TRANSMIT
    .TR
.VALIDATEMAP
    .VALIDATEM
    .VALMAP
.VALIDATEFIELD
    .VALIDATEF
    .VALFLD

4.3.9 SQL Keywords

The following are SQL keywords for CA DATACOM/DB, DB2, and IDMS.

For CA DATACOM/DB

The following are keywords for command-level CICS statement templates available for programming with CA DATACOM available for programs with a Database Type Attribute of DATACOM and a Database Access Language attribute of either SQL or MIXED. The SQL keywords listed below can be used with either CA DATACOM/DB 8.0 or higher, or with DB2.

.SQLCLOSE
    .QCC
.SQLCOMMIT
    .QCM
.SQLDELETE
    .QDL
.SQLFETCH
    .QFC
.SQLINSERT
    .QIN
.SQLLOCK
    .QLK
.SQLOPEN
    .QOC
.SQLROLLBACK
    .QRL
.SQLSELECT
    .QES
.SQLSUBQUERY
    .QSQ
.SQLUNION
    .QUN
.SQLUPDATE
    .QUP
.SQLWHENEVER
    .QWH

For DB2

The following are keywords for SQL statement templates for programs with a Database Type Attribute of DB2.

.SQLCLOSE
.QCC
.SQLCOMMIT
.QCM
.SQLDELETE
.QDL
.SQLEXECUTE
.QXI
.SQLFETCH
.QFC
.SQLID
.QID
.SQLINSERT
.QIN
.SQLLOCK
.QLK
.SQLOPEN
.QOC
.SQLROLLBACK
.QRL
.SQLSELECT
.QES
.SQLSUBQUERY
.QSQ
.SQLUNION
.QUN
.SQLUPDATE
.QUP
.SQLWHENEVER
.QWH

For IDMS

The following are keywords for IDMS statement templates for SQL statement templates available for programs with the Database Type attribute of IDMS:

.SQLCLOSE
.QCC
.SQLCOMMIT
.QCM
.SQLDELETE
.QDL
.SQLEXECUTE
.QXI
.SQLFETCH
.QFC
.SQLID
.QID
.SQLINSERT
.QIN
.SQLLOCK
.QLK
.SQLOPEN
.QOC
.SQLROLLBACK
.QRL
.SQLSELECT
.QES
.SQLSUBQUERY
.QSQ
.SQLUNION
.QUN
.SQLUPDATE
.QUP
.SQLWHENEVER
.QWH

For Batch Only

The following are keywords for IDMS statement templates available for programs with a Database Type attribute of IDMS and a Program Type attribute of BATCH:

.BINDTASK

5. Customizing CA-MetaCOBOL+

The Profile Table enables you to specify or modify site-dependent global parameters that are used to generate JCL through the link-edit step. You can specify job card information, JCL procedures, CA-MetaCOBOL+ Translator Options, stub generator macros, formatting macros, optimization macros, and the like. All of the information in this one location can ensure the consistency of all the code generated at your site.

During installation, the systems programmer will specify defaults for your site. If your programming environment or site requirements change, the CA-MetaCOBOL+ Dialog Facility provides Profile Maintenance services to help you create and maintain a profile of CA-MetaCOBOL+ session defaults for Job Control, CA-MetaCOBOL+ development, and CA DATACOM/DB facilities.

For example, to change your compiler option, access the Dialog Facility Main Menu, select Profile Maintenance (PM) which puts you in a sub-menu, select Job Control Information on the sub-menu, then change the compiler specification. The next time you use the Job Submission services to compile a program, the JCL will include the new compiler option.

For more information on the CA-MetaCOBOL+ Dialog Facility, please refer to Chapter 3, "CA-MetaCOBOL+ Facilities."

6. The CA-MetaCOBOL+ Translator

This chapter describes the CA-MetaCOBOL+ Translator, its required and optional input and output files, how to control a translation, and how to run a translation.

6.1 The Translation Process

The CA-MetaCOBOL+ Translator is a COBOL source code macro processor. It translates input source programs according to translation rules specified in macros. A macro identifies an occurrence of words in the input source program that causes the Translator to act upon the instructions specified in the macro. Only those input source words that are specifically identified by a macro are processed; all other words pass directly to the COBOL output file. Thus macros determine how the source input differs from the source output.

A *macro set* is a collection of macros. Macro sets are distributed by Computer Associates or can be developed by users. Examples of macro sets are the CA-MetaCOBOL+ Structured Programming Facility SPP macro set and the CA-MetaCOBOL+ Quality Assurance macro set.

Various options exist to customize the translation process. For example, the FORMAT= translate-time option specifies the format that the CA-MetaCOBOL+ Translator uses to format the source output. These are described in Section 6.2.

CA distributed macro sets may have prerequisite Translator Options. More than one macro set may be specified for a translation, but restrictions for CA distributed macro sets do exist. In addition, if more than one macro set is loaded in a single translation, they may have to be specified in a certain order. Refer to the appropriate CA-MetaCOBOL+ documentation for restrictions and requirements pertaining to CA distributed macro sets.

For more information on the CA-MetaCOBOL+ Macro Facility, refer to the CA-MetaCOBOL+ *Macro Facility Reference* or to the *Macro Facility Tutorial*.

Translator Options, macros, and input source records are the primary input to the CA-MetaCOBOL+ Translator. Various output listings, including the source output file, are produced. Figure 1 shows the input and output of the CA-MetaCOBOL+ Translator.

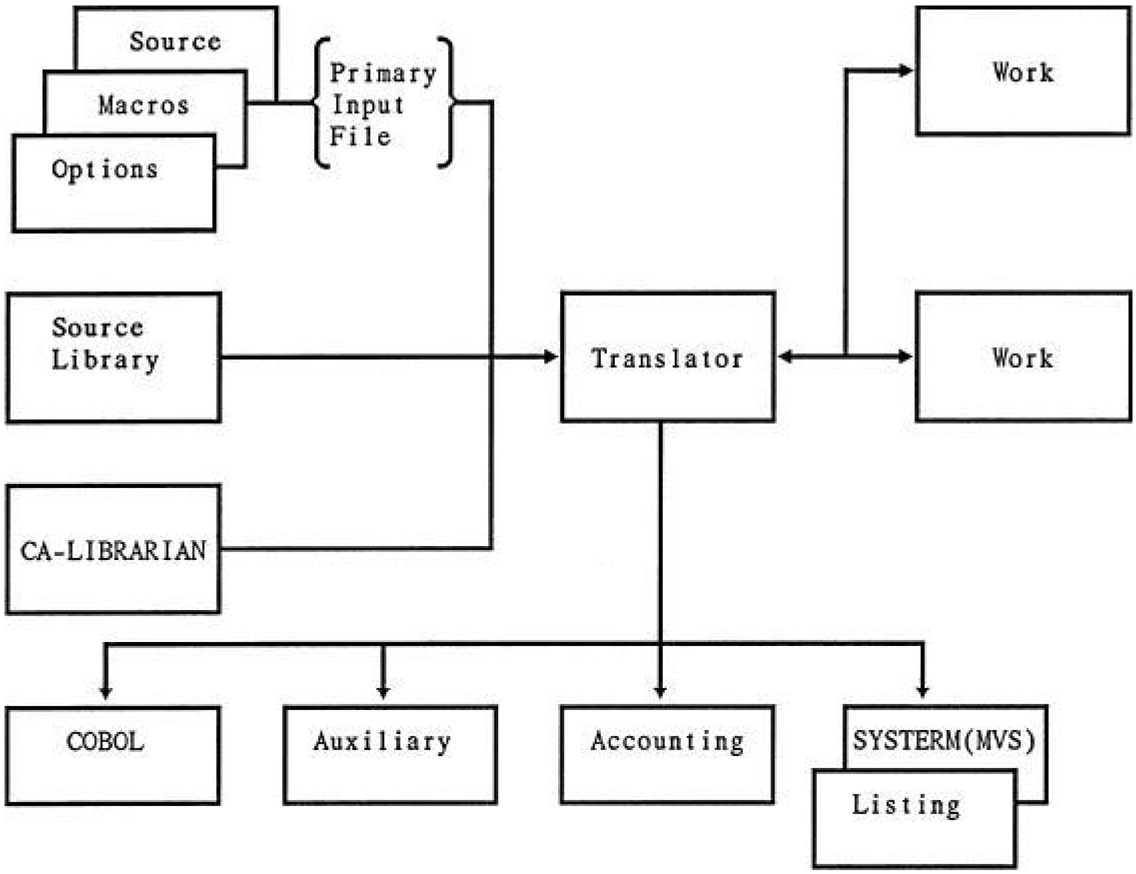

Figure 2. CA-MetaCOBOL+ Translator Input Output System Flow

6.1.1 CA-MetaCOBOL+ Translator Input

The primary input to the CA-MetaCOBOL+ Translator consists of:

- Translator Options
- Macros
- Source program

The source program and macros may be retrieved from a standard source library or from a CA LIBRARIAN or CA PANVALET master file.

CA-MetaCOBOL+ also provides the Input Exit Facility to allow the user to:

- Pass input to the CA-MetaCOBOL+ Translator from other than the normal sources, for example, when input is not from a standard library or from a CA LIBRARIAN or CA PANVALET master file.

- Process data that is not "COBOL-like" and therefore not acceptable to CA-MetaCOBOL+. For example, you can use an input exit to retrieve information from a DATADICTIONARY.

During translation, an input exit program functions as a subprogram of the CA-MetaCOBOL+ Translator. The IXIT= translate-time option identifies the desired input exit program. For more information on the Input Exit Facility and the specialized tasks it performs, refer to the CA-MetaCOBOL+ *Macro Facility Reference*.

Primary Input File

The Primary Input File contains four logical record groups, which must be arranged in the following sequence:

- CA-MetaCOBOL+ Translator Options (optional)

- CA-MetaCOBOL+ Translator Directives (optional)

- CA-MetaCOBOL+ macros (optional)

- COBOL source program

Translator Options and certain Translator Directing statements control the CA-MetaCOBOL+ translation. These are discussed in Section 6.2.

Macros specify rules for translating input source into COBOL output. They must precede the source program in the primary input file. Macros are not required; a source program input to the CA-MetaCOBOL+ Translator without macros is not translated, but is reformatted and may be re-sequenced.

The source program can contain COBOL source or CA-MetaCOBOL+ source, or both. CA-MetaCOBOL+ source statements are translated into standard COBOL by the CA-MetaCOBOL+ Translator. The first COBOL division header of the source program initiates the translation process. The source program must conform to standard COBOL formatting conventions: sequence number in columns 1-6, continuation in column 7, Area A in columns 8-11, Area B in columns 12-72, and identifying information in columns 73-80. The output produced by CA-MetaCOBOL+ also conforms to this convention.

Source Library Input File

COBOL source libraries may contain CA-MetaCOBOL+ macros and COBOL source. Macros or source are included in the input with the *$COPY Translator Directing statement, which is specified in the Primary Input File. The COBOL COPY statement is supported and may be controlled with a translate-time option.

For MVS systems, CA-MetaCOBOL+ supports the ANSI 1968 and 1974 COPY statements and the nested COPY feature of VS COBOL II. However, the VS COBOL II nested COPY feature requires the translate-time option DIALECT=XO or DIALECT=YO.

The CA LIBRARIAN and CA PANVALET Input Files

CA LIBRARIAN or CA PANVALET input files residing on either tape or disk can contain modules of both CA-MetaCOBOL+ macros and COBOL source. They are included in the input with a *$LIBET or *$LIBED Translator Directing statement within the Primary Input File.

When retrieving data from CA LIBRARIAN, the *$LIBED and *$LIBET Translator Directing statements permit the user to specify a password and the archival level of the desired module. *$LIBED also allows you to specify the beginning and ending sequence numbers to retrieve part of a CA LIBRARIAN source module.

Input and Output Files for Pre-Compiled Macros

Pre-compiled macros are macro sets that are stored and retrieved in their "parsed" form. You must unload pre-compiled macro sets to an external data set and then reload them in subsequent translations. Using pre-compiled macro sets makes the CA-MetaCOBOL+ translation more efficient and thus faster because pre-compiled macros are not parsed when they are reloaded.

To unload macros to a data set, specify the UNLOAD translate-time option and include the UNLOAD data set definition in the translation JCL. To reload macros from the UNLOAD data set, specify the RELOAD translate-time option and include the RELOAD data set definition in the translation JCL. The UNLOAD and RELOAD Translator Options are described in Section 6.2. The MVS, VSE, and CMS file requirements for unloading and reloading macros are provided in Section 6.3.

Input Listing Codes

The CA-MetaCOBOL+ Input Listing prints a special character on each line at the extreme left to indicate the origin of the line content. These characters are defined as follows:

Character	Origin
blank	Primary input
C	Retrieved by a *$COPY statement
L	Retrieved by a *$LIBED statement
*	Retrieved by a COBOL COPY statement
&	Retrieved by an input exit routine
I	Retrieved by the CA LIBRARIAN -INC command

Refer to Section 6.2 for more information on CA-MetaCOBOL+ source library input and a description of the *$COPY, *$LIBED, *$LIBET, and -INC statements.

6.1.2 CA-MetaCOBOL+ Translator Output

The primary output of the CA-MetaCOBOL+ Translator is a COBOL source program. The other available types of output are the listings of the input and output source, an auxiliary 80-column output data set, and an accounting data set.

Source Output File

The Source Output File contains the processed CA-MetaCOBOL+ input. The following rules apply to the output:

- CBL statements, if present, are formatted to begin in column 2 (column 1, if 72 characters of compiler options have been coded).
- Notes and Remarks sentences and paragraphs, debugging lines, and American National Standard COBOL comment records are not reformatted.
- The contents of columns 1-6 and 73-80 are dependent upon the RESEQ=, ISEQ=, and ID= Translator Options described in Section 6.2.
- Source input beginning in Area A is output to Area A and source input beginning in Area B is output to Area B, unless the input is redirected by a macro definition.
- Non-numeric literals are continued (with a hyphen in column 7) on subsequent lines. A non-numeric literal is the only output continued in this manner.

- Each level number begins a new output COBOL record.

CA-MetaCOBOL+ Listing File

All CA-MetaCOBOL+ input and output is listed on the CA-MetaCOBOL+ Listing File. The types of listings are:

INPUT lists the macros and source code input to the CA-MetaCOBOL+ Translator. Errors and warnings, which are produced by the CA-MetaCOBOL+ Translator, and Notes and Flags, which are produced by macros, are displayed following the input record causing the diagnostic. The Input Listing can be suppressed. However, Translator Directing statements, if specified, and all generated diagnostics are always displayed.

At the beginning of the Input Listing, the options in effect during the CA-MetaCOBOL+ translation are displayed. The OPTIONS listing consists of:

- The translate-time parameters specified for the current translation by the MVS PARM field or on OPTION records.
- The Translator Options specified at installation.
- The status of all options for the current translation.

Diagnostics for invalid parameters are provided.

OUTPUT contains the translated COBOL source. Optionally, the listing includes Note and Flag diagnostics produced by the macros. This listing can be suppressed; however, generated diagnostics are always displayed.

AUXILIARY shows all macro-generated statements directed to the Auxiliary File during translation.

LOST TEXT lists text directed out-of-line to a location that does not exist in the generated program. This report sets the return code to 12. For more information on out-of-line code, refer to the CA-MetaCOBOL+ *Macro Facility Reference Manual* or the *Guide to Macro Writing*.

STATISTICS summarizes the diagnostics, translation, and merge statistics. These are defined as follows.

DIAGNOSTICS:
The total number of error, warning, note, and flag diagnostics generated. The translation Return Code is also shown.

TRANSLATION:
The sizes of the internal symbol table (TSY), macro table (TMS), data structure table (TDS), and macro variables (TVA). If the sizes of these tables exceed the amount of core storage available, CA-MetaCOBOL+ terminates with Fatal Error B06. The sizes shown represent actual sizes in bytes.

MERGE:
The number of physical blocks written to the in-line work file (MBLOCKS) during translation. The value displayed is the number of bytes required to store the out-of-line work file in memory.

Terminal File (MVS only)

If the TERM=YES installation parameter is specified during CA-MetaCOBOL+ installation, the SYSTERM data set is available to display condensed information concerning the CA-MetaCOBOL+ translation. This information includes the input and output diagnostics and the Auxiliary Listing. This file can be directed to a remote terminal or to a standard batch output device. The content of this file is controlled by the TERM= translate-time option.

Auxiliary File

An Auxiliary File can be generated under macro control as a result of a CA-MetaCOBOL+ translation. This file can contain macro-generated job control language, assembler code, parameter-records, macros, messages, or any other type of 80-column records. The records written to the Auxiliary File may also be displayed on the Auxiliary Listing.

Accounting File

Accounting records can be generated to an optional Accounting File as a result of CA-MetaCOBOL+ translation when the &ACCT macro directive is used. Accounting File records include the current date, program name, author name, and optional, user-determined text. The Account Management Review COBOL program (AMR) accumulates this data and produces reports on CA-MetaCOBOL+ usage by both programmer and type of activity.

Under MVS, the Accounting File is written to an extendable data set, which is capable of gathering such records from many executions of CA-MetaCOBOL+.

Under VSE, the file cannot be extended.

For more information on the CA-MetaCOBOL+ Account Management Review, refer to Appendix D.

Listing Heading

The following is a description of the heading for CA-MetaCOBOL+ listings.

Field Description	Content
Product Name	CA-MetaCOBOL+
Report Identification	INPUT OUTPUT AUXILIARY LOST TEXT STATISTICS
Version.Release	vv.r
Date of Run	mm/dd/yy
Time of Run	hh:mm
Page No.	PAGE 9999
PROGRAM-ID	X(8)
TITLE Statement title	X(57)

If a VS COBOL II TITLE statement having the format:

```
TITLE 'literal' [.]
```

where *literal* has a maximum length of 57 characters, is encountered in the CA-MetaCOBOL+ input and if the translate-time option DIALECT=XO or DIALECT=YO, a page eject is performed and the literal is placed in the CA-MetaCOBOL+ listing heading. The TITLE statement is passed to the Source Output File.

6.2 Translation Control

A CA-MetaCOBOL+ translation is controlled with Translator Options, Translator Directing statements, and macros.

Translator Options enable CA-MetaCOBOL+ Translator functions that remain in effect for the duration of the translation. They are specified at the beginning of the Primary Input File. Specifying the COPY=ACTIVE translate-time option, for example, means that any time a COBOL COPY statement is encountered in the source, the COPY statement is replaced by the corresponding text from the source library. Translator Options are described in Section 6.2.1.

Translator Directing statements enable CA-MetaCOBOL+ Translator functions that take effect immediately. For example, when the Translator Directing statement *$COPY *module-name* is encountered in the primary input, it causes immediate inclusion of input from a source library. Translator Directing statements are described in Section 6.2.2.

Macros determine which input source is translated and what the translated output will be. The CA-MetaCOBOL+ *Macro Facility Tutorial* explains the CA-MetaCOBOL+ macro language in a step-by-step manner. The complete explanation of all the features of the macro facility is in the CA-MetaCOBOL+ *Macro Facility Reference*. Section 6.2.3 describes rules governing CA-MetaCOBOL+ source library access.

6.2.1 Translate-time Options

CA-MetaCOBOL+'s Translator Options enable specific CA-MetaCOBOL+ Translator functions during a CA-MetaCOBOL+ translation. In MVS, the options may be specified in the PARM field of the EXEC statement or on the OPTION card(s). In VSE and CMS, the options must be specified on the OPTION card(s).

The OPTION card(s) must be the first record(s) of the Primary Input File. The OPTION card format requires that the word OPTION begin in columns 1-11, followed by one or more spaces. (Columns 1-6 may contain a sequence number.) Each option must be separated from the subsequent option by a comma, with no intervening spaces. More than one OPTION card can be specified, but no translate-time option can extend beyond column 71 in the OPTION card. Option cards may not be continued to subsequent lines. The following example illustrates valid OPTION card formats.

```
. . .
  OPTION LISTIN,APOST,DIALECT=WO,RESEQ=NUM
```

or

```
  123456 OPTION COPY=ACTIVE
```

When an option is specified for a specific translation, it overrides any corresponding default option specified during CA-MetaCOBOL+ installation. Options may be specified using the MVS PARM field or OPTION lines (cards). An option specified with an OPTION line overrides any MVS PARM specified for the option or any specification on a preceding OPTION line.

This section lists and describes the CA-MetaCOBOL+ Translator Options.

Delimiting Literals

The APOST and QUOTE options specify the delimiting character for non-numeric, double-byte character (DBCS), graphic (EGCS), and hexadecimal literals in the generated program text. The delimiting character will be the same throughout the generated output, regardless of which characters were used in the input.

Format:

$$\left\{\begin{array}{l}\texttt{APOST}\\\texttt{QUOTE}\end{array}\right\}$$

APOST
Specifies that the single quote ('), or apostrophe, will be used as the delimiting character.

QUOTE
Specifies that the double quote (") will be used as the delimiting character.

Displaying Elapsed Time

The CLOCK option displays the elapsed CPU time as each COBOL division header of the input program is processed, and enables the &CLOCK macro directive. (MVS only)

Format:

```
CLOCK
```

Specifying Case

This option determines the case used for all or part of the generated program.

Format:

$$\left\{\begin{array}{l}\texttt{CASE}\\\texttt{KEYWORDS}\\\texttt{NAMES}\end{array}\right\} = \left\{\begin{array}{l}\texttt{LOWERCASE or L}\\\texttt{UPPERCASE or U}\\\texttt{CAPITALIZED or C}\end{array}\right\}$$

CASE
Indicates that the specified case is used for the entire generated program, except for figurative constants, which are always uppercase.

KEYWORDS
Indicates that the specified case is used for COBOL keywords in the generated program.

NAMES
Indicates that the specified case is used for user-defined names in the generated program.

LOWERCASE
Generates the entire program, the COBOL keywords, or the user-defined names in lowercase characters.

UPPERCASE
Generates the entire program, the COBOL keywords, or the user-defined names in uppercase characters.

CAPITALIZED
Capitalizes the first character of every word, including the first character after an integer or hyphen, of the COBOL keywords, or of the user-defined names.

Comment Conversion

The COMMENT, XCOM, or ICOM option specifies conversion of existing lines in the COBOL program to comments.

Format:

$$\left\{\begin{array}{l}\texttt{COMMENT}\\ \texttt{XCOM}\\ \texttt{ICOM}\end{array}\right\}$$

COMMENT
Converts obsolete paragraphs in the IDENTIFICATION DIVISION (including AUTHOR, INSTALLATION, DATE-WRITTEN, DATE-COMPILED, SECURITY, and REMARKS) and NOTE statements and paragraphs in the PROCEDURE DIVISION to comments by placing an asterisk (*) in the Indicator Area (column 7). This conversion of source text occurs before macro processing, and the converted items will be typed as comments (&n'T EQ 'N') in symbolic operands when acquired by the &GET macro directive. This option is the default.

XCOM
Suppresses the conversion to comments of obsolete paragraphs in the IDENTIFICATION DIVISION and of NOTE statements and paragraphs in the PROCEDURE DIVISION. These items will be typed as comments (&n'T EQ 'N') in symbolic operands when acquired by the &GET macro directive.

ICOM
Suppresses the conversion to comments of obsolete paragraphs in the IDENTIFICATION DIVISION and of NOTE statements and paragraphs in the PROCEDURE DIVISION. However, unlike the XCOM option, with this option, if these items are present in the source text, they will match macros. If these items include apostrophe (') or quote (") characters, the Translator will attempt to interpret them as nonnumeric literals.

Controlling COPY Processing

The COPY option controls the processing of COBOL COPY statements in the input program.

Format:

```
         ┌ IGNORE  ┐
COPY =  <  ACTIVE   >
         └ PASSIVE ┘
```

IGNORE

Suppresses processing of COBOL COPY statements. The COPY statements will appear in the generated program as they appeared in the input, unless they are removed by macro processing.

ACTIVE

Replaces COBOL COPY statements with text from the COBOL COPY library (SYSLIB in MVS, SYSLB in VSE, or the library name specified in the COPY statement. COBOL COPY statements will not appear in the generated program, and will not be available for macro processing.

PASSIVE

Suppresses processing of COBOL COPY statements. COBOL COPY statements will appear in the generated program, unless they are removed by macro processing. However, when macro directives requiring data item attributes are used (that is, when symbolic operands of the form "&n'a" appear in the macro code), the copy text is retrieved to complete the Data Attribute Table (TDS). This only occurs in the DATA DIVISION; the text retrieved will not appear in the generated program. The COPY statement is available for macro processing after the retrieval is completed.

Note: The 1968, 1974, and 1985 standards for the COPY statement are all supported.

Setting the Size of the COPY REPLACING Table

The CRSTACK option controls the size of the table used during translation for the COPY REPLACING statement. The table contains the arguments and results from the REPLACING clauses of COPY statements.

Format:

CRSTACK=*n*

n

Is the size in bytes of the COPY REPLACING table. To determine the required size, use the following formula:

s + (*n* x 7) + 68

where *s* is the sum of the lengths of the individual words (including the individual words from pseudo-text segments), and *n* is the total number of words.

Identifying the DATADICTIONARY Database

The DDID option identifies the DATADICTIONARY database ID to the CA-MetaCOBOL+ DATADICTIONARY Interface. This option is used only with the DATACOM/DB Facility.

Format:

DDID=*nnn*

nnn

Is the 1-3 digit numeric literal that is the database ID of the DATADICTIONARY database to be accessed. The value of *nnn* must be greater than or equal to zero and less than or equal to 999. The default is 000, which indicates that the CA DATADICTIONARY database ID should be retrieved from the System Resource table (DDSRTLM).

Controlling Generation of a Source Program

The DECK and NODECK options determine whether or not a source program is to be generated as output.

Format:

$$\left\{ \begin{array}{l} \texttt{DECK} \\ \texttt{NODECK} \end{array} \right\}$$

DECK

Produces a generated source program as output. This is the default.

NODECK

Suppresses generation of the source program as output.

Specifying Lines per Page

The DEPTH option determines the number of lines on each report page.

Format:

`DEPTH=`*nn*

nn

Is an integer specifying the number of lines per printed page. For example, DEPTH=55 specifies 55 lines per page. 66 lines per page is the default.

Note: Of the specified number of lines, 10 lines are reserved for top and bottom margins.

Identifying Input Text

The DIALECT option determines the input source text words that will be recognized as verbs, special registers, figurative constants, and keywords.

Format:

$$\texttt{DIALECT =}\left\{\begin{matrix}\texttt{2}\\\texttt{3}\\\texttt{4}\\\texttt{V}\\\texttt{W}\\\texttt{X}\\\texttt{Y}\end{matrix}\right\}\left\{\begin{matrix}\texttt{O}\\\texttt{D}\end{matrix}\right\}$$

The first character specifies the version of the COBOL compiler:

2 OS or DOS ANSI 1968 Version 1 or 2
3 OS or DOS ANSI 1968 Version 3
4 OS ANSI 1968 Version 4
V OS/VS ANSI 1968 (LANGLVL(1)) or DOS/VS
W OS/VS ANSI 1974 (LANGLVL(2)) or DOS/VS (default)
X VS COBOL II Release 1 or 2
Y VS COBOL II Release 3

The second character specifies the target operating system.

O OS (MVS)
D DOS (VSE)

Selecting Comments for Replication

The ENABLE option determines which Enable Comment Translator Directing statements are to be replicated.

Format:

```
ENABLE[=n...]
```

n

Is an integer value, and is optional. The value can be either a single digit or a series of digits, with each digit having a value between 0 and 7, inclusive. The value specified for *n* causes all corresponding *$***n*** Enable Comment Translator Directing statements to be replicated. For example, ENABLE by itself replicates the Enable Comment Translator Directing statements beginning with *$. ENABLE=46 replicates those statements beginning with *$4 and *$6.

If the comment text is not removed by macro processing, the comment text and the original comment will appear in the output.

Formatting the Translated Output

The FORMAT option controls the format of the generated output. For information on the FORMAT= parameters, refer to Appendix C.

Format:

```
         ⎧ ADR                ⎫
         ⎪ OPEN               ⎪
FORMAT = ⎨ CLOSED             ⎬
         ⎪ name               ⎪
         ⎩ code[,code]...)    ⎭
```

Controlling the Output ID

The ID option controls the content of columns 73 through 80 in the generated program.

Format:

```
       ⎧ name   ⎫
       ⎪ *ID    ⎪
ID =   ⎨ *SEQ   ⎬
       ⎪ *MC    ⎪
       ⎩ *BLANK ⎭
```

name

Is a 1-8 character identification that is reproduced in columns 73-80 of the output.

***ID**

Specifies that the contents of columns 73-80 of the first input source record will appear in columns 73-80 of each record in the generated program.

***SEQ**

Specifies that the contents of the input record sequence field will be moved to columns 75-80 of each record in the generated program. Columns 73-74 contain 00.

If RESEQ= is also specified, each record is renumbered, and the new sequence numbers are placed in columns 1-6 and 75-80.

***MC**

Specifies that the input line number will be placed in columns 73-80 of the output to allow tracing of compiler diagnostics back to CA-MetaCOBOL+ input.

***SAVE**

Specifies that the original contents of columns 73-80 from each input record will be retained in the generated program. However, macro processing and formatting requirements may cause values to be duplicated or lost.

***BLANK**

Specifies that columns 73-80 will be spaces.

Note: If the ID option is not specified, the contents of columns 73-80 of the output source records will be spaces.

Ignoring the Continuation Character

The IGNORE option causes the translator to ignore hyphens (-) in the Indicator Area (column 7) of the input source text. Using the IGNORE option will probably cause translation errors that produce unpredictable results for the interpretation of continued, non-numeric literals.

Format:

```
IGNORE
```

Note: Use IGNORE to suppress H02 diagnostics (QUOTE LITERAL IMPROPERLY CONTINUED) if hyphens have been omitted in column 7 to continue non-numeric literals. CA-MetaCOBOL+ continues non-numeric literals correctly when reformatting, and IGNORE enables the reformatting process to take place because it suppresses the H02 diagnostic.

Including Library Text

The -INC option is used to replace CA LIBRARIAN -INC statements or CA PANVALET ++INCLUDE statements embedded in text retrieved with a *$LIBED Translator Directing statement by the module text from the CA LIBRARIAN or CA PANVALET library. The -INC or ++INCLUDE statements will not appear in the generated program, and will not be available for macro processing. Replacement occurs in macro text as well as in COBOL source program text.

Format:

```
-INC
```

When this option is not specified, -INC or ++INCLUDE statements are not replaced by library text. The -INC or ++INCLUDE statement will appear in the generated program unless removed by macro processing. If the macro set requires data item attributes, the -INC or ++INCLUDE module text is retrieved by the Translator to complete the Data Attribute Table (TDS). This only occurs in the DATA DIVISION of a COBOL source program, and the retrieved text will not appear in the generated program. The -INC or ++INCLUDE statement is available for macro processing as a CA-MetaCOBOL+ 'Note' after the retrieval is completed.

Specifying the Decimal Symbol

The INVDEC option determines whether the decimal symbol is a period or a comma.

Format:

```
INVDEC
```

If the INVDEC option is specified, CA-MetaCOBOL+ recognizes a comma (,) as the decimal character. For example, when INVDEC is specified, CA-MetaCOBOL+ recognizes the numeric literal 9999,99 but it does not recognize 9999.99. If the INVDEC option is not specified, the decimal symbol is assumed to be a period (.), unless the clause DECIMAL-POINT IS COMMA appears in the source program.

Locating the Sequence Number Field

The ISEQ and NOISEQ options define the location of the sequence number field in the input source, so it can be transferred to the generated program.

Format:

$$\left\{\begin{array}{l}\texttt{ISEQ}\\\texttt{NOISEQ}\end{array}\right\}$$

ISEQ Specifies that the contents of columns 75 through 80 of the source program text are to be transferred to columns 1 through 6 (the Sequence Number Area) of the generated program, replacing what was there originally.

NOISEQ Specifies that the contents of columns 75 through 80 of the source program text are not to be transferred to columns 1 through 6. This option need be specified only if ISEQ was specified during installation.

Specifying the Input Exit Program

The IXIT option specifies the name of the input exit program.

Format:

`IXIT=`*program-name*

program-name
Is the 1- to 8-character program name.

Note: Macros distributed by Computer Associates require IXIT=ADRXIXIT.

Specifying the Size of the CA PANVALET Master

The LCORE option specifies the block size of the CA PANVALET master. This option is used for CA PANVALET only.

Format:

`LCORE=`*n*

Listing CA-MetaCOBOL+ Input and Output

The LISTALL option provides a complete listing of all CA-MetaCOBOL+ input and output.

Format:

```
LISTALL
```

Always specify this option when unloading macros with the UNLOAD option to obtain a complete listing of the unloaded macros.

This option preempts the NOLISTIN and NOLISTOUT options, and the list off (*$NOLIST) Translator Directing statement.

Listing CA-MetaCOBOL+ Input

The LISTIN and NOLISTIN options control the printing of the Input Listing.

Format:

$$\left\{\begin{array}{l}\texttt{LISTIN}\\\texttt{NOLISTIN}\end{array}\right\}$$

LISTIN
Prints the Input Listing (default). The List Off (*$NOLIST) Translator Directing statement can override this option.

NOLISTIN
Suppresses the Input Listing; however, diagnostic messages will always be printed. The List On (*$LIST) Translator Directing statement can override this option.

Note: If the specified option is overridden with a Translator Directing statement during macro loading, the option is re-established when the first COBOL division header is encountered.

Listing CA-MetaCOBOL+ Output

The LISTOUT and NOLISTOUT options control the printing of the Output Listing.

Format:

$$\left\{\begin{array}{l}\texttt{LISTOUT}\\\texttt{NOLISTOUT}\end{array}\right\}$$

LISTOUT
Prints the Output Listing.

NOLISTOUT
Suppresses the Output Listing (default). This option does not affect the Lost Text, Auxiliary, and Statistics reports or the NOTE and FLAG macro directive messages sent to the Output Listing, all of which are always printed.

Placing the Input Sequence Field

The LSEQ option causes the input sequence field to appear to the right of the translated source in the Output Listing.

Format:

```
LSEQ
```

Note: Sequence numbers may be missing or duplicated due to macro processing.

Printing NOTES and FLAGS

The NOTE option determines where &NOTE and &FLAG macro directive messages are printed.

Format:

```
NOTE=[I] [O]
```

I Prints &NOTE and &FLAG macro directive messages in the Input Listing (default).

O Prints &NOTE and &FLAG macro directive messages in the Output Listing.

IO Prints &NOTE and &FLAG macro directive messages in both the Input Listing and the Output Listing.

Controlling Processing of SQL and CICS Statements

The PRESERVE and NOPRESERVE options control the processing of SQL and CICS statements embedded in the source program.

Format:

$$\left\{\begin{array}{l} \texttt{PRESERVE = } \left\{\begin{array}{l}\texttt{SQL}\\ \texttt{CICS}\end{array}\right\} \\ \\ \texttt{NOPRESERVE} \end{array}\right\}$$

PRESERVE

Permits a COBOL program with embedded SQL and CICS statements to be processed without the embedded statements being affected.

SQL

Specifies that SQL statements are present and should not be affected by the translation process.

CICS
: Specifies that CICS statements are present and should not be affected by the translation process.

NOPRESERVE
: Resets the PRESERVE option. SQL and CICS statements are then available for macro processing by CA-MetaCOBOL+.

Notes:

- When both CICS and SQL statements are present in the input source program, specify both PRESERVE options in any order. For example:

  OPTION PRESERVE=SQL,PRESERVE=CICS

- If both CICS and SQL were specified in the PRESERVE option at installation, use NOPRESERVE followed by PRESERVE to restore only one of the options:

  OPTION NOPRESERVE,PRESERVE=CICS

Specifying Program Status

The PSTAT option specifies program status for CA DATACOM/DB Facility programs with a DATADICTIONARY interface. The PSTAT option also specifies the disposition of a program entity-occurrence to DATADICTIONARY.

Format:

$$\texttt{PSTAT =} \left\{ \begin{matrix} \texttt{TEST} \\ \texttt{PROD} \end{matrix} \right\}$$

TEST
: Specifies that the program is in TEST status in DATADICTIONARY.

PROD
: Specifies that the program is in PROD (production) status in DATADICTIONARY.

Specifying the Program Version

The PVER option specifies the program version for CA DATACOM/DB Facility programs with a DATADICTIONARY interface.

Format:

`PVER=`*nnn*

nnn Is the version number of the program entity-occurrence to DATADICTIONARY. The default is 000.

Using Pre-Compiled Macros

The RELOAD option specifies that pre-compiled macros are used in the translation. The JCL RELOAD DD statement determines which pre-compiled macros are retrieved.

Format:

```
RELOAD = 'xxx'
```

Note: The pre-compiled macros are not listed on the Input Listing. However, the CA-MetaCOBOL+ line number reflects the lines of macro code that have been loaded.

Additional macros may be loaded after the pre-compiled macros, and they will be printed on the Input Listing.

Refer to Section 6.3 for the MVS, VSE, and CMS requirements for loading pre-compiled macros.

Processing REPLACE Statements

The REPLACE option determines whether or not the VS COBOL II REPLACE compiler directing statement will be processed.

Format:

$$\texttt{REPLACE} \left\{ \begin{array}{l} \texttt{ACTIVE} \\ \texttt{IGNORE} \end{array} \right\}$$

ACTIVE
Implements the REPLACE statement.

IGNORE
Suppresses the processing of the REPLACE statement.

Controlling Sequence Numbering

The RESEQ option determines how the generated program is sequence numbered in columns 1 through 6 of COBOL statements. Specifying RESEQ=*n* or RESEQ=NUM will avoid sequencing errors due to macro processing or reformatting. For example, if the input records have consecutive sequence numbers, duplicate sequence numbers will result if lines are added due to reformatting.

Format:

```
         ⎧0  ⎫
RESEQ =  ⎨n  ⎬
         ⎩NUM⎭
```

0 Does not resequence the input source. However, this may result in duplicate or out of sequence numbers because of macro processing or reformatting.

n Is the 1 to 3 digit integer value, which is right-justified with zeros and assigned to the first COBOL source record. Each succeeding record sequence number is incremented by *n*. If *n* is specified at installation, it is the default value.

NUM Retains the input sequence numbers but renumbers the records when additional lines are inserted.

Separating Subscripts from Data Names

The SEPPAR and NOSEPPAR options control the separation of subscripts from data names.

Format:

```
⎧SEPPAR  ⎫
⎩NOSEPPAR⎭
```

SEPPAR

Adds a space between contiguous data-names and subscripts in the PROCEDURE DIVISION. For example, PLANT(5) becomes PLANT (5).

NOSEPPAR

Suppresses the insertion of a space between contiguous data-names and subscripts. This option need only be specified if the SEPPAR option was specified during installation.

Note: If the source program contains subscripted data-names without a space separating the data-name from the subscript, and any of the resulting character strings exceed the 30-character maximum, a Translator diagnostic may occur. To prevent the occurrence of the diagnostic, you must use the SEPPAR option when the source program contains subscripted data-names without a space separating the data-name from the subscript.

Initializing the &VSOURCE Variable

The SOURCE option initializes the reserved variable &VSOURCE to a specified character. This character can then be inserted in a macro call routine. Refer to the *Macro Reference Guide* for more information.

Format:

```
SOURCE=c
```

c is any alphabetic or numeric character. The default value for the &VSOURCE Reserved Variable is a blank.

Suppressing the Printing of COPY Text

The SUPPRESS option suppresses the printing of COPY Library text on the Input Listing when the SUPPRESS clause is coded in the COPY statement and the COPY option is set to PASSIVE.

Format:

```
SUPPRESS
```

Concatenating the Primary Input File

The SYSIPT option concatenates the SYSIPT file after the SYSRDR file, so the two files can be treated as one file, the Primary Input file. This option is valid only in VSE, and is necessary only if the SYSIPT installation option has not been specified.

Format:

```
SYSIPT
```

Controlling the Term Report

The TERM option determines what information will be printed in the CA-MetaCOBOL+ Term Report. This option is valid only in MVS, and can only be specified if the TERM installation option was set to YES.

Format:

```
TERM=[{parameter,...}]
```

When TERM is specified with no additional parameters, the basic Term Report, including the Options in Effect, Statistics, Fatal Errors, and Return Code, is printed. The lines required to print this basic report are not included in the line limits specified in the optional LIMIT and CANCEL parameters. The other optional parameters determine additional report content.

Valid optional parameters for this option are:

DIAGIN[=*n*] Specifies that diagnostic messages from the Input Listing (NOTE=I) are to be included in the Term Report. The value of *n* must be an integer that indicates the number of messages to be printed. If *n* is not specified, all Input Listing diagnostic messages are printed. When the LISTIN option is specified, the line preceding each diagnostic message is also printed.

DIAGOUT[=*n*] Specifies that diagnostics from the Output Listing (NOTE=O) are to be included in the Term Report. The value of *n* must be an integer that indicates the number of diagnostic messages to be printed. If *n* is not specified, all Output Listing diagnostic messages are printed. When the LISTOUT option is specified, the line preceding each diagnostic message is also printed.

LOST[=*n*] Specifies that the Lost Text Report is to be included in the Term Report. The value of *n* must be an integer that indicates the number of Lost Text Report lines to be printed. If *n* is not specified, all Lost Text Report lines are printed.

AUX[=*n*] Specifies that the Auxiliary Report is to be included in the Term Report. The value of *n* must be an integer that indicates the number of Auxiliary Report lines to be printed. If *n* is not specified, all Auxiliary Report lines are printed.

LIMIT=*n* Specifies a line limit for the Term Report. The value of *n* must be an integer that indicates the total number of lines to be printed in the Term Report. If the LIMIT parameter is specified, *n* is required. Do not specify the LIMIT parameter if the CANCEL parameter is specified.

CANCEL=*n* Specifies a line limit for the Term Report. The translation is cancelled when the limit is exceeded. The value of *n* must be an integer that indicates the number of lines to be printed before the job will be cancelled. If the CANCEL parameter is specified, *n* is required. Do not specify the CANCEL parameter if the LIMIT parameter is specified.

Note: The optional subparameters must be enclosed in parentheses and separated by commas. For example,

OPTION TERM=(DIAGIN,DIAOUT=5,LOST,CANCEL=50)

will print all input diagnostics, up to five output diagnostics, and all lost text, and will cancel the run when the total number of lines in addition to the lines in the basic report exceeds 50 lines.

Activating the Trace Facility

The TRACE option activates the *$TRACE Translator Directing statement, which is a CA-MetaCOBOL+ debugging feature. See the CA-MetaCOBOL+ *Macro Facility Reference Manual* for more information.

Format:

```
TRACE=[D] [G] [S] [T]
```

D Prints the sequence number of the macro directive that caused a diagnostic message immediately after the diagnostic line.

G Prints the sequence number of the &GET macro directive being executed and the word acquired by the &GET directive.

S Prints the input sequence number of the &SCAN macro directive being executed and the data-name acquired by the directive.

T Prints the input sequence number of the macro line causing transfer of control (branching).

Note: The parameters may be specified in any combination.

Storing Macros in a Data Set

The UNLOAD option stores the macros loaded for a translation into a sequential data set or a member of a partitioned data set. The UNLOAD JCL statement determines where the macros are stored.

Format:

UNLOAD='*name*'

name Is a string of 1 to 8 characters that identifies the unloaded macros.

Notes:

- Refer to the MVS, VSE, and CMS requirements for the UNLOAD translate-time option in Section 6.3.
- The macros will not appear on the Input Listing when they are reloaded. Use the LISTALL option to obtain a listing of the macros for reference purposes.

Supplying Information to Macros

The UPSI options provide a single character of information to macros that can be retrieved using the Reserved Variables &VUPSI1 through &VUPSI8.

Format:

`UPSI`*n*=*c*

n Is an integer with a value of 1 to 8.

c Is any single alphabetic or numeric character.

Note: If not specified, the value *c* defaults to blank.

Some CA supplied macro sets use the UPSI option. Refer to the appropriate documentation for a description on how to specify the UPSI option for a translation involving a specific CA-MetaCOBOL+ macro set.

Supplying a Character String to a Macro

The VAR option provides a character string with a maximum length of 30 characters to macros.

Format:

`VAR=`*word*

word Is the character string to be supplied to the macro. *Word* can have a maximum length of 30 characters. The default value for the VAR Special Register is a blank. If *word* is not specified, the value of *word* is blank by default.

Abbreviations of Translate-time Options

Below is a list of keywords and abbreviations for all CA-MetaCOBOL+ Translator Options.

Keyword	Abbreviation	Keyword	Abbreviation
-INC	-I	NAMES=	NM=
APOST	AP	NODECK	ND
CASE=	CA=	NOISEQ	NI
CLOCK	CL	NOLISTIN	XI
COMMENT	CM	NOLISTOUT	XO
COPY=	CY=	NOPRESERVE	XP
CRSTACK=	CR=	NOSEPPAR	NP
DDID=	DD=	NOTE=	NO=
DECK	DK	QUOTE	QU
DEPTH=	DP=	PRESERVE=	PR
DIALECT=	DI=	PSTAT=	PS=
ENABLE=	EN=	PVER=	PV=
FORMAT=	FO=	RELOAD	RL
ICOM	IC	REPLACE	RP
ID=	ID=	RESEQ=	RE=
IGNORE	IG	SEPPAR	SE
INVDEC	IN	SOURCE=	SO=
ISEQ	IS	SUPPRESS	SU
IXIT=	IX=	SYSIPT	SI
KEYWORDS=	KY=	TERM=	TE=
LCORE=	LC=	TRACE=	TR=
LISTIN	LN	UNLOAD=	UL=
LISTOUT	LI	UPSI*n*=	Un=
LISTALL	LA	VAR=	VA=
LSEQ	LS	XCOM	XC

6.2.2 Translator Directing Statements

Translator Directing statements cause the CA-MetaCOBOL+ Translator to perform a task immediately. For example, several Translator Directing statements enable the retrieval of macro definitions or COBOL source. Other Translator Directing statements suppress or activate subsequent functions that are in effect when the input stream is processed.

Translator Directing statements are embedded in the input stream as special COBOL comment records; that is, they are input records that contain a asterisk (*) in the continuation column (column 7) and a dollar sign ($) in column 8. Translator Directing statements must be entered before the first COBOL division header.

All Translator Directing statements can be used in macro text, but are not translated by macros. These statements can also be used in source program text; however, the Let On, Let Off, Trace On, Trace Off, and Region statements are ineffective and unsuitable in source program text.

The Copy, Include, and Call IXIT statements are only effective when encountered in the primary input (with the OPTION statements). They are considered comments when retrieved from a secondary input source (SYSLIB, SYSSLB, MASTER, MASTIN, TMAST). All the other Translator Directing statements remain functional when retrieved from a secondary source of input, including retrieval by a COBOL COPY statement.

All Translator Directing statements except the Enable Comment statement (*$[*n*]) are always printed in the Input Listing. Translator Option settings and the results of other Translator Directing statements that manipulate the Input Listing cannot suppress the printing of these commands.

The statement *$HDR begins all CA distributed procedure sets. It is used during CA-MetaCOBOL+ installation to facilitate the selection of macro sets.

Printing a Header

The *$HDR statement is used to print a descriptive header.

Format:

```
*$HDR [comment]
```

comment
Is the text that is to be printed in the Input Listing.

Note: This statement is always printed, even when the NOLISTIN Translator Option has been specified, or when this statement is preceded by a List Off (*$NOLIST) Translator Directing statement.

Controlling Printing of the Input Listing

The *$LIST and *$NOLIST Translator Directing statements activate and suppress the Input Listing:

Format:

```
*$LIST [comment]
*$NOLIST [comment]
```

***$LIST**
Activates the Input Listing and overrides the NOLISTIN Translator Option and any preceding List Off (*$NOLIST) Translator Directing statements.

***$NOLIST**
Suppresses the Input Listing and overrides the LISTIN Translator Option and any preceding List On (*$LIST) Translator Directing statements. Use *$NOLIST to suppress all or part of the macro listing.

*$NOLIST does not override the LISTALL Translator Option. The printing of Translator Directing statements and diagnostics is not affected.

comment Any text normally placed within a comment record.

Note: The first COBOL division header resets the listing as specified by the LISTIN/NOLISTIN Translator Options.

The following example demonstrates the use of a *$NOLIST statement to suppress the listing of subsequent macros. Notice that *$LIST is not required in the example below, since the Identification Division header resets listing control because the LISTIN Translator Option is specified.

```
OPTION LISTIN, . . .
      *$HDR HTMACROS 7/21/91
      *$NOLIST            SUPPRESS LISTING OF MACROS
      (macros)
       IDENTIFICATION DIVISION.  . . .
```

Notice that the *$HDR statement is included to print a header for the macro set, regardless of how the LISTIN or NOLISTIN option is set.

Copying Modules from a Standard Library

The *$COPY Translator Directing statement retrieves macros and source from the SYSLIB (MVS) and SYSSLB (VSE) source libraries, and returns the source library member's content in the place of the original Copy statement.

Format:

```
     *$COPY member [comment]
```

member Is the name of the SYSSLB or SYSLIB member to be retrieved. The member may contain macros, macros and COBOL source, or just COBOL source text. COBOL COPY statements embedded in the retrieved member will be processed according to the COPY Translator Option.

comment Can be any text normally placed within a comment record.

The following example demonstrates the use of a *$COPY statement to include the macro set module SPP from a source statement library.

```
OPTION
 . . .
      *$COPY SPP  STRUCTURED PROGRAMMING PROCEDURE
       IDENTIFICATION DIVISION.
 . . .
```

Copying Modules from Disk or Tape

The *$LIBED and *$LIBET Translator Directing statements retrieve macros and source text from disk and tape, respectively. These statements must be primary input statements.

Format:

```
{*LIBED}
{*LIBET} member[,pppp][([seq1][,seq2])] [,ARC=-n] [comment]
```

***$LIBED**
Retrieves one member from the CA LIBRARIAN MASTER or CA PANVALET library, and returns the member's content in the place of the original Include statement.

***$LIBET**
Retrieves one member from the CA LIBRARIAN MASTIN (MVS) or TMAST (VSE) tape data set, and returns the member's content in the place of the original Copy statement.

member
Is the name of the module to be retrieved. The member may consist of macros, macros and COBOL source, or just COBOL source text. COBOL COPY statements embedded in the COBOL source text will be processed according to the COPY Translator Option setting. Input retrieved with *$LIBED cannot contain a *$COPY, *$LIBED, or *$LIBET statement. JCL statements at the beginning of the member are not retrieved.

Optional Parameters for CA LIBRARIAN Only

The following optional parameters apply only with CA LIBRARIAN.

pppp Is a 4-character password for the member. Proper specification of the password is required when the CA-MetaCOBOL+ Translator is installed with password protection for CA LIBRARIAN; otherwise, it is ignored.

seq1 Is the first CA LIBRARIAN sequence numbers to be retrieved. If *seq1* is not specified, retrieval begins with the first record. If neither *seq1* nor *seq2* is specified, the entire member is retrieved.

seq2 Is the last sequence number to be retrieved. If *seq2* is not specified, retrieval continues to the end of the member. If neither *seq1* nor *seq2* is specified, the entire member is retrieved.

ARC=-*n*
Specifies the member's relative (-*n*) archival level. When omitted, the most current version is retrieved.

comment
Any text normally placed within a comment record.

Note: If a module contains CA LIBRARIAN -INC or CA PANVALET ++INCLUDE statements, the -INC Translator Option must be specified. If -INC is not specified, these statements are treated as comments. The -INC command is only valid when retrieved with *$LIBED. Expansion of nested ++INCLUDE statements is supported to the nesting level that CA PANVALET supports.

Enabling Comments

The *$*n* Translator Directing statement permits a replica of the text portion of selected comments to become macro or COBOL source text. Both the COBOL compiler and the CA-MetaCOBOL+ Translator handle these statements as comments, unless CA-MetaCOBOL+ is instructed to treat them as source records.

Format:

*$[*n*] *text*

n Is an integer from 0 through 7, and is optional. Each integer corresponds to a parameter of the ENABLE Translator Option, which can permit up to eight different sets of enabled comments. The ENABLE option, with no selection criteria specified, will enable all Enable Comment statements without an integer identifier. ENABLE=0 will enable statements both without an integer identifier and with an identifier of 0 (*$0). The ENABLE option permits selecting multiple sets of Enable Comments by specifying more than one selection criteria. For example, ENABLE=36 will select both *$3 and *$6 statements.

When the ENABLE Translator Option is specified, the comment record is treated as both a comment and a source statement. Otherwise, it is treated as a comment record. If the comment text is not removed by macro processing, both the enabled comment and the translated source appear in the generated COBOL output.

text Is the comment text, and must be separated from *n* by at least one space. *Text* is treated as CA-MetaCOBOL+ input if the comment record in which they are contained is enabled.

The following example demonstrates the use of enabled comments.

Input:

```
OPTION ENABLE=2, . . .
      *$LIBED MACROS          LOAD 'MACROS'
       . . .
        PROCEDURE DIVISION.
       . . .
      *$1  IF MONDAY GO TO MONDAY-ROUTINE.
      *$2  IF TUESDAY GO TO TUESDAY-ROUTINE.
      *$3  IF WEDNESDAY GO TO WEDNESDAY-ROUTINE.
       . . .
```

Output:

```
 PROCEDURE DIVISION.
*$1  IF MONDAY GO TO MONDAY-ROUTINE.
*$2  IF TUESDAY GO TO TUESDAY-ROUTINE.
     IF TUESDAY
         GO TO TUESDAY-ROUTINE.
*$3  IF WEDNESDAY GO TO WEDNESDAY-ROUTINE.
```

Regionalized Macro Sets

The *$REGION Translator Directing statement permits the same global variable and label names to be used in different macro regions. Up to ten macro regions are supported. The *$REGION statement separates the macros into sets, each set in its own region. The initial region is implied. Subsequent regions (up to 9) must be specified by entering the *$REGION Translator Directing statement to separate the macros.

Format:

`*$REGION [`*comment*`]`

***$REGION** This statement is recognized within the Primary Input File and within input text retrieved by a *$CALL, *$COPY, *$LIBED, or *$LIBET Translator Directing statement. Global variable and label names preceding the *$REGION statement may only be referenced by macro code also preceding the statement. The same names may then be used following the statement to define different variables and labels, but they may only be referenced by macro code following the statement (and before any other *$REGION statement).

comment Can be any text normally placed within a comment record.

Note: A *$REGION statement encountered following the first division header is treated as a comment record.

In the following example, &V@PGM is used in two regions.

```
OPTION . . .
      SI   PROGRAM-ID &1.
           &GLOBAL &V@PGM X(8)
           &SET &V@PGM = &1
           &GO &L@EDIT-NAME
         . . .
       &L@EDIT-NAME
         . . .
      *$REGION
      SP   CALL &1
           &GLOBAL &V@PGM X(30)
           &SET &V@PGM=&1
           &GO &L@EDIT-NAME
         . . .
       &L@EDIT-NAME
         . . .
```

Suppressing and Restoring Translation

The MCT Off (*$NOMCT) and MCT On (*$MCT) Translator Directing statements suppress and restore macro translation.

Format:

```
*$NOMCT        [comment]
*$MCT          [comment]
```

***$NOMCT**

Suspends translator processing. Recognition of other Translator Directing statements is suspended, with one exception; Free Form processing continues. (See the Free Form Translator Directing statements Column On (*$COL) and Column Off (*$NOCOL)). Macro processing of input is suspended. Formatting of output text is suspended. COBOL COPY statement processing is suspended. Any macro text input after this statement is ignored. Any COBOL source text input after this statement will appear in the generated program unchanged from the form in which it was input. Input continues to be printed in the Input Listing.

***$MCT**

Restores translator processing following an MCT Off (*$NOMCT) statement. Recognition of other Translator Directing statements resumes. Macro processing of input resumes. Formatting of output text resumes. COBOL COPY statement processing is restored. Any macro text entered after this statement is processed. Any COBOL source text entered after this statement will appear in the generated program as processed by macros, formatted according to FORMAT option specifications.

comment Can be any text normally placed within a comment record.

Notes: *$NOMCT and *$MCT can be used to pass input through the Translator without manipulating the input in any way. Selected macros within a macro set can be isolated and prevented from loading. COBOL statements, or statements intended for another COBOL compiler pre-processor, can be isolated and prevented from being processed by CA-MetaCOBOL+.

*$MCT and *$NOMCT are recognized only if encountered within the Primary Input File or within input text that is directly subordinate to *$CALL, *$COPY, *$LIBED, or *$LIBET.

In the following example, the macro-level CICS statement is not translated.

```
 PROCEDURE DIVISION.
*$NOMCT
 DFH16 PARMO,PARMP
*$MCT
```

Free Format Input

Free form input, which is delimited with the *$COL and *$NOCOL Translator Directing statements, makes it easier to input code from a terminal.

Format:

```
 *$COL[n]   [comment]
7*$NOCOL    [comment]
```

***$COL**

Initiates free form input, and must begin in column 7. Beginning with the line following this statement, coding can begin in column 1 and continue for the length specified by *n*. Column 1 is assumed to be column 12 (Area B). Code an 8 in column 1 to indicate that a line's content should be understood to begin in column 8 (Area A) rather than column 12. Code a 7 in column 1 to indicate that a line's content should be understood to begin in column 7 (Indicator Area) rather than column 12.

n A value from 10 to 80, which specifies the last column of a record to be interpreted as input. If *n* is not specified, 80 is assumed.

***$NOCOL**

Terminates free form input, and must be coded as free form input, that is, it must begin in column 2 following a 7 in column 1. Input following this statement requires standard CA-MetaCOBOL+ and COBOL margins and sequence numbers (Area A, Area B, and the Sequence Number Area).

comment

Can be any text normally placed within a comment record.

Note: The *$COL and *$NOCOL statements do not appear in the generated output.

The following example shows free form input and the resulting output.

Input:

```
123456*$COL72   Begin free form input using 72 columns
8PROCEDURE DIVISION.
7* THIS IS A COMMENT
MOVE A TO B.
8PARAGRAPH-1.
GO TO PARAGRAPH-2.
7*$NOCOL        End free form input
```

Output:

```
 PROCEDURE DIVISION.
* THIS IS A COMMENT
     MOVE A TO B.
 PARAGRAPH-1.
     GO TO PARAGRAPH-2.
```

Controling Macro Debugging

The *$TRACE and *$NOTRACE Translator Directing statements delimit the macro code to be debugged as specified by the TRACE Translator Option. For a complete discussion of the macro debugging facilities, see the CA-MetaCOBOL+ *Macro Facility Reference Manual* or the *Guide to Macro Writing*.

Format:

```
*$TRACE      [comment]
*$NOTRACE    [comment]
```

***$TRACE**

Defines where the Debugging Trace begins when the TRACE option is specified. The macro code that follows the Trace On (*$TRACE) statement must be executed in order to be traced.

***$NOTRACE**

Defines where the Debugging Trace ends. The macro code that follows the Trace Off (*$NOTRACE) statement will not be traced.

comment

Can be any text normally placed within a comment record.

Notes: The TRACE Translator Option must be specified to implement a trace, and also determines the type of trace desired. The Trace On (*$TRACE) and Trace Off (*$NOTRACE) statements may remain in the macro code, since they are ignored when the TRACE option is not specified.

The messages produced by a trace are always printed in the Input Listing, regardless of how the LISTIN and NOLISTIN options are set.

In the following example, only NEWMAC is traced.

```
OPTION TRACE=. . .
      *$TRACE
      *$LIBED NEWMAC       LOAD NEW MACROS
      *$NOTRACE
      *$LIBED OLDMAC
       IDENTIFICATION DIVISION.  . . .
```

Suppression of Unresolved Label Diagnostics

To suspend or restore verification of macro label (&lname) references, use the Let On (*$LET) or Let Off (*$NOLET) Translator Directing statement.

Format:

```
*$LET     [comment]
*$NOLET   [comment]
```

***$LET** Suspends verification of macro label references to prevent undefined labels from producing a C33 diagnostic during macro loading. If the macro containing the undefined label is executed, however, an N01 diagnostic will be produced, and translation will be terminated.

***$NOLET** Restores macro label (&lname) reference verification. The C33 diagnostic will be produced when a label reference cannot be verified.

comment Can be any text normally placed within a comment record.

The *$LET and *$NOLET Translator Directing statements permit the implementation of macro sets that must satisfy the following requirements:

- Two or more macro sets must function independently of one another.
- These macro sets must function together in the same translation.
- The combined macro sets must share code in the same region.

The following abbreviated example illustrates the problem.

```
* macro set 1
      &GLOBAL &VT = 1 X
      &IF &VT = 2 &GO &LFRIEND
* macro set 2
      &GLOBAL &VT = 2 X
 &LFRIEND
```

The ability to have duplicate variable definitions and to use the last definition to determine the value of the variable makes it possible to use the variable &VT in macro set 1 to determine that macro set 2 is loaded. However, any attempt to execute macro set 1 independent of macro set 2 will result in a C33 diagnostic, because the label &LFRIEND is only defined in macro set 2. The Let On (*$LET) and Let Off (*$NOLET) statements can overcome this shortcoming as follows:

```
* macro set 1
      &GLOBAL &VT = 1 X
*$LET
      &IF &VT = 2 &GO &LFRIEND
*$NOLET
* macro set 2
      &GLOBAL &VT = 2 X
 &LFRIEND
```

This arrangement will permit macro set 1 to function without a C33 diagnostic when macro set 2 is not loaded.

Invoking Subprograms

The *$CALL Translator Directing statement permits retrieval of input accessed through a user-defined subprogram. This statement must be a primary input statement.

Format:

`*$CALL [`*parameter*`] [`*comment*`]`

parameter

Is a continguous character string from 1 to 56 characters long that is passed to the called subprogram. The format of the parameter is determined by user specifications.

comment

Can be any text normally placed within a comment record.

Note: The IXIT Translator option must be specified to provide the name of the subprogram to be invoked. See the CA-MetaCOBOL+ *Macro Facility Reference Manual* for information on how to prepare an IXIT subprogram.

6.2.3 Source Library Input

CA-MetaCOBOL+ supports source library module access according to the following hierarchy:

1. Within the Primary Input File, modules may be retrieved with:
    - A *$COPY Translator Directing statement
    - A *$LIBED Translator Directing statement
    - A *$LIBET Translator Directing statement
2. COBOL source text may contain COBOL COPY statements to retrieve additional source.
3. COBOL source text retrieved with *$COPY may retrieve additional source with a COBOL COPY statement.
4. COBOL source retrieved with *$LIBED may retrieve additional source with:
    - A COBOL COPY statement
    - A CA LIBRARIAN -INC command
5. COBOL source text retrieved with a *$LIBET may retrieve additional source with a COBOL COPY statement.
6. COBOL source text retrieved with -INC may retrieve additional source with:
    - A COBOL COPY statement
    - A CA LIBRARIAN -INC command
7. Macro source text retrieved with *$LIBED may retrieve additional source text with a CA LIBRARIAN -INC command.
8. COBOL source text retrieved with a COBOL COPY statement may retrieve additional source text, if DIALECT=XO, with a COBOL COPY statement.

Note: If a CA LIBRARIAN -INC command or a *$COPY, *$LIBED, or *$LIBET Translator Directing statement does not follow these rules, it is treated as a comment record.

Up to five levels of nesting are permitted for nested CA LIBRARIAN -INC commands.

The figure below illustrates the types of secondary and tertiary input that may be retrieved from primary input.

Primary Input	Secondary Input	Tertiary Input
Macro Source		
*$COPY	SYSLIB	
*$LIBED	MASTER -INC	MASTER
*$LIBET	MASTIN	
COBOL Source		
COPY	SYSLIB COPY **	SYSLIB
*$LIBED	MASTER -INC* COPY	MASTER SYSLIB COPY**
*$LIBET	MASTIN COPY	SYSLIB COPY**
*$COPY	SYSLIB COPY	SYSLIB COPY**

Figure 3. Secondary/Tertiary Input from Primary Input

*   Up to five levels of input from CA LIBRARIAN can be retrieved with -INC.
**   Only when DIALECT=XO or DIALECT=YO.

6.3 How to Run a Translation

The following sections describe the requirements to execute CA-MetaCOBOL+ in the MVS, VSE, and CMS environments.

6.3.1 Translations Under MVS

The following sections describe the requirements to execute CA-MetaCOBOL+ in the MVS environment.

MVS File Requirements

The following list contains the applicable MVS DDnames with a brief description of their use. The following subsection gives an example on how to specify the DD statements for each DDname.

ACCT Accounting File; required when accounting data is generated.

AUX Auxiliary File; required when auxiliary data is punched or written as an output data set with the &AUX and &AUXN directives.

CARDF Primary Input File; required. This file contains the input Translator Options, macros, and source program.

CYCLE CA LIBRARIAN Tape Master Cycle Control File; required when the input contains *$LIBET statements and the CA LIBRARIAN tape master is under cycle control (also see MASTIN).

FE Out-of-Line Work File; required.

FM In-Line Work File; required.

LSTIN Listing File; the printer file for the Input, Output, Statistics, Auxiliary, and Lost Text Listings.

MASTER CA LIBRARIAN or CA PANVALET Disk Master Input File; required when the input contains *$LIBED statements.

MASTIN CA LIBRARIAN Tape Master Input File; required when the input contains *$LIBET statements (also see CYCLE).

PUNCHF Generated COBOL Output File; required when the DECK option is specified.

RELOAD Pre-Compiled Macros; required when pre-compiled macros are to be retrieved. This DD statement must reference a sequential data set with a logical record length of 80 bytes that contains the desired pre-compiled macros.

SYSLIB COBOL Source Library Input; required when the input contains *$COPY or COBOL COPY statements.

Note: Additional COBOL source library files may be required if the *library-name* option of the COPY statement (OS/VS 74 standard) is used.

SYSTERM Terminal Output; required when the TERM option is specified.

UNLOAD Pre-Compiled Macros; required for storing pre-compiled macros. This DD statement must define an existing sequential data set or member of a partitioned data set that will contain the pre-compiled macros. The data set or member must have a logical record length of 80 bytes and must be large enough to contain the unloaded macros.

MVS Translation Execution JCL

The following JCL is supplied as a guide for executing CA-MetaCOBOL+ in an MVS environment. The SYSOUT class and BLKSIZE (in multiples of sizes shown), SPACE, and UNIT parameters may be altered.

```
 //[stepname] EXEC PGM=usermct[,PARM='options']
 //STEPLIB DD DSN=user.loadlib,DISP=SHR
[//        DD DSN=LIBRARIAN.loadlib,DISP=SHR]
 //LSTIN   DD SYSOUT=A[,DCB=BLKSIZE=121]
 //PUNCHF  DD SYSOUT=B[,DCB=BLKSIZE=80]
[//AUX     DD SYSOUT=B[,DCB=BLKSIZE=80]]
[//SYSTERM DD SYSOUT=A[,DCB=BLKSIZE=80]]
[//ACCT    DD DSN=user.acct,DISP=SHR]
 //FE      DD UNIT=SYSDA,
              SPACE=(TRK,(20,10))[,DCB=BLKSIZE=1480]
 //FM      DD UNIT=SYSDA,
 //           SPACE=(TRK,(20,10))[,DCB=BLKSIZE=1480]
 //UNLOAD DD DSN=user.maclib,
 //          UNIT=SYSDA,
 //          SPACE=(TRK,10),
 //          DCB=(RECFM=FB,LRECL=80,BLKSIZE=8000),
                     CATLG
[//          DISP=(,[KEEP ],DELETE)                  ]
[//RELOAD DD DSN=user.maclib,DISP=(SHR)]
[//SYSLIB DD DSN=user.srclib,DISP=SHR]
[//MASTER DD DSN=user.disklibr,DISP=SHR]
 //CYCLE  DD DSN=user.cycle,DISP=OLD
 //MASTIN DD UNIT=(TAPEx,,DEFER),
 //          VOL=SER=x,
             DCB=(DEN=x,BLKSIZE=bbbb)
[            DISP=OLD,               ]
 //CARDF DD *[,DCB=BLKSIZE=80]
  [OPTION options]
  [macros]
   source program
 /*
```

JCL to execute a CA-MetaCOBOL+ translation can be placed in a PROC like the one following. Optional files have been specified as NULLFILE dummy data sets, to be employed as needed. Blocksizes may be altered as needed to optimize DASD usage for your installation.

```
//METACBL PROC META=MCT,                        CA-MetaCOBOL+ Prog. Name
//             STEPLIB='MCT.LOAD',              Load Library Name
//             COPYLIB='MCT.SYSLIB',            COPY Library
//             LIBMSTR='MCT.DISKLIBR',          Disk CA LIBRARIAN Master
//             COBOL=NULLFILE,                  COBOL Output
//             AUX=NULLFILE,                    Auxiliary Output
//             TERM=NULLFILE,                   Terminal Data Set
//             CYCLE=NULLFILE,                  Tape CA LIBRARIAN Cycle
//             LIBTAPE=NULLFILE                 Tape CA LIBRARIAN Master
//MCT     EXEC PGM=&META
//STEPLIB   DD DSN=&STEPLIB,DISP=SHR
//LSTIN     DD SYSOUT=A,
//             DCB=(RECFM=FBA,LRECL=121,BLKSIZE=605)
//PUNCHF    DD DSN=&COBOL,
//             UNIT=SYSDA,
//             SPACE=(TRK,100),
//             DCB=BLKSIZE=7200,
//             DISP=(,PASS,DELETE)
//AUX       DD DSN=&AUX,
//             UNIT=SYSDA,
//             SPACE=(TRK,20),
//             DCB=BLKSIZE=7200,
//             DISP=(,PASS,DELETE)
//SYSLIB    DD DSN=&COPYLIB,DISP=SHR
//MASTER    DD DSN=&LIBMSTR,DISP=SHR
//SYSTERM   DD DSN=&TERM,
//             UNIT=SYSDA,
//             SPACE=(TRK,20),
//             DCB=BLKSIZE=80,
//             DISP=(,PASS,DELETE)
 //CYCLE     DD DSN=&CYCLE,DISP=SHR
 //MASTIN    DD DSN=&LIBTAPE,
 //             UNIT=(TAPE,,DEFER),
 //             VOL=SER=X,
 //             DCB=(DEN=3,BLKSIZE=32000),
 //             DISP=OLD
 //FE        DD UNIT=SYSDA,
 //             SPACE=(TRK,(20,10)),
 //             DCB=BLKSIZE=7400
 //FM        DD UNIT=SYSDA,
 //             SPACE=(TRK,(20,10)),
 //             DCB=BLKSIZE=7400
 //UNLOAD    DD DSN=user.maclib,
 //             UNIT=SYSDA,
 //             SPACE=(TRK,10),
 //             DCB=(RECFM=FB,LRECL=80,BLKSIZE=7200),
                       CATLG
[//          DISP=(,[KEEP ],DELETE)      ]
 //RELOAD    DD DSN=user.maclib,DISP=(SHR)
 //CARDF     DD DDNAME=SYSIN
```

Concatenating Source Input

To concatenate a macro set from a macro library and the input source, specify the CARDF DD statement as follows:

```
//CARDF DD DSN=user.srclib(member),DISP=SHR
//      DD *
           . . .
```

To include input from a source library, specify a SYSLIB DD statement and the *$COPY Translator Directing statement as follows:

```
//SYSLIB DD DSN=user.srclib,DISP=SHR
//CARDF  DD *
      *$COPY module-name
         . . .
```

To include input from a CA LIBRARIAN disk master, specify the MASTER DD statement and the *$LIBED Translator Directing statement as follows:

```
//MASTER DD DSN=user.disklibr,DISP=SHR
//CARDF  DD *
      *$LIBED module-name
         . . .
```

To include input from a CA LIBRARIAN tape master, specify the CYCLE and MASTIN DD statements and the *$LIBET Translator Directing statement as follows:

```
//CYCLE  DD DSN=user.cycle,DISP=OLD
//MASTIN DD UNIT=(TAPEx,,DEFER),
//          VOL=SER=volserno,
//          DCB=(DEN=x,BLKSIZE=xxxx),
//          DISP=OLD
//CARDF  DD *
      *$LIBET module-name
         . . .
```

Note: CARDF data sets may be concatenated without consideration of BLKSIZE.

Specifying Translate-time Options

Translator Options can be specified in the PARM field of the EXEC statement or in one or more OPTION cards. These options control specific functions for the current execution of the CA-MetaCOBOL+ Translator. (Refer to Section 6.2.1 for a description of the OPTION card and Translator Options.)

The following example illustrates the two methods for specifying Translator Options:

MVS PARM:

```
//sample JOB ...
//STEP EXEC proc-name,PARM='DECK,LISTIN'
         . . .
```

OPTION card:

```
//CARDF DD *
       OPTION DECK,LISTIN
       *$COPY ...
        IDENTIFICATION DIVISION.
         . . .
```

MVS Condition Codes

The CA-MetaCOBOL+ Translator passes a condition code representing the severity of the highest-level diagnostic in the step. The condition code can be tested by the JCL in subsequent job steps. Condition codes may be set in macro definitions by the &FLAG, &NOTE, or &COND macro language directives, which are described in the CA-MetaCOBOL+ *Macro Facility Reference*.

The following are the default condition codes. Additional codes may be set by macro definitions.

Code	Meaning
0	No diagnostic generated.
4	The highest-level diagnostic is caused by a &FLAG (N98) or &NOTE (N99) macro language directive.
8	The highest-level diagnostic is a CA-MetaCOBOL+ Translator WARNING.
12	The highest-level diagnostic is a CA-MetaCOBOL+ Translator ERROR or LOST TEXT source.
16	The highest-level diagnostic is a CA-MetaCOBOL+ Translator FATAL ERROR; the step is terminated prior to completion of output source generation.

Since the condition code reflects only the most severe error conditions, less severe errors may exist.

Memory Considerations

The following formula can be used to determine how much memory is needed to execute CA-MetaCOBOL+. Specify this value as the MVS REGION size.

```
region size = 184K + buffers + COBOL DDs + CRSTACK + input exits + macro sets
```

184K	Minimum MVS REGION size of CA-MetaCOBOL+. The size will vary slightly depending upon the DIALECT option. DIALECT=WO requires the most storage. The other dialects require slightly less. Buffers are included in the 184K size for the FM, FE, LSTIN, PUNCHF, and CARDF data sets.
Buffers	Size of the buffers for the following data sets: ACCT, AUX, CYCLE, MASTER, RELOAD, SNAPER, SYSLIB, SYSTERM, and UNLOAD. Buffer space need only be specified for data sets that are required by the translation.
COBOL DDs	COBOL Data Definitions. 62 bytes are required for each data definition. Storage is allocated in 2K blocks, so storage must be rounded up to the next 2K boundary.
CRSTACK	Value of the CRSTACK option.
input exits	Size of any input exits called in the translation.
macro sets	Size of any macro sets loaded for the translation.

Module Name	Description	Memory Required
DATACOM/DB Facility		
SPP	Structured Programming Procedure and	
DLM	Data Language/DATADICTIONARY Interface Macros	420K
DLM	Data Language/DATADICTIONARY Interface Macros	380K
Input Exits		56K
Quality Assurance (QA) Facility		
CQA	COBOL Quality Assurance	490K
DCQA	COBOL w/ DL verbs QA	
SQA	Structured Programming Facility QA	600K
DSQA	Structured Programming Facility w/ DL verbs QA	
VQA	VS COBOL II Quality Assurance	
DVQA	VS COBOL II w/ DL verbs QA	
Structured Programming (SP) Facility		
SPP	Structured Programming Procedure	64K
SP2	VS COBOL II Structured Programming	590K
SPP	Structured Programming Procedure	
SPD	and SP Documentor	64K
SPP	Structured Programming Procedure	
SPS	and SP Stub Generator	120K
GPV	General Programming Verbs	170K
Formatting		
HLF	High-level Formatter	64K
Online Programming Language		
OPLPS	OPL Program Structuring	290K
OPL	Online Programming Language	300K

Figure 4. Memory Required for Combinations of CA supplied Modules

6.3.2 Translations Under VSE

The following sections describe the requirements to execute CA-MetaCOBOL+ in a VSE environment.

VSE File Requirements

The following list contains the applicable VSE file assignments and file names with a brief description of their use. (The file names are in parentheses.)

Note: When the SYSIPT installation parameter has been specified, SYSIPT is the Primary Input File. No concatenation is permitted.

SYSCAT (IJSYSCT or IJSYSUC)	VSAM catalog for CA LIBRARIAN VSAM disk master input; required with SYS005 when the CA LIBRARIAN input is a VSAM disk master file.
SYSIPT (IJSYSIN)	Primary input concatenation to SYSRDR on cards, tape, or disk; requires SYSIPT Translator Option.
SYSLST (IJSYSLS)	Listing File; the printer file for the Input, Output, Statistics, Auxiliary, and Lost Text Listings.
SYSRDR (IJSYSIN)	Primary Input File; the primary source of Translator Options, macros, and source code.
SYSSLB (IJSYSSL)	Source Library Input File; required for recognition of private source statement libraries when the input contains &COPY, *$COPY, and COBOL COPY statements. If LIBDEF control is used, the file name(s) may vary.
SYS001 (IJSYS01)	Required In-Line Work File.
SYS002 (IJSYS02)	Required Out-of-Line Work File.
SYS003 (IJSYS03)	Pre-Compiled Macros; required when *storing* pre-compiled macros. UNLOAD must be a sequential file.
SYS004 (IJSYS04)	Pre-Compiled Macros; required when retrieving pre-compiled macros. RELOAD must be a sequential file.
SYS005 (MASTER)	CA LIBRARIAN Disk Master Input; required when the input contains *$LIBED statements.
SYS006 (IJSYS06)	Generated COBOL Output File; required when generated COBOL is punched or written to tape or disk.

SYS007 (IJSYS07)	Auxiliary File; required when generated auxiliary data is punched or written to tape or disk.
SYS008 (TMAST)	CA LIBRARIAN Tape Master Input File; required when the input contains *$LIBET statements.
SYS009 (CYCLE)	CA LIBRARIAN Tape Master Cycle Control File; required when the input contains *$LIBET statements and the CA LIBRARIAN tape master is under cycle control.
SYS010 (IJSYS10)	Accounting File; required when low-volume accounting data is punched.

VSE Translation Execution JCL

The job stream outlined below is supplied as a guide for executing CA-MetaCOBOL+ in a VSE environment. (Standard labels for the disk work areas may be used.):

```
// LIBDEF ...
// DLBL   IJSYS01,...
// EXTENT SYS001,...                     Work File
// ASSGN  SYS001,DISK,VOL=volserno,SHR
// DLBL   IJSYS02,...
// EXTENT SYS002,...                     Work File
// ASSGN  SYS002,DISK,VOL=volserno,SHR
// DLBL   IJSYS06,...                    To 'punch' CA-MetaCOBOL+
// EXTENT SYS006,...                     output to a disk
// ASSGN  SYS006,DISK,VOL=volserno,SHR
// DLBL   IJSYS07,...
// EXTENT SYS007                         Auxiliary output
// ASSGN  SYS007,X'cuu'
// DLBL   IJSYSSL,...                    Private source
// EXTENT SYSSLB,...                     statement library
// ASSGN  SYSSLB,...
// EXEC METACOB
     [OPTION options]
     [macros]
     source program
/*
```

Note: In DOS/VS or DOS/VSE, do not specify SIZE=AUTO on the EXEC statement.

Concatenating Source Input

The SYSRDR containing macros and source input can indicate that all input originates from the same device or that part originates from a source library.

To include input from a source library, specify the *$COPY Translator Directing statement:

```
     // EXEC METACOB
             [OPTION options]
             [macros]
            *$COPY module-name ...
```

To include input from a CA LIBRARIAN disk master, specify the master file (SYS005). The input module is retrieved during translation with the *$LIBED Translator Directing statement. If the input file is a VSAM file, the VSAM catalog for CA LIBRARIAN disk master input must be specified as shown below.

```
 // DLBL IJSYS CT ,...,VSAM If VSAM master file.
              {UC}
 // EXTENT SYSCAT,volser
[// ASSGN SYSCAT,X'cuu'    ]

 // DLBL MASTER,..., DA
                    {VSAM}
 // EXTENT SYS005,...
 // ASSGN SYS005,X'cuu'
 // EXEC METACOB
       [OPTION options]
       [macros]
       *$LIBED module-name ...
```

To include input from a CA LIBRARIAN tape master, specify the tape master input (SYS009) and the CYCLE control file (SYS009). The input module is retrieved during translation with the *$LIBET Translator Directing statement.

```
      // ASSGN SYS008,X'cuu'
      // DLBL CYCLE,...
      // EXTENT SYS009,...
      // ASSGN SYS009,X'cuu'
      // EXEC METACOB
              [OPTION options]
              [macros]
             *$LIBET module-name ...
```

In the examples above, note that macro sets must be input ahead of the source input.

Specifying Translate-Time Options

The options specified in one or more OPTION cards specify the Translator Options in effect for the current execution of the CA-MetaCOBOL+ Translator. Refer to Section 6.2.1 for a description of the OPTION card and a complete list of the options.

The following example illustrates the placement of the OPTION card:

```
// EXEC METACOB
      OPTION LISTIN,DECK
      *$COPY ...
       IDENTIFICATION DIVISION.  ...
```

When CA-MetaCOBOL+ is installed with no SYSIPT parameter, the Primary Input File is assigned to SYSRDR. However, a facility similar to concatenation under MVS is provided by the SYSIPT Translator Option. If the option SYSIPT is specified in an OPTION card obtained from SYSRDR, then the SYSIPT file is accessed as the Primary Input File when SYSRDR is exhausted.

For example, if the source program resides on tape or disk, SYSIPT can be assigned to that device and SYSRDR to a card reader. The //EXEC statement and SYSRDR input can provide the proper OPTION card and macro library access statements as follows:

```
// EXEC METACOB
      OPTION SYSIPT, ...
      *$COPY MACROS
       IDENTIFICATION DIVISION.  ...
```

After loading the macros, SYSRDR is exhausted, and the CA-MetaCOBOL+ Translator reads the remainder of the Primary Input File from SYSIPT.

When CA-MetaCOBOL+ is installed with the SYSIPT parameter, the Primary Input File is assigned to SYSIPT, concatenation is not supported, and specification of the SYSIPT Translator Option is ignored.

VSE Condition Codes

The CA-MetaCOBOL+ Translator passes a condition code representing the severity of the highest-level diagnostic in the step. This condition code can be tested. Condition codes may be set in macro definitions by the &FLAG, &NOTE, or &COND macro language directives, which are described in the CA-MetaCOBOL+ *Macro Facility Reference*.

The following are the default condition codes. Additional codes may be set by macro definitions.

Code	Meaning
0	No diagnostic generated.
4	The highest-level diagnostic is caused by a &FLAG (N98) or &NOTE (N99) macro language directive.
8	The highest-level diagnostic is a CA-MetaCOBOL+ Translator WARNING.
12	The highest-level diagnostic is a CA-MetaCOBOL+ Translator ERROR or LOST TEXT source.
16	The highest-level diagnostic is a CA-MetaCOBOL+ Translator FATAL ERROR; the step is terminated prior to completion of output source generation.

Memory Considerations

The following formula can be used to determine how much memory is required to execute CA-MetaCOBOL+.

```
region size = 184K + buffers + COBOL DDs + CRSTACK + input exits + macro sets
```

184K	Minimum size of CA-MetaCOBOL+. This space includes the size for the IJSYS01, IJSYS02, IJSYS06, and IJSYSIN files.
Buffers	Size of the buffers for the following files: CYCLE, IJSYS03, IJSYS04, IJSYS07, IJSYS10, MASTER, and TMAST. Space need only be reserved for a file if it is required by the translation.
COBOL DDs	COBOL Data Definitions. 62 bytes are required for each data definition. Storage is allocated in 2K blocks, so storage must be rounded up to the next 2K boundary.
CRSTACK	Value of the CRSTACK option.
input exit	Size of any input exits called in the translation.
macro sets	Size of any macro sets loaded for the translation.

6.3.3 Translations Under CMS

The following sections describe the CA-MetaCOBOL+/CMS operating instructions.

Executing the CA-MetaCOBOL+ Translator

The CA-MetaCOBOL+ Translator is installed as a MODULE file. Execution can be initiated by specifying the module-name as follows:

METACOB *operand* [*operand*] ... [([PRT] [PCH])]

Each operand in the command is used by the CA-MetaCOBOL+ Translator as the filename of an input file on an accessed disk. At least one operand must be specified. Multiple operands are treated as a concatenated input specification, thus permitting OPTION statement files, files established by the LIBFETCH option of CA LIBRARIAN, and other existing files to be defined as multi-file input to the CA-MetaCOBOL+ Translator.

The filetype of all CA-MetaCOBOL+ input must be MCTIN. All input files must consist of fixed-length 80-byte records. The search for each specified file is in standard disk-search order. The last operand specified is also used as the filename for all work files and output files created by the CA-MetaCOBOL+ Translator. The filetypes used are the following:

MCTFM	Required In-Line work file.
MCTFE	Required Out-of-Line work file.
COBOL	COBOL Output File; required when generated COBOL is written to an output file.
MCTAUX	Auxiliary File; required when generated auxiliary data is punched or written as an output file.
MCTLIST	Listing File; printer file required for the Input, Output, CA-MetaCOBOL+ Statistics, Auxiliary, and Lost Out-of-Line Listings.
MLOAD	RELOAD or UNLOAD File; required when pre-compiled macros are *retrieved* or *stored* during translation.

The work files are created on the READ/WRITE disk having the most available free space. The output files are created on the disk that meets the highest of the following high-to-low ordered criteria:

- The disk that contains the last specified (prime) input file, if the disk is READ/WRITE.
- Its READ/WRITE parent, if it is a READ/ONLY extension thereof.
- The A-disk, if the A-disk is READ/WRITE.
- The disk containing the work file.

An informational terminal message (X07) is produced at CA-MetaCOBOL+ Translator initialization that indicates which disks are used for output and work files. Any existing file on the selected disk is destroyed if an output or work file of the same filename and filetype is used during a CA-MetaCOBOL+ Translator execution.

The output control codes PRT and PCH may be specified in an option list appended to the command. PRT causes direct spooling of the CA-MetaCOBOL+ Translator listing output to the virtual printer in place of the creation of a listing file. PCH causes direct spooling of the standard output to the virtual punch in place of the creation of a standard output file. PCH also automatically forces a DECK Translator Option status.

In the following command, METACOB is the name under which the CA-MetaCOBOL+ Translator is installed. The input files to be used during this execution of CA-MetaCOBOL+ are FILE1, FILE2, and FILE3. FILE3 is used as the filename of all work and output files. Both of the control codes governing spooling are specified.

```
METACOB FILE1 FILE2 FILE3 (PRT PCH)
```

Note that when these files were created, they were identified as FILE1 MCTIN, FILE2 MCTIN, and FILE3 MCTIN. Remember that the search for the input files is in standard disk-search order, so care must be exercised that similarly named files are not inadvertently input to the CA-MetaCOBOL+ Translator.

GLOBAL MACLIB Definitions

The CA-MetaCOBOL+ Translator uses the current GLOBAL MACLIB definition to resolve standard COBOL COPY, *$COPY, and &COPY statements. Specification of the GLOBAL must be done before CA-MetaCOBOL+ Translator execution if any of these are to be resolved. If any of the libraries is a PDS on an MVS format disk, an appropriate FILEDEF must be issued for each PDS before CA-MetaCOBOL+ Translator execution, for example:

```
FILEDEF SYSLIB DISK Z MACLIB C DSN MCT MACS (CONCAT)
```

defines the data set MCT.MACS on the MVS C-disk, with concatenation to be indicated in the current GLOBAL MACLIB. Z is the PDS's reference in the GLOBAL MACLIB.

If the alternate COPY library facility of OS/VS COBOL is used in any COBOL COPY statement, a FILEDEF command for each library referenced must be issued before CA-MetaCOBOL+ Translator execution (with the DDNAME of the FILEDEF identical to the library name in the COPY statement). This is required for both PDSs on MVS format disks and MACLIBs on CMS format disks. Such libraries must not be included in any GLOBAL MACLIB unless they are also to take part in the resolution of any standard COBOL COPY, *$COPY, or &COPY statements. In such cases, a separate SYSLIB FILEDEF must be issued for each libraries.

The CA LIBRARIAN Tape Master Usage

If a CA LIBRARIAN tape master is used to provide input during a CA-MetaCOBOL+ Translator execution, a tape drive must be attached to the user's CMS machine and the tape mounted prior to execution of the CA-MetaCOBOL+ Translator module. The tape must be positioned at the start of the CA LIBRARIAN file (after any standard labels). In addition, an appropriate FILEDEF must be in effect for the CA-MetaCOBOL+ Translator execution:

```
FILEDEF MASTIN TAPn BLKSIZE b DEN d
```

Cycle-controlled mounting is not available.

Translator Functions Under CMS

Differences in the following functions should be noted when comparing MVS or VSE with CMS use of the CA-MetaCOBOL+ Translator:

- The CLOCK translate-time is always available. The CLOCK Translator Option and the &CLOCK directive displays are normally in elapsed CMS dispatched time, unless VM uses the real-time option.

- The *$LIBED function is not supported.

- The Accounting File (ACCT) is not supported.

- The DECK Translator Option is forced whenever the output control code PCH is specified.

- The Terminal File facility is available if it has been generated into the CA-MetaCOBOL+ Translator. The TERM= Translator Option produces no disk file output; it outputs the SYSTERM file contents to the terminal.

7. Directory of Macro Sets and Programs

This section describes the CA-MetaCOBOL+ standard programs and product options provided by Computer Associates. In each description, the appropriate technical documentation is referenced for more information.

7.1 Standard Programs

The following programs are provided with each installation of CA-MetaCOBOL+.

7.1.1 Account Management Review

The Account Management Review (AMR) is a CA-MetaCOBOL+ source program that produces reports on CA-MetaCOBOL+ usage by both programmer and type of activity. AMR evaluates the usage of CA-MetaCOBOL+ by programmers and provides status reports of systems under development.

Appendix E describes how to operate AMR.

7.1.2 CA-MetaCOBOL+ Procedure Documentor

The CA-MetaCOBOL+ Procedure Documentor (MPD) aids macro set development and maintenance. The input to MPD may be any macro set or series of macro sets. The output consists of a listing of the macro set(s) and such optional reports as cross-references of variables, labels and tags, a macro index, a list of all NOTEs and FLAGs, and a special word directory.

For additional information on MPD, refer to the CA-MetaCOBOL+ *Macro Facility Reference*.

7.1.3 CA-MetaCOBOL+ Structured Program Documentor

The CA-MetaCOBOL+ Structured Program Documentor (SPDCOB) aids structured program development and maintenance. The input to SPDCOB is data extracted from a CA-MetaCOBOL+ structured program translated using SPP and SPD macro sets or the SP2 macro set. The output consists of a module hierarchy report and a cross-reference listing of where the modules are invoked.

For additional information on SPDCOB, see the CA-MetaCOBOL+ *Structured Programming Guide*.

7.2 CA-MetaCOBOL+ Macro Sets

CA-MetaCOBOL+ product options consist of macro sets that solve common problems with programming in COBOL. Each product option and its macro sets are described in this section. You can use the CA-MetaCOBOL+ facilities and their corresponding macro sets alone or in any combination. These macro sets <u>MUST be loaded in the order shown below</u>. If you are using the Structured Programming Facility, refer to Section 7.2.4 for further important information about loading the macro sets.

- Structured Programming
- Online Programming
- CA DATACOM/PC
- String Manipulation Language
- General Programming Verbs

7.2.1 Online Programming Facility

The CA-MetaCOBOL+ Online Programming Facility provides COBOL language extensions for psuedo-conversational CICS transactions. These macro sets allow you to embed Online Programming Language (OPL) statements in COBOL programs and generate psuedo-conversational CICS transactions.

There are two macro sets:

OPLPS	Provides for structuring of the input program source, preparing the input source for pseudo-conversational code generation.
OPL	Provides for translation of high-level OPL statements into conventional COBOL and command-level CICS statements.

For more information, see the CA-MetaCOBOL+ *Online Programming Language Reference* manual.

7.2.2 CA DATACOM/DB Facility

The CA-MetaCOBOL+ CA DATACOM/DB Facility provides COBOL language extensions to access CA DATACOM/DB databases. These macro sets allow you to embed CA DATACOM/DB Facility statements in COBOL programs that generate a CALL-level access to CA DATACOM/DB. CA DATACOM/DB Facility statements are easy to code because they are high-level and COBOL-like.

There is one CA DATACOM/DB macro set for use with CA-MetaCOBOL+. This macro set, called DLM, has several features:

- Provides basic driver macros for the CA DATACOM/DB interface, translating high-level COBOL/DL statements into CA DATACOM/DB parameters, work areas, and CALL statements in standard COBOL.
- Supports CA DATADICTIONARY access.
- Supports coding with CA-MetaCOBOL+ structured programming constructs
- Supports coding with VS COBOL II constructs
- Prepares a program for OPL translation

For more information, refer to the CA-MetaCOBOL+ *Program Development Reference - CA DATACOM/DB* manual.

7.2.3 The String Manipulation Language (SML) Facility

The COBOL String Manipulation Language (COBOL/SML) is an extension to the COBOL programming language providing string handling and inspection capabilities unavailable in COBOL.

String manipulation functions can be achieved using COBOL language statements such as STRING, UNSTRING, INSPECT, and TRANSFORM. Unfortunately, TRANSFORM is a statement discontinued in VS COBOL II. Also, with some COBOL compilers, INSPECT, STRING, and UNSTRING require the use of operating system services, prohibiting their use in CICS programs. The String Manipulation Language does not require the use of operating system services.

The syntax for COBOL string manipulation statements mentioned above is not consistent between any two statements. The COBOL/SML language has the consistency that each statement has a STARTING AT and FOR LENGTH OF clause available to it, making them easier to use than their COBOL counterpart. The presence of these two clauses in each of the COBOL/SML statements allows for the manipulation to be confined to specific parts of fields--a capablility not found in all COBOL string manipulation statements.

The macro set for string manipulation is called SML. For further instructions to use the SML macro set, refer to the CA-MetaCOBOL+ *String Manipulation Guide*.

7.2.4 The Structured Programming (SP) Facility

CA-MetaCOBOL+'s Structured Programming Procedures extend the COBOL language by providing comprehensive modularization and contemporary control structures for repetition and selection.

The Structured Programming macro sets are shown below. Only the SPP macro set can be used with the OPL and CA DATACOM/DB macro sets. Therefore, if your program uses OPL or CA DATACOM/DB statements in adddition to Structured Programming Statements, you must use the SPP macro set. The SPP macro set MUST be loaded first.

The SP2 macro set, if used, MUST be used alone. It cannot be used with any other Structured Programming macro set or any other CA-MetaCOBOL+ macro set. The SP2 macro set provides VS COBOL II constructs that perform the same function as the SPP macro set. In addition, SP2 includes the same capabilities as SPS and SPD. The macro sets are:

SPP Structured Programming Procedure generates ANSI 74 compilable COBOL source from the repetition and selection constructs (LOOP... WHILE... ENDLOOP, IF... ELSE... ENDIF, and SELECT... WHEN... ENDSELECT)

SP2 Is an alternative to the SPP macro set. It generates code that makes use of the new control structures available with the VS COBOL II compiler.

SPS Structured Programming Stub Generation automatically defines referenced procedure names that are undefined. This ensures that a compiler error due to an undefined procedure name will not occur.

SPD Structured Programming Documentation. When used with its companion source program, SPDCOB, SPD produces both a module hierarchy report and a report of where modules are invoked. This aids in top-down program development and maintenance.

For more information, refer to the CA-MetaCOBOL+ *Structured Programming Guide*.

7.2.5 General Programming Verbs

The General Programming Verbs (GPV) are coded in CA-MetaCOBOL+ input and facilitate program development. The following verbs are provided by the GPV macro set:

`ARRANGE`	Sorts tables.
`CLEAR/RESET`	Clears data areas.
`COLLATE`	Controls reading of two sequential input files to facilitate the updating of a master file from an update file, matching files for reporting purposes, or merging two sequential files.

`OPEN FAIR`
`START FAIR`
`READ FAIR`
`CLOSE FAIR`
	Accesses CA LIBRARIAN master.

`PARM`	Analyzes an MVS PARM field.

`IF-BITS`
`EXPAND-BITS`
`COMPRESS-BITS`
	Enables bit manipulation.

Refer to the CA-MetaCOBOL+ *Structured Programming Guide* for a detailed description and the syntax of each function.

7.2.6 The Quality Assurance (QA) Facility

The CA-MetaCOBOL+ Quality Assurance (QA) Facility product option protects important program assets by providing quality assurance, documentation, and standardization of COBOL source programs. QA Facility features include:

- Formatting, editing, and standardizing procedures to enhance program readability.
- Program-auditing and data-mapping facilities to guarantee a consistent standard of program quality.

The macro sets are:

CQA COBOL Quality Assurance (CQA) is a procedure for auditing and standardizing conventional COBOL source programs.

DCQA COBOL Quality Assurance (CQA) is a procedure for auditing and standardizing conventional COBOL source programs that contain DL verbs.

SQA Structured Programming Quality Assurance (SQA) is a procedure for auditing and standardizing COBOL/SP source programs.

DSQA Structured Programming Quality Assurance (SQA) is a procedure for auditing and standardizing COBOL/SP source programs that contain DL verbs.

VQA VS COBOL II Quality Assurance (VQA) is a procedure for auditing and standardizing VS COBOL II source programs.

DVQA VS COBOL II Quality Assurance (VQA) is a procedure for auditing and standardizing VS COBOL II source programs that contain DL verbs.

For more information, refer to the CA-MetaCOBOL+ *Quality Assurance Guide*.

7.2.7 The CA-MetaCOBOL+ Formatter

The CA-MetaCOBOL+ Formatter (HLF) formats programs to provide consistent indentation of SP Facility, CA DATACOM/DB Facility, and command-level CICS statements. It also formats VS COBOL II as well as COBOL 74 syntax. HLF automatically indents nested syntax, clauses, and conditions subordinate to a particular statement automatically, thus relieving the programmer of this time-consuming task.

The HLF macro set cannot be loaded for translation with any other macro set, except SPD. For details on other HLF requirements, refer to the CA-MetaCOBOL+ *Structured Programming Guide*.

7.2.8 Data Structure Table Mapping

The macro set MAPTDS is provided to aid in the debugging of user-defined and CA supplied macros. It can be included with any CA-MetaCOBOL+ translation, and produces an auxiliary report which maps the contents of the Data Structure Table for that translation.

Appendix A. Glossary

Active division

The COBOL source text division being processed. A COBOL division header (or the abbreviation $ID, $ED, $DD, or $PD) must be present in the source text to establish it as active.

You can limit the action of a macro set to a specific division by entering an active division code in Area A of a macro definition, next to the Macro Type coded in the Indicator Area. The macro will be executed when the specified division is active. The active division codes are:

I	Identification
E	Environment
D	Data
P	Procedure

If all active division codes are omitted, the macro will be active in all divisions.

When an out-of-line macro directive is used, the Active Division will change if the directive is for a division other than the currently active division. The change in the active division will change which macros execute until another directive specifying a different division is executed, the macro terminates, or the &END macro directive is executed. At this point, the original active division is restored. If the &NOEND macro directive is used, the active division established by the out-of-line macro directive remains in effect after the macro terminates, until another out-of-line macro directive executes or until the &END directive is issued.

Alphabetic character

Characters in the sets a through z and A through Z.

Area A

Columns 8-11 of a COBOL source record. Only the first word that begins in Area A is an Area A word. Another word on the same line that starts in Area A is actually an Area B word. COBOL Division and Section headers, paragraph headers and names, level indicators, and level numbers 01 and 77 must be in Area A. CA-MetaCOBOL+ active division codes must also be in Area A.

Area A indicator

A code which is inserted ahead of a word during translation when the source text word or macro model constant is found in Area A. The Area A indicator causes the word following the indicator to be placed in Area A of the output.

Area B

Columns 12-72 of a COBOL source record. Any word within these columns, or a word beginning in Area A after another word, is an Area B word.

Character

A letter, digit, or symbol. Major classifications of characters are:

- Alphabetic (letters a-z and A-Z)
- Numeric (digits 0-9)
- Punctuation (comma, semicolon, period, and colon)
- Sign (plus (+) and minus (-))
- Special characters (symbols that are neither alphabetic nor numeric).

Comment lines

Any line with an asterisk or slash in the indicator area. These lines may contain special codes or commands to control CA-MetaCOBOL+ input and output, or they may be standard COBOL comments.

Constructs (control structures)

Combinations of macro directives that define the logic and scope of selection and repetition.

Continuation column

Column 7 of source records, also known as the Indicator Area.

Current value

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

Any sentence in the DATA DIVISION beginning with a level number and ending with a period. These definitions may include a name and any clauses essential to describing the item.

Data-name

A word containing at least one alphabetic character that defines a data item in the Data Division.

Directive

A word coded in a macro model that starts a Translator action. These words will not appear in the output. They are similar in function to COBOL verbs that define the actions taken by a COBOL program. A directive always begins with an ampersand (&).

Directive expression

A series of words, beginning with a directive, that defines a specific Translator action. These expressions are similar in function to COBOL statements.

Division header

The words that indicate the beginning of a COBOL division. Division headers must begin in Area A. The following are valid COBOL division headers:

ID DIVISION	DATA DIVISION
IDENTIFICATION DIVISION	PROCEDURE DIVISION
ENVIRONMENT DIVISION	PROCEDURE DIVISION USING ...

CA-MetaCOBOL+ also recognizes the codes $ID, $ED, $DD, and $PD in Area A as division headers.

Event

The end of a division, section, data definition, paragraph, or statement. Predefined event macro names provide a method of executing a macro-guided task when an event occurs. The event macro names include:

- $-LEVEL executes at the end of a data definition in the Data Division.
- $DDE executes at the end of the Data Division. Data Division and Procedure Division headers must be present in the source text.
- $DDX executes at the end of the Data Division, after any $DDE macro has ended. Data Division and Procedure Division headers must be present in the source text.
- $-PROC executes at the end of a paragraph in the Procedure Division, and is prompted by any word coded in Area A.
- $-VERB executes at the end of a paragraph or statement in the Procedure Division, and is prompted by a verb or period ending a header or sentence. If $-PROC is also coded for a verb in Area A, the $-PROC macro has priority over the $-VERB macro.
- $PDE executes at the end of the source text or the end of a nested program. A division header for the next program must exist.
- $PDX executes at the end of the source text or the end of a nested program after the $PDE macro has ended.

Execute time

The time when a COBOL program is run, after translation, compilation, and link-editing.

Figurative constant

The reserved names for the manifest constants ZERO, ZEROS, ZEROES, SPACE, SPACES, HIGH-VALUE, HIGH-VALUES, LOW-VALUE, LOW-VALUES, QUOTE, QUOTES, and ALL 'literal', and the user-defined names for symbolic characters as defined within COBOL.

Floating point literal

A number represented by a mantissa and an exponent. The mantissa can be 1 to 16 digits long, must include a decimal point, and can have a leading sign. An E followed by an optional sign and one or two digits represents the exponent.

Generated program text

The output produced by a translation. The generated program text is usually a complete COBOL program.

Hexadecimal literal

In macro models, a string of 1 to 128 pairs of hexadecimal digits (0-9, A-F) bound by pairs of single or double quotation marks (for example, ""C1"" or ''C1''). Each digit pair represents one EBCDIC character. The first and the last quotation mark must be preceded and followed by a space or margin. Macro assembly converts the literal to a non-numeric literal. A hexadecimal literal is valid anywhere a non-numeric literal is valid and continuation rules are the same for both types of literal.

In a COBOL program, a hexadecimal literal is a string of 1 to 160 pairs of hexadecimal digits (0-9, a-f, and A-F), bound by single or double quotation marks, and preceded by an X (for example, X"C1" or X'C1'). A hexadecimal literal is valid anywhere a non-numeric literal is valid, and continuation rules are the same for both types of literal. However, the initial X may not be separated from the hexadecimal digit pairs by a continuation.

Identification Area

Columns 73 through 80 in a COBOL source program. This area can be used to make a brief comment or a consistent identification throughout the program.

Identifier

A unique data name, or a data name made unique by qualification, subscripts, or both.

Indicator Area

Column 7 in a COBOL source program. The indicator area is used to specify continuation of words or non-numeric literals from the previous line only the current line, treatment of text as documentation (comments), or debugging lines.

Keyword

A constant that can be used in a string macro prototype after the macro name. The compiler directing words *CBL, *CONTROL, CBL, CONTROL, EJECT, NOTE, REMARKS, SKIP1, SKIP2, SKIP3, and TITLE cannot be used as keywords in string macro prototypes.

Label

A logical destination defined within a macro model. Labels take the form &L*name* and may be coded in Area A or Area B of the model. A label must be unique within the macro region where it is defined. It can be referenced by macro directives in other macro models.

Level indicator

Identifies a specific type of file or a position in the data hierarchy. Level indicators include the words FD, SD, RD, and CD.

Level number

an integer at the beginning of a line in the Data Area that can be preceded by an Area A indicator.

Literal

A word or value that represents itself. For information on specific types of literals, see the entries for Figurative constant, Non-numeric literal, Hexadecimal literal, and Numeric literal in this glossary.

Logical destination

Special words preceding directive expressions and replacement words in a macro model (see the entries for Label and Tag in this glossary). Logical destinations declare a place within the model. When specified in a directive expression, the logical destination names the location to which control is to be transferred.

Macro

The equation of a word or word pattern (macro prototype) in the input source program to pre-defined rules (the macro model). The macro may generate output statements either through direct substitution or through conditional, logical analysis.

Macro call

The execution of a macro model initiated by a match of the macro prototype from the input stream, by a word substituted from another macro model, or by an event. A completed macro call removes any matching words from the text.

Macro definition

The explicit definition of a macro type, active division, macro prototype, and macro model. A macro model can be defined through omission.

Macro loading

Reading and assembling macros in preparation for translation.

Macro model

The specific translation action executed upon recognition of a source word, syntax pattern, word prefix, or event that matches the corresponding macro prototype. When omitted, the action consists of removal of any text matching the prototype.

Macro name

The first word of a macro prototype. The macro name cannot exceed 30 alphanumeric characters. (The arithmetic operators * and / can be specified as one-character prefix macro names only. A prefix macro name cannot exceed 29 characters.) A literal cannot be specified as a macro name nor can the compiler directing words *CBL, *CONTROL, CBL, CONTROL, EJECT, SKIP1, SKIP2, SKIP3, and TITLE. If the COMMENT option is specified, the words NOTE and REMARKS are not available. If the COPY option is PASSIVE or IGNORE, the word COPY is available.

Macro prototype

The explicit definition of a single word, a series of words, a partial word, or an event. The macro prototype, depending on its active division, is compared to words and events in the input stream and words output by other macros to determine when the macro model will execute.

Macro set

A group of macros intended to process a COBOL program. Macro sets can be independent or interdependent.

Macro type

There are 5 macro types: prefix (P), string (S), unverb (U), verb (V), and word (W). Code the macro type in the Indicator Area (column 7). For a complete definition and description of macro type, refer to the *Macro Reference Guide*.

Margin A
Area A.

Margin B
Area B.

Model
(see Macro model)

Nest level
The level at which a macro executes in a series of nested macro calls. A nested macro call is a macro definition invoked by a macro call embedded in another macro. Up to nine nested macro calls are supported. Any attempt to nest a tenth level will cause an error. Only one prefix macro can participate in a series of nested macro calls.

Nonnumeric literal
A constant of 1-160 characters and spaces bound by quotation marks. Double quotation marks (") or single quotation marks (') are equally acceptable in the same source text; however, they can not be mixed in the same literal. For example, a literal can not be marked by a quotation mark at the beginning and an apostrophe at the end.

The QUOTE and APOST translator options define how literals will be bound in the generated output text.

Numeric character
A character from 0 through 9.

Numeric literal
A constant containing from 1 to 18 digits. A numeric literal may contain a decimal point, and, as the left-most character, a sign character.

Out-of-line
The substitution of words in places other than directly in place of the input source statements.

Procedure header
The headers PROCEDURE DIVISION, DECLARATIVES, and END DECLARATIVES.

Procedure name
A paragraph name or section name in the Procedure Division. When referenced, a procedure name must be unique either in itself or by the addition of a section name qualifier.

Prototype
(see Macro prototype)

Procedure set
(see Macro set)

Punctuation character
Any of the following characters: colon (:), comma (,), equal sign (=), parenthesis (), period (.), semicolon (;), and space ( ).

Qualifier
A unique name hierarchically superior to a non-unique data name or paragraph name. To reference a non-unique name using a qualifier, follow the name with the preposition IN or OF and the qualifier. Paragraph names can have only one qualifier: the section name in which the paragraph is defined. Data names may have more than one qualifier.

Region
The arbitrary subdivision of multiple macro sets by the *$REGION Translator Directing statement. An initial region is implied, and ten regions are supported. Regions prevent identical variable and label names in different macro sets from conflicting when the combined macro sets execute in the same translation, but different regions.

Reserved variables
Predefined, external, nonnumeric variables with the following reserved names: &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
Columns 1 through 6 in COBOL. Use this area to label a source statement line. Any character may be coded in this area.

Sign character
Plus (+) and minus (-).

Space
One or more contiguous blanks (hexadecimal 40).

Source program text
The program text to be processed by a translation. It is usually a complete COBOL program.

Special character
Any character other than a space, an alphabetic character, or a numeric character.

Subscript
Acceptable macro variable subscripts are:

- Numeric literals
- Character variables (the value must be numeric)
- Numeric variables
- Reserved variables (the value must be numeric)

Substitution Word
A macro model symbolic or constant word substituted directly or indirectly into the generated output.

Symbolic operand

The macro notation & or &n, where n is 0-15. This notation represents the current value determined by words from the input stream, by source item acquisition, or by initialization.

Tag

A logical destination defined in the form "&Tname" within a macro model. The tag may be coded in "Area A" or "Area B" of the model and must be unique within the macro model in which it is defined. A tag can not be referenced by macro directives in other macro models.

TDS relative address

identifies the position of an entry in the TDS table (attribute table). Each entry on the table can be accessed according to its position. Each data name in the generated program can have an entry in the TDS table.

Terminating punctuation character

A punctuation character followed by a space, margin, or right parenthesis.

Translation

The modification, expansion, and analysis of standard COBOL or CA-MetaCOBOL+ input to produce COBOL output as determined by CA-MetaCOBOL+ macros, options, and Translator Directing statements.

Translate time

The time needed for the Translator to accomplish a task.

Variable

A named storage area available to macros.

Word

A string of characters that fits one of the following descriptions:

- Numeric literal
- Non-numeric literal
- Terminating punctuation character
- Parenthesis
- Picture character string
- Colon
- A contiguous character string not exceeding 30 characters that is bounded by a space, margin, or terminating punctuation character. The string may not contain an embedded space, quotation mark, or punctuation character. Examples of such words are a verb, reserved word, data name, and procedure name.

Appendix B. CA-MetaCOBOL+ Reserved Words

The following Reserved Words list consists of COBOL Reserved Words predefined to CA-MetaCOBOL+ for various COBOL compilers.

The CA-MetaCOBOL+ DIALECT option determines which reserved words are effective for a translation. A default DIALECT is established at installation. This default can be overridden.

The DIALECT option is a two character code, the first character specifies the compiler, the second character specifies the Operating system. The table below lists all the DIALECT option codes and the corresponding compiler description.

DIALECT option codes	COBOL Compiler
2D	DOS ANSI 68 Versions 1 and 2
3D	DOS ANSI 68 Version 3
VD	DSO/VS ANSI 68 Release 1
WD	DOS/VSE ANSI 74
2O	OS ANSI 68 Versions 1 and 2
3O	OS ANSI 68 Version 3
4O	OS ANSI 68 Version 4
VO	OS/VS ANSI 74 Language Level 1
WO	OS/VS ANSI 74 Language Level 2
XO	VS COBOL II Release 1 and 2
YO	VS COBOL II Release 3

The Reserved Words List has five columns. The first column lists the Reserved Words. Column two shows a one character code (I, E, D, P) for the COBOL division or divisions within which the word is used. Column three indicates the word type: Verb (V); Special Register (S); Figurative Constant (F); ordinary keyword (blank). The CA-MetaCOBOL+ Symbol Table will reflect the Verb, Figurative Constant, and Special Register attributes for those words in the list with these types. There is no attribute for an ordinary keyword. The fourth column indicates the MVS (OS) compiler codes (as in the above table) where the word is reserved. The fifth column indicates the same for VSE (DOS) compiler codes.

Reserved Word	COBOL Division(s)	Figurative Special Verb	Constant Register MVS	VSE
ACCEPT	P	V	234VWXY	23VW
ACCESS	ED	V	234VWXY	23VW
ACTUAL	E	V	234VW	23VW
ADD	P	V	234VWXY	23VW
ADDRESS	IEDP	S	XY	
ADVANCING	P		Y	
AFTER	P		Y	
ALL	EDP		234VWXY	23VW
ALPHABET	E	V	234VWXY	23VW
ALPHABETIC	P		234VWXY	23VW
ALPHABETIC-LOWER	P		Y	
ALPHABETIC-UPPER	P		Y	
ALPHANUMERIC	P		XY	
ALPHANUMERIC-EDITED	P		XY	
ALSO	P		Y	
ALTER	P	V	234VWXY	23VW
ALTERNATE	E	V	VWXY	VW
AND	P		Y	
AND	P	V	234VWX	23VW
ANY	P		Y	VW
APPLY	E	V	234VWXY	23VW
ARE	ED		234VWXY	23VW
AREA	E		Y	
AREAS	E		Y	
ASCENDING	DP		Y	
ASSIGN	E	V	234VWXY	23VW
AT			D	WXY
AT	P	V	234VWXY	23VW
AUTHOR	I		Y	
BEFORE	P		Y	

Reserved Word	COBOL Division(s)	Figurative Special Verb	Constant Register MVS	VSE
BEGINNING	P		Y	
BINARY	D		XY	
BLANK	D		234VWXY	23VW
BLOCK	D	V	234VWXY	23VW
BOTTOM	D		WXY	W
BY	IEDP		234VWXY	23VW
C01	E	V	234VWXY	23VW
C02	E	V	234VWXY	23VW
C03	E	V	234VWXY	23VW
C04	E	V	234VWXY	23VW
C05	E	V	234VWXY	23VW
C06	E	V	234VWXY	23VW
C07	E	V	234VWXY	23VW
C08	E	V	234VWXY	23VW
C09	E	V	234VWXY	23VW
C10	E	V	234VWXY	23VW
C11	E	V	234VWXY	23VW
C12	E	V	234VWXY	23VW
CALL	P	V	234VWXY	23VW
CANCEL	P	V	4VWXY	W
CD	D		4VWXY	
CHANGED	P		Y	
CHARACTER	ED		Y	
CHARACTERS	EDP		Y	
CLASS	E	V	234VWXY	23VW
CLOSE	P	V	234VWXY	23VW
CODE	D	V	VWXY	23VW
CODE-SET	D	V	WXY	W
COLLATING	E	V	234VWXY	23VW
COLLATING	P		Y	
COLUMN	D		234VWXY	23VW
COMMA	E		234VWXY	23VW
COMMON	I		Y	
COMMUNICATION	D		4VWXY	
COMP	D		234VWXY	23VW
COMP-1	D		234VWXY	23VW
COMP-2	D		234VWXY	23VW
COMP-3	D		234VWXY	23VW
COMP-4	D		234VWXY	23VW

Reserved Word	COBOL Division(s)	Figurative Special Verb	Constant Register MVS	VSE
COMPUTATIONAL	D		234VWXY	23VW
COMPUTATIONAL-1	D		234VWXY	23VW
COMPUTATIONAL-2	D		234VWXY	23VW
COMPUTATIONAL-3	D		234VWXY	23VW
COMPUTATIONAL-4	D		Y	
COMPUTE	P	V	234VWXY	23VW
CONFIGURATION	E		Y	
CONSOLE	E	V	234VWXY	23VW
CONTAINS	ED	Y		
CONTENT	P		Y	
CONTINUE	P	V	XY	
CONTROL	D	V	234VWXY	23VW
CONTROLS	D	V	234VWXY	23VW
COPY	IEDP		234VWXY	23VW
CORR	P		Y	
CORRESPONDING	P		Y	
COUNT	D		4VWXY	W
CSP	E	V	234VWXY	23VW
CURRENCY	E	V	234VWXY	23VW
CURRENT-DATE	IEDP	S	234VW	23VW
DATA	D	V	234VWXY	23VW
DATA	P		Y	
DATE	IEDP	S	4VWXY	W
DATE-COMPILED	I		Y	
DATE-WRITTEN	I		Y	
DAY	IEDP	S	4VWXY	W
DAY-OF-WEEK	IEDP	S	XY	
DBCS	P		XY	
DEBUG-CONTENTS	IEDP	S	WXY	
DEBUG-ITEM	IEDP	S	WXY	W
DEBUG-LINE	IEDP	S	WXY	
DEBUG-NAME	IEDP	S	WXY	
DEBUG-SUB-1	IEDP	S	WXY	
DEBUG-SUB-2	IEDP	S	WXY	
DEBUG-SUB-3	IEDP	S	WXY	
DECIMAL-POINT	E	V	234VWXY	23VW
DECLARATIVES	P		234VWXY	23VW
DELETE	P	V	234VWXY	VW
DELIMITER	E	E	Y	

Reserved Word	COBOL Division(s)	Figurative Special Verb	Constant Register MVS	VSE
DEPENDING	D		Y	
DEPTH	D		4V	
DESCENDING	DP		Y	
DESTINATION	D		4VWXY	
DISABLE	P	V	WXY	
DISPLAY	D		234VWXY	23VW
DISPLAY	P	V	234VWXY	23VW
DISPLAY-1	D		XY	
DISPLAY-ST	D		234VW	23VW
DIVIDE	P	V	234VWXY	23VW
DIVISION	IEDP		234VWXY	23VW
DOWN	P		Y	
DUPLICATES	E		Y	
DYNAMIC	E		Y	
EBCDIC	E		Y	
ELSE	P	V	234VWXY	23VW
ENABLE	P	V	WXY	
END	E		Y	
END	D		4VW	
END	P	V	234VWXY	23VW
END-ADD	P	V	XY	
END-CALL	P	V	XY	
END-COMPUTE	P	V	XY	
END-DELETE	P	V	XY	
END-DIVIDE	P	V	XY	
END-EVALUATE	P	V	XY	
END-IF	P	V	XY	
END-MULTIPLY	P	V	XY	
END-OF-PAGE	P	V	234VWXY	23VW
END-PERFORM	P	V	XY	
END-READ	P	V	XY	
END-RETURN	P	V	XY	
END-REWRITE	P	V	XY	
END-SEARCH	P	V	XY	
END-START	P	V	XY	
END-STRING	P	V	XY	
END-SUBTRACT	P	V	XY	
END-UNSTRING	P	V	XY	

Reserved Word	COBOL Division(s)	Figurative Special Verb	Constant Register MVS	VSE
END-WRITE	P	V	XY	
ENDING	P		Y	
ENTER	P	V	234VWXY	23VW
ENTRY	P	V	234VWXY	23VW
ENVIRONMENT	E		234VWXY	23VW
EOP	P	V	234VWXY	23VW
EQUAL	P		234VWXY	23VW
ERROR	D		4VWXY	
ERROR	P			
EVALUATE	P	V	XY	
EVERY	E	P	Y	
EXAMINE	P	V	234VW Y	23VW
EXCEPTION	P	V	XY	
EXHIBIT	P	V	234VWXY	23VW
EXIT	P	V	234VWXY	23VW
EXTEND	P		Y	
EXTERNAL	D		XY	
FALSE	P		Y	
FD	D		234VWXY	23VW
FILE	E	V	VWXY	VW
FILE	DP		234VWXY	23VW
FILE-CONTROL	E	V	234VWXY	23VW
FILE-LIMIT	E	V	234VW	23VW
FILE-LIMITS	E	V	234VW	23VW
FILLER	D		234VWXY	23VW
FIRST	P		Y	
FOOTING	D	V	WXY	W
FOR	E P		Y	
FOR	E	V	V	V
FOR	D		234	23
FROM	DP		Y	
GENERATE	P	V	234VW	23VW
GIVING	P		Y	
GLOBAL	D		XY	
GO	P	V	234VWXY	23VW
GOBACK	P	V	234VWXY	23VW
GREATER	P		234VWXY	23VW
GROUP	D		234VWXY	23VW

Reserved Word	COBOL Division(s)	Figurative Special Verb	Constant Register MVS	VSE
HIGH-VALUE	EDP	F	234VWXY	23VW
HIGH-VALUES	EDP	F	234VWXY	23VW
I-O	P		Y	
I-O-CONTROL	E		Y	
ID	I		Y	
IDENTIFICATION	I		Y	
IF	P	V	234VWXY	23VW
IN	EDP		234VWXY	23VW
INDEX	D		234VWXY	23VW
INDEXED	E	V	Y	
INDEXED	D		234VWXY	23VW
INITIAL	I		Y	
INITIAL	D		WXY	
INITIAL	P		Y	
INITIALIZE	P	V	XY	
INITIATE	P	V	234VW	23VW
INPUT	D		4VW	W
INPUT	P		Y	
INPUT-OUTPUT	E		Y	
INSPECT	P	V	WXY	W
INSTALLATION	I		Y	
INTO	P		Y	
INVALID	P	V	234VWXY	23VW
IS	I		Y	
IS	EDP		234VWXY	23VW
JUST	D		234VWXY	23VW
JUSTIFIED	D		234VWXY	23VW
KANJI	P		XY	
KEY	EDP		234VWXY	23VW
LABEL	D	V	234VWXY	23VW
LABEL	P		Y	
LABEL-RETURN	IEDP	S	234V	
LEADING	DP		Y	
LEFT	D		Y	
LENGTH	IEDP		S	XY
LENGTH	D		4VW	
LESS	P		234VWXY	23VW
LINAGE	D	V	WXY	W

Reserved Word	COBOL Division(s)	Figurative Special Verb	Constant Register MVS	VSE
LINAGE-COUNTER	IEDP	S	WXY	W
LINE	D		234VWXY	23VW
LINE-COUNTER	IEDP	S	234VW	23VW
LINES	D	V	WXY	W
LINES	P		Y	
LINKAGE	D		234VWXY	23VW
LOCK	P		Y	
LOW-VALUE	EDP	F	234VWXY	23VW
LOW-VALUES	EDP	F	234VWXY	23VW
MEMORY	E	V	234VWXY	23VW
MERGE	P	V	VWXY	VW
MESSAGE	D		4VWXY	
MODE	E		WXY	W
MODE	D		Y	
MODULES	E		Y	
MOVE	P	V	234VWXY	23VW
MULTIPLE	E	V	VWXY	VW
MULTIPLY	P	V	234VWXY	23VW
NAMED	P		Y	
NATIVE	E		Y	
NEGATIVE	P		Y	
NEXT	D		234VWXY	23VW
NEXT	P	V	234VWXY	23VW
NO	P		4VWXY	W
NOMINAL	E	V	234VW	23VW
NOT	P	V	234VWXY	23VW
NULL	EDP	F	XY	
NULLS	EDP	F	XY	
NUMERIC	P		234VWXY	23VW
OBJECT-COMPUTER	E		Y	
OCCURS	D		234VWXY	23VW
OF	EDP		234VWXY	23VW
OFF	E		234VWXY	23VW
OMITTED	D		Y	
ON	ED		234VWXY	23VW
ON	P	V	234VWXY	23VW
OPEN	P	V	234VWXY	23VW
OPTIONAL	E		Y	
OR	P		Y	

Reserved Word	COBOL Division(s)	Figurative Special Verb	Constant Register MVS	VSE
OR	P	V	234VWX	23VW
ORGANIZATION	E	V	VWXY	W
OTHER	P		Y	
OTHERWISE	P	V	234VW	23VW
OUTPUT	D		4VWXY	W
OUTPUT	P			
OVERFLOW	P	V	4VWXY	W
PACKED-DECIMAL	D		X	
PADDING	E	V	234VWXY	23VW
PAGE	D	V	234VWXY	23VW
PAGE-COUNTER	IEDP	S	234VW	23VW
PASSWORD	E	V	VWXY	VW
PERFORM	P	V	234VWXY	23VW
PIC	D		234VWXY	23VW
PICTURE	D		234VWXY	23VW
POINTER	D		XY	
POSITION	E		Y	
POSITIONING	P		Y	
POSITIVE	P		Y	
PRINT-SWITCH	IEDP	S	234VW	23VW
PROCEDURE	P		234VWXY	23VW
PROCEED	P		Y	
PROCESS	P	V	Y	
PROCESSING	E	V	VW	VW
PROGRAM	I P		Y	
PROGRAM	E	V	234VWXY	23VW
PROGRAM-ID	I		Y	
QUEUE	D		4VWXY	
QUOTE	EDP	F	234VWXY	23VW
QUOTES	EDP	F	234VWXY	23VW
RANDOM	E		Y	
RD	D		234VWXY	23VW
READ	P	V	234VWXY	23VW
READY	P	V	234VWXY	23VW
RECEIVE	P	V	4VWXY	
RECORD	E	V	234VWXY	23VW
RECORD	D	V	234VWXY	23VW
RECORD	P		Y	

Reserved Word	COBOL Division(s)	Figurative Special Verb	Constant Register MVS	VSE
RECORDING	D	V	234VWXY	23VW
RECORDS	ED		234VWXY	23VW
REDEFINES	D		234VWXY	23VW
REEL	E P		Y	
REFERENCE	P		Y	
RELATIVE	E	V	WXY	W
RELEASE	P	V	234VWXY	23VW
REMAINDER	P		Y	
REMOVAL	P		Y	
RENAMES	D		234VWXY	23VW
REPLACING	P		Y	
REPORT	D	V	234VW	23VW
REPORTING	P		Y	
REPORTS	D	V	234VW	23VW
RERUN	E	V	234VWXY	23VW
RESERVE	E	V	234VWXY	23VW
RESET	D		234VWXY	23VW
RESET	P	V	234VWXY	23VW
RETURN	P	V	234VWXY	23VW
RETURN-CODE	IEDP	S	234VWXY	W
REVERSED	P		Y	
REWIND	P		Y	
REWRITE	P	V	234VWXY	23VW
RIGHT	D		Y	
ROUNDED	P		Y	
RUN	P		Y	
S01	E	V	234VWXY	23VW
S02	E	V	234VWXY	23VW
S03	E	V	234VWXY	23VW
S04	E	V	234VWXY	23VW
S05	E	V	234VWXY	23VW
SAME	E	V	234VWXY	23VW
SD	D		234VWXY	23VW
SEARCH	P	V	234VWXY	23VW
SECTION	EDP		234VWXY	23VW
SECURITY	I		Y	
SEEK	P	V	234VW	23VW
SEGMENT-LIMIT	E	V	234VWXY	23VW

Reserved Word	COBOL Division(s)	Figurative Special Verb	Constant Register MVS	VSE
SELECT	E	V	234VWXY	23VW
SEND	P	V	4VWXY	
SENTENCE	P		Y	
SEPARATE	D		34VWXY	3VW
SEQUENCE	E	V	234VWXY	23VW
SEQUENCE	P		Y	
SEQUENTIAL	E	V	Y	
SET	P	V	234VWXY	23VW
SHIFT-IN	IEDP	S	XY	
SHIFT-OUT	IEDP	S	XY	
SIGN	ED		Y	
SIZE	ED		Y	
SIZE	P	V	234VWXY	23VW
SORT	E		Y	
SORT	P	V	234VWXY	23VW
SORT-CONTROL	IEDP	S	XY	
SORT-CORE-SIZE	IEDP	S	34VWXY	23VW
SORT-FILE-SIZE	IEDP	S	34VWXY	23VW
SORT-MERGE	E		Y	
SORT-MESSAGE	IEDP	S	34VWXY	
SORT-MODE-SIZE	IEDP	S	34VWXY	23VW
SORT-RETURN	IEDP	S	34VWXY	23VW
SOURCE	D		234VWXY	23VW
SOURCE-COMPUTER	E		Y	
SPACE	EDP	F	234VWXY	23VW
SPACES	EDP	F	234VWXY	23VW
SPECIAL-NAMES	E		234VWXY	23VW
STANDARD	DP		Y	
STANDARD-1	E		Y	
STANDARD-2	E		Y	
START	P	V	234VWXY	23VW
STATUS	E	V	VWXY	VW
STATUS	D		234VW	23VW
STOP	P	V	234VWXY	23VW
STRING	P	V	4VWXY	W
SUB-QUEUE-1	D		4VWXY	W
SUB-QUEUE-2	D		4VWXY	W
SUB-QUEUE-3	D		4VWXY	W

Reserved Word	COBOL Division(s)	Figurative Special Verb	Constant Register MVS	VSE
SUBTRACT	P	V	234VWXY	23VW
SUM	D		234VWXY	23VW
SUPPRESS	P		Y	
SYMBOLIC	E	V	234VWXY	W
SYMBOLIC	D		4VW	
SYNC	D		234VWXY	23VW
SYNCHRONIZED	D		234VWXY	23VW
SYSIN	E	V	234VWXY	23VW
SYSIPT	E	V	234VWXY	23VW
SYSLIST	E	V	234VWXY	23VW
SYSLST	E	V	234VWXY	23VW
SYSOUT	E	V	234VWXY	23VW
SYSPCH	E	V	234VWXY	23VW
SYSPUNCH	E	V	234VWXY	23VW
TABLE	D		WXY	
TALLY	IEDP	S	234VWXY	23VW
TALLYING	P		234VWXY	23VW
TAPE	E		Y	
TERMINATE	P	V	234VW	23VW
TEST	P		XY	
TEXT	D		4VWXY	
THAN	P		234VWXY	23VW
THEN	P		Y	
THROUGH	EDP		234VWXY	23VW
THRU	EDP		234VWXY	23VW
TIME	IEDP	S	4VWXY	W
TIME-OF-DAY	IEDP	S	234VW	23VW
TIMES	DP		234VWXY	23VW
TO	EDP		234VWXY	23VW
TOP	D	V	WXY	W
TRACE	P	V	Y	
TRACK-AREA	E	V	234VW	23VW
TRACK-LIMIT	E	V	234VW	
TRAILING	D		Y	
TRANSFORM	P	V	234VWXY	23VW
TRUE	P		Y	
TYPE	D		234VWXY	23VW
UNIT	E P	Y		
UNSTRING	P	V	4VWXY	W

Reserved Word	COBOL Division(s)	Figurative Special Verb	Constant Register MVS	VSE
UNTIL	P		XY	
UP	P		Y	
UPON	P		Y	
UPSI-0	E	V	234VWXY	23VW
UPSI-1	E	V	234VWXY	23VW
UPSI-2	E	V	234VWXY	23VW
UPSI-3	E	V	234VWXY	23VW
UPSI-4	E	V	234VWXY	23VW
UPSI-5	E	V	234VWXY	23VW
UPSI-6	E	V	234VWXY	23VW
UPSI-7	E	V	234VWXY	23VW
USAGE	D		234VWXY	23VW
USE	P	V	234VWXY	23VW
USING	P		Y	
VALUE	D	V	234VWXY	23VW
VALUES	D		234VWXY	23VW
VARYING	D		Y	
VARYING	P		XY	
WHEN	D		234VWXY	23VW
WHEN	P	V	234VWXY	23VW
WHEN-COMPILED	IEDP	S	VW	VW
WHEN-COMPILED	IEDP	S	XY	
WITH	E		Y	
WITH	D	V	WXY	W
WITH	P		XY	
WORDS	E		Y	
WORKING-STORAGE	D		234VWXY	23VW
WRITE	P	V	234VWXY	23VW
WRITE-ONLY	E		Y	
ZERO	EDP	F	234VWXY	23VW
ZEROES	EDP	F	234VWXY	23VW
ZEROS	EDP	F	234VWXY	23VW
<	E	P	234VWX	23VW
>	E	P	234VWX	23VW

Appendix C. Output Format Control

This appendix describes the FORMAT option and its parameters.

The FORMAT option determines how the generated COBOL source text is formatted. Macro-controlled formatting can enhance or even replace the FORMAT option.

Three named predefined sets of FORMAT option parameters may be established by the user at installation:

- ADR
- OPEN
- CLOSED

Refer to Section C.3, "Predefined Format Alternatives," for a definition of the parameters for ADR, OPEN, and CLOSED.

To use a set different form the default, specify the set you want with a FORMAT option similar to the one shown below.

```
OPTION FORMAT=CLOSED
```

You may establish additional sets of named, predefined FORMAT parameters at installation. Use the MCDEF and MCDEFCHK installation macros to define your own named sets of FORMAT parameters. Make sure your names are not the same as the three sets provided.

Use the FORMAT parameter of the MCGEN installation macro (not to be confused with the translate-time FORMAT option) to specify the name of the default FORMAT parameter set you want. You can define this set yourself or use one of the three provided. If you do not specify a set, the OPEN set will be used as the default.

You can override individual parameters of the default FORMAT option parameter at translate time by using a FORMAT option similar to the one shown below. When overriding parameters, you must enclose the new parameters within parentheses.

```
OPTION FORMAT=(PVBCD)
```

You can select a named set of FORMAT option parameters and, if necessary, use a second format option to override specific parameters, as shown below.

```
OPTION FORMAT=CLOSED,FORMAT=(NOIC)
```

You could also use the two separate OPTION cards below to accomplish the same task.

```
OPTION FORMAT=CLOSED
OPTION FORMAT=(NOIC)
```

If a large number of overrides will not fit onto a single card, use two format options on separate cards, as shown below.

```
OPTION FORMAT=(NOIC,NOEC,EV,NOES,E12,NODF,DT31,DI2X3)
OPTION FORMAT=(PC,PP,PI2X1,PVBCDEH)
```

C.1 FORMAT Option Parameters

This section provides the FORMAT parameters for each COBOL division. Parameters are:

- Grouped according to COBOL division
- Enclosed within parentheses. (One or more may be enclosed within the same set of parentheses.)
- Separated with commas
- May be specified in any order

C.1.1 Identification Division

IC The sentence after a paragraph header is placed on a new line. Use the NOIC option to keep the sentence after a paragraph header on the same line. An example of each type is shown below.

FORMAT=(IC)

```
IDENTIFICATION DIVISION.
PROGRAM ID.
   PGNAME.
AUTHOR.
   JOHN DOE.
```

FORMAT=(NOIC)

```
IDENTIFICATION DIVISION.
PROGRAM ID. PGNAME.
AUTHOR. JOHN DOE.
```

C.1.2 Environment Division

EC The sentence after a paragraph header is placed on a new line. Use the NOEC option to keep the sentence after a paragraph header on the same line. An example of each type is shown below.

FORMAT=(EC)

```
ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
SOURCE-COMPUTER.
           IBM-370.
OBJECT-COMPUTER.
IBM-370.
```

FORMAT=(NOEC)

```
ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
SOURCE-COMPUTER. IBM-370.
OBJECT-COMPUTER. IBM-370.
```

EV The following elements each begin a new line:

- Environment Division statements SELECT, APPLY, RERUN, and SAME
- All the clauses in the OBJECT-COMPUTER and SPECIAL-NAMES paragraphs.

Each clause and statement begins in column 12.

Using **NOEV** removes this formatting.

FORMAT=(EV,EC)

```
I-O-CONTROL.
           APPLY WRITE-ONLY ON MASTER-FILE
           APPLY CORE-INDEX ON ISAM-FILE.
```

FORMAT=(NOEV,NOEC)

```
I-O-CONTROL. APPLY WRITE-ONLY ON MASTER-FILE APPLY
           CORE-INDEX ON ISAM-FILE.
```

ESC*nn* Each SELECT statement clause (ASSIGN, RESERVE, ORGANIZATION, etc.) is indented to column *nn*, where *nn* is any column 12-42. The first clause is placed on a new line.

NOES will format the SELECT statement without indenting.

FORMAT=(EC,EV,ESC31)

```
FILE-CONTROL.
   SELECT MASTER
           ASSIGN TO UT-S-MFILEIN
           RESERVE NO ALTERNATE AREAS.
   SELECT LISTING
           ASSIGN TO UT-S-LISTOUT.
```

FORMAT=(EC,EV,NOES)

```
FILE-CONTROL.
   SELECT MASTER ASSIGN TO UT-S-MFILEIN RESERVE NO
   ALTERNATE AREAS.
   SELECT LISTING ASSIGN TO UT-S-LISTOUT.
```

EST*nn* Each SELECT statement clause (ASSIGN, RESERVE, ORGANIZATION, etc.) is indented to column *nn*, where *nn* is any column 12-42. The first clause is placed on a new line.

NOES will format the SELECT statement without indenting.

FORMAT=(EC,EV,EST31)

```
FILE-CONTROL.
   SELECT MASTER ASSIGN TO UT-S-MFILEIN
                 RESERVE NO ALTERNATE AREAS.
   SELECT LISTING          ASSIGN TO UT-S-LISTOUT.
```

EI*n* For statements requiring multiple lines, each line after the first will be indented "n" columns where n=1-6. **NOEI** removes this indentation.

FORMAT=(EC,EV,NOEI,NOES)

```
FILE-CONTROL.
   SELECT MASTER ASSIGN TO UT-S-MFILEIN RESERVE NO
   ALTERNATE AREAS.
   SELECT LISTING ASSIGN TO UT-S-LISTOUT.
```

FORMAT=(EC,EV,EI6,NOES)

```
FILE-CONTROL.
   SELECT MASTER ASSIGN TO UT-S-MFILEIN RESERVE NO
      ALTERNATE AREAS.
   SELECT LISTING ASSIGN TO UT-S-LISTOUT.
```

FORMAT=(EC,EV,EI2,ESC31)

```
FILE-CONTROL.
   SELECT MASTER
           ASSIGN TO UT-S-MFILEIN
           FILE-LIMIT IS RANDOM-ACCESS-LIMIT
           THRU RANDOM-ACCESS-LIMIT-END.
```

C.1.3 Data Division

DFC*nn* Each FD, SD, RD, and CD statement clause is indented to column "nn" where "nn" is column 12-42. The first clause is placed on a new line. Use **NODF** to produce the statement without this formatting.

FORMAT=(DFC25)

```
FD MASTER-FILE
           RECORDING MODE IS F
           BLOCK CONTAINS 10 RECORDS
           LABEL RECORD IS STANDARD
           DATA RECORD IS MASTER-IN.
```

FORMAT=(NODF)

```
FD MASTER-FILE RECORDING MODE IS F BLOCK CONTAINS 10
           RECORDS LABEL RECORD IS STANDARD
           DATA RECORD IS MASTER-IN.
```

DFT*nn* Each FD, SD, RD, and CD statement clause is indented to column *nn* where *nn* is any column 12-42.

FORMAT=(DFT25)

```
FD MASTER-FILE   RECORDING MODE IS F
                 BLOCK CONTAINS 10 RECORDS
                 LABEL RECORD IS STANDARD
                 DATA RECORD IS MASTER-IN.
```

DT*nn* indents the first clause of each data definition to column *nn* where *nn* is any column 12-42. If the statement requires more than one line, all lines after the first will either align with column 12 or the beginning of the statement or may be indented. Refer to the **DI** parameter for more information. Use **NODT** to produce the statement without this formatting. An example of each is shown below.

FORMAT=(DT31)

```
77 ITEM-A                PIC X VALUE 'A'.
77 ITEM-B                PIC 9.
77 ITEM-CHARLIE PIC X(20).
```

FORMAT=(NODT)

```
77 ITEM-A PIC X VALUE 'A'.
77 ITEM-B PIC 9.
77 ITEM-CHARLIE PIC X(20).
```

DTT*nn* indents the first clause of each data definition to column *nn* where *nn* is any column 12-42. If the statement requires more than one line, all lines after the first will also be indented. Use **NODT** to produce the statement without this formatting. An example of each is shown below.

FORMAT=(NODT)

```
77 ITEM-A PIC X VALUE 'A'.
77 ITEM-B PIC 9.
77 ITEM-CHARLIE PIC X(20).
```

FORMAT=(DTT31)

```
77 ITEM-A          PIC S9(5)V99 USAGE IS COMP-3
                   VALUE +10000.00.
77 ITEM-B          PIC 9.
77 ITEM-CHARLIE    PIC X(20).
```

DI*a***X***b* Each data definition subordinate to a definition beginning in column 12 is indented "a" columns for "b" levels where "a" is 1-6, and "b" is 1-6. Use **NODI** to format all statements in column 12.

FORMAT=(DI2X3)

```
01  GROUP-LEVEL-1.
   02 GROUP-LEVEL-2.
      05 GROUP-LEVEL-5.
         10 GROUP-LEVEL-10.
            15 GROUP-LEVEL-15.
            20 ELEMENTARY-ITEM-20 PIC X.
```

FORMAT=(NODI)

```
01  GROUP-LEVEL-1.
   02 GROUP-LEVEL-2.
   05 GROUP-LEVEL-5.
   10 GROUP-LEVEL-10.
   15 GROUP-LEVEL-15.
   20 ELEMENTARY-ITEM-20 PIC X.
```

C.1.4 Procedure Division

PC begins a new line at column 12 for the sentence following a procedure header, paragraph, or section name. Use **NOPC** to keep the sentence on the same line.

FORMAT=(PC)

```
ACCUMULATE-AMOUNTS.
    ADD ABLE TO BAKER. GO TO CHARLIE.
```

FORMAT=(NOPC)

```
ACCUMULATE-AMOUNTS. ADD ABLE TO BAKER. GO TO CHARLIE.
```

PP begins a new line at column 12 for the sentence after any period, except the period ending a procedure header. Use **NOPP** to keep the sentence on the same line.

FORMAT=(PP)

```
ADD ABLE TO BAKER.
MOVE CHARLIE TO DOG-HOUSE.
GO TO SAM.
```

FORMAT=(NOPP)

```
ADD ABLE TO BAKER. MOVE CHARLIE TO DOG-HOUSE. GO TO
SAM.
```

PIaXb Specifies two types of indentation:

For statements requiring two or more lines, each line after the first is indented "a" columns where a= 1-6. For subordinate statements, each statement subordinate to a conditional statement or a "node" (i.e., begins with ELSE, WHEN, etc.) of a conditional statement is indented "a" columns. If the subordinate statement requires two or more lines, each line after the first is indented "a" columns.

This type of indentation continues for "b" levels where b = 1-6. Both the right and left margins are indented when conditional statements are nested.

Use NOPI to format all statements in column 12.

FORMAT=(PI2X1)

```
MOVE CHARLIE OF DOG-HOUSE (BAD) TO CHARLIE OF
   PEOPLE-HOUSE (GOOD).
```

FORMAT=(NOPI)

```
MOVE CHARLIE OF DOG-HOUSE (BAD) TO CHARLIE OF
PEOPLE-HOUSE (GOOD).
```

PV*x...x* There are 9 **PV** subparameters (A through I) that can be used in various combinations to determine how Procedure Division statements are formatted. The "x" in PVx...x represents one or more of these parameters.

If not specified, COBOL verbs have no effect upon line spacing. Use **NOPV** to format all statements in columns 12-72. Use PV to override both **NOPC** and **NOPP**.

A The COBOL verb IF and its ELSE or OTHERWISE clause always start a new line. Other statements do not begin on a new line if the statement can be placed on the line beginning with IF, ELSE, or OTHERWISE. The ELSE or OTHERWISE clauses and long conditioned statements are indented when the **PI** option is specified.

When **PVA** is specified, all other **PV** options are ignored.

FORMAT=(PVA,PI2X1)

```
IF ABLE = BAKER MOVE SAM TO DOG
   ELSE MOVE SAM TO EASY MOVE CHARLIE TO DOG-HOUSE.
```

B All COBOL verbs and any word assigned the verb attribute with a verb macro start a new line. However, the **F** and **G** subparameters described below are exceptions.

If any or all of the PV codes C, D, and E are specified, PVB must also be specified.

FORMAT=(PVB,PI2X3)

```
IF ABLE = BAKER
MOVE SAM TO DOG ELSE
MOVE SAM TO EASY
MOVE CHARLIE TO DOG-HOUSE.
```

Note: ELSE and OTHERWISE are not verbs; they are an optional clause of the IF verb. Another name for this type of clause is "conditional node." In SEARCH and EVALUATE verbs, AT END and WHEN are clauses.

The PI option will not indent conditional statements but will indent long sentences.

The PVB option only formats verbs and is insensitive to verb type and clauses.

C The COBOL verbs IF, SEARCH, and EVALUATE and their clauses ELSE, OTHERWISE, AT END, and WHEN always start a new line. If the PI option has been specified, these verbs and clauses will be indented when they are embedded within a conditional sentence. The C subparameter is similar to the A subparameter; however, in the A subparameter, conditioned statements do not start a new line. For the C subparameter to take effect, the B subparameter must also be specified.

Refer to the H subparameter for an alternative alignment for the clauses.

FORMAT=(PVBC,PI2X3)

```
IF BLACK
   IF ROUND
      IF MINERAL
         DISPLAY 'BRM'
         ELSE
            DISPLAY 'BRX'
      ELSE
         DISPLAY 'BXX'
   ELSE
      DISPLAY 'XXX'.
```

D The logical connectives AND and OR always begin a new line. The B subparameter must also be specified for the D subparameter to take effect.

FORMAT=(PVBCD,PI2X3)

```
IF ABLE = BAKER
   AND CHARLIE = DOG
   GO TO JOE.
```

E The optional standard condition clauses of imperative statements with an alternate conditional form--AT END, ON SIZE, and INVALID KEY, etc.--begin a new line. The B subparameter must be specified for the E subparameter to take effect.

FORMAT=(PVBCE,PI2X3)

```
    READ MASTER-FILE
       AT END
       GO TO WINDUP.
```

F The COBOL verb following an ELSE or OTHERWISE does not begin a new line, even if the B subparameter is specified.

FORMAT=(PVBCF,PI2X3)

```
 IF ABLE = BAKER
    MOVE SAM TO DOG
    ELSE MOVE SAM TO EASY.
```

G The COBOL verb GO does not begin a new line within a conditional, even if the B subparameter is specified.

FORMAT=(PVBCG,PI2X3)

```
 IF ABLE = BAKER
    MOVE SAM TO DOG GO TO CHARLIE.
```

H This subparameter alters the alignment of the IF, SEARCH, and EVALUATE verb clauses when the B and C subparameters are also specified. The clauses are aligned with their corresponding verbs. The H subparameter does not affect indentation caused by any other conditioned statements or their clauses.

FORMAT=(PVBCH,PI2X3)

```
 IF BLACK
    IF ROUND
       IF MINERAL
          DISPLAY 'BRM'
       ELSE
          DISPLAY 'BRX'
    ELSE
       DISPLAY 'BXX'
 ELSE
    DISPLAY 'XXX'.
```

I Using the verb IF after ELSE does not begin a new line. The B, C, and H subparameters must also be specified for the I subparameter to take effect.

FORMAT=(PVBCHI,PI2X3)

```
IF BLACK
   DISPLAY 'BLACK'
ELSE IF ROUND
   DISPLAY 'ROUND'
ELSE IF MINERAL
   DISPLAY 'MINERAL'
```

C.2 Translate-Time Overrides

FORMAT option parameters specified at installation can be overridden. For example, if FORMAT=ADR is specified at installation, the IC parameter can be added to the format list by specifying FORMAT=(IC) on an OPTION card.

FORMAT parameters can be cancelled by prefixing the parameter with NO. For example, if FORMAT=(DT31) was specified at installation, the value 31 could be overridden by specifying FORMAT=(NODT).

Numeric values can be changed for a translation by specifying a new value. For example, if FORMAT=(DT31) was specified at installation, the value 31 could be overridden by specifying FORMAT=(DT40) at translation.

The list of PV subparameters can be extended or decreased for a translation by specifying a new list. For example, if PVA was specified at installation and PVBC is desired, specify FORMAT=(PVBC). IF PVBC is specified and PVB is desired, specify FORMAT=(PVB).

The following example illustrates how the FORMAT= options are specified in the OPTION card.

```
OPTION FORMAT=(DT40,PVB)
      *$COPY ...
       IDENTIFICATION DIVISION.
```

Refer to Section 6.3, "How to Run a Translation," for a description of how to specify the OPTION cards in MVS and VSE.

C.3 Predefined Format Alternatives

The table below lists the FORMAT option parameters assigned to the predefined FORMAT alternatives established at installation.

FORMAT	Predefined FORMAT Alternatives		
Parameters	FORMAT=ADR	FORMAT=OPEN	FORMAT=CLOSED
IC	NOIC	IC	NOIC
EC	NOEC	EC	NOEC
EV	EV	EV	NOEV
ES	NOES	ESC31	NOES
EI	EI2	EI2	NOEI
DF	NODF	DFC31	NODF
DT	DT31	DT31	NODT
DI	DI2X3	DI2X5	NODI
PC	PC	PC	NOPC
PP	PP	PP	NOPP
PI	PI2X1	PI2X4	NOPI
PV	PVA	PVBCDEH	NOPV

Appendix D. Diagnostics

The CA-MetaCOBOL+ Translator generates the following types of diagnostics:

FATAL ERROR	immediately terminate all translation. A condition code of 16 is generated.
ERROR	diagnose erroneous conditions that may produce some unpredictable results but are not so severe as to immediately terminate translation. A condition code of 12 is generated.
WARNING	diagnose erroneous conditions where a corrective assumption is applied. A condition code of 8 is generated.
NOTE and **FLAG**	programmed into macro procedures by Computer Associates and/or users. Their severity is determined in context. The default condition code is 4.

A93 **SYSTEM ERROR** (FATAL ERROR)

Explanation: The Translator attempts to execute a function not supported in this system. This is caused by an internal Translator error.

Action: Report this error to your technical representative with supporting documentation.

B01 **GETMAIN/GETVIS ERROR** (FATAL ERROR)

Explanation: (MVS and VSE only) Occurs during macro loading translation. Table space or addressing capacity is exceeded on TSY, TMS, or TDS. Probable causes:

- Source Data Division or macro definitions exceed addressable table capacity.
- No COBOL division header precedes the source, causing TMS to overflow.

Action: Verify the presence of a COBOL division header. If a header is not present, include the appropriate header before the first source statement. If a division header exists, a table has exhausted its addressing capability. If running in a small partition or region, increase core size or retry. Otherwise, review the utilization of table space provided on the STATISTICS listing.

B02 **SYS002 MAXIMUM EXTENTS EXCEEDED** (FATAL ERROR)

Explanation: (VSE only) Occurs during translation. The Out-of-Line Work File (SYS002) exceeds the maximum number of extents provided by the installation parameter OXTNT.

Action: Either reinstall CA-MetaCOBOL+ with a greater OXTNT operand or reallocate the data set, increasing the size of each extent specified.

B03 **INPUT RECORD LENGTH ERROR** (FATAL ERROR)

Explanation: (VSE only) Occurs during translation. The length of the first SYSRDR or SYSIPT block is other than 80 or 81 characters, or the length of a subsequent block is different than the length of the first block on that file.

Action: Correct the input file and rerun.

B04 **INPUT ERROR - CODE** *n* **IN** *file* (FATAL ERROR)

Explanation: (CMS only) The *file* is the input file-name currently under process, and *n* is the error code defined for an FSREAD.

Action: Check the CMS appendix in the appropriate manual for the meaning of *n* and correct as indicated.

B05 **WORK FILE ERROR - CODE** *n* (FATAL ERROR)

Explanation: (CMS only) A system error occurred in writing to MCTFM or MCTFE, where *n* is the error code defined for an FSWRITE.

Action: Check the CMS appendix in the appropriate manual for the meaning of *n* and correct as indicated.

B06 **LISTING ERROR - CODE** *n* (FATAL ERROR)

Explanation: (CMS only) *n* is the error code defined for an FSWRITE. (This diagnostic is written to the terminal only, regardless of the TERM= option.)

Action: Refer to the appropriate CMS manual to determine the meaning of *n.*

B07 **WORK FILE {OOL|INL} AT MAXIMUM** (FATAL ERROR)

Explanation: Occurs during translation. A work file (INL indicates the in-line file, OOL indicates the out-of-line file) has reached its maximum logical capacity. The Translator terminates.

Action: The probable cause is an output loop from a macro call; examine the input program and macro listing for such a possibility. If no loop can be found, increase the block size for the work file to maximum and rerun the Translator.

B08 **FREEMAIN/FREEVIS ERROR** (FATAL ERROR)

Explanation: Occurs when CA-MetaCOBOL+ de-allocates memory that has already been de-allocated.

Action: Attempt to re-execute MetaCOBOL. Contact your ADR support representative if CA-MetaCOBOL+ still terminates with this error.

C01 **INVALID CONTROL PUNCH IN COLUMN 7** (ERROR)

Explanation: Occurs during macro loading. The continuation column contains a control punch other than a space, *, -, /, L, P, S, T, U, V, W, or X.

Action: Correct the statement causing the error.

C02 **INVALID DIVISION CODE** *word* (ERROR)

Explanation: Occurs during macro loading and following recognition of a macro type code (P, S, V, U, or W) in the continuation column. An Area A word contains characters other than I, E, D, or P. Probable causes:

- Division codes(s) are not separated from macro names by one or more spaces.
- Division code(s) are not intended and macro name begins in Area A.

Action: Ensure that division codes are contiguous, begin in column 8, and are followed by one or more spaces. If no division codes are specified (the macro applies to all COBOL divisions), the macro name must begin in Area B.

C03 **MISSING COLON IN MACRO DEFINITION** (WARNING)

Explanation: Occurs during macro loading. No colon follows a Word or Prefix macro name. A colon is assumed after the first word of a Word or Prefix definition.

Probable causes:

- Colon is omitted.
- Division code appears in Area B and is interpreted as macro name.
- Prototype appears in Area A and is interpreted as division code. This diagnostic is often displayed in conjunction with others.

Action: Include the colon at the location intended or correct division code/macro name area alignment.

C05 MISSING COLON IN A PRIOR STRING (ERROR)

Explanation: Occurs during macro loading. A String macro has no colon separator between the prototype and model. The diagnostic may appear several lines below the actual error, or immediately following the next macro definition or first COBOL division header. It also may appear in conjunction with other diagnostics.

Action: Separate the prototype from the model by inserting the colon at the proper location.

C06 INVALID OPERAND MODIFIER IN MACRO DEFINITION (ERROR)

Explanation: Occurs during macro loading. A String macro contains an invalid symbolic operand modifier. Probable causes:

- Symbolic operand in macro prototype contains code other than L, Q, or S (or a combination of these separated by commas) as a parenthetical expression following the symbolic operand number.
- Symbolic operand modifier specified in a macro model.

Action: Specify only the codes L, Q, and/or S as symbolic operand modifiers in String macro prototypes.

C07 TOO MANY $DDX OR $PDX (ERROR)

Explanation: Occurs during macro loading. More than nine $DDX or $PDX macros are encountered. Only the first nine are active; the rest are ignored.

Action: Limit $DDX and $PDX macros to nine each.

C08 DUPLICATE $-PROC, $-VERB, OR $-LEVEL (ADVISORY)

Explanation: Occurs during macro loading. The event-dependent macros $-PROC, $-VERB, or $-LEVEL override all macros of the same name that have been previously loaded.

Action: If the override is unintentional, review the uses of the macro(s) in question. Combine macro logic into a single macro, if necessary.

C09 OCCURRENCE ILLEGAL ON BOOLEAN VARIABLE (ERROR)

Explanation: Occurs during macro loading. Definition of or reference to a Boolean variable includes occurrence notation. Multiple occurrences of Boolean variables are not supported.

Action: Define conventional variables as switches if a table of variables is required and/or remove the occurrence number from the Boolean variable.

C10 VARIABLE NAME TOO LONG (WARNING)

Explanation: Occurs during macro loading. A variable name exceeds 30 alphanumeric characters in length, inclusive of the beginning characters &V.

Action: Shorten the variable name to 30 characters or less.

C11 DUPLICATE TAG OR LABEL DEFINITION *word* (WARNING)

Explanation: Occurs during macro loading. Either a macro model contains identical tag definitions or the macro set contains identical label definitions. Probable causes:

- Tag or label is not unique in context.
- Prior errors occurred, causing the Translator to interpret reference to tag or label as definition.

Action: Alter the tag name so that it is unique within the macro, alter the label name so that it is unique within the region, or correct the condition which caused the erroneous interpretation.

C12 UNVERB NAME NOT FOUND *word* (ERROR)

Explanation: Occurs during macro loading. An Unverb macro defines *word* to be removed from the verb list. The *word* is unmatched within the verb list. The macro is ignored. The following are the probable causes of this error.

- Specified *word* is not defined as verb.
- The *word* is not spelled correctly.

Action: Review the CA-MetaCOBOL+ reserved word list in Appendix B, "Reserved Words," for the word or correct the spelling as appropriate.

C13 ILLEGAL USE OF S-TYPE VARIABLE (ERROR)

Explanation: Occurs during macro loading. A symbolic operand storage variable is referenced in a directive expression other than an &EQU or Format 1 of a $SET. The directive expression is not executed.

Action: Review definitions of S-type variables and all associated references to ensure proper usage.

C14 IMPROPER USE OF &(or &) (ERROR)

Explanation: Occurs during macro loading. A concatenation expression in a macro model does not pair the open and close symbols. Probable causes:

- Open or close symbol is missing or incorrectly specified.
- Attempt to nest concatenation is invalid.
- Open or close concatenation is incorrectly specified. The missing close symbol &) is often diagnosed several lines below its intended location. The missing &) can cause other diagnostics.

Action: Add the missing open/close symbol, remove nested symbols, or correct the erroneous symbol.

C15 VARIABLE OCCURRENCE INVALID *word* (ERROR)

Explanation: Occurs during macro loading. The multiple occurrence value, in parentheses, following the variable name contains non-numeric characters. Only integers can be specified.

Action: Using an integer, specify the proper number of occurrences for the variable.

C16 ILLEGAL USE OF DIRECTIVE *word* (ERROR)

Explanation: Occurs during macro loading. Probable causes:

- Directive expression is not syntactically correct or complete.
- Directive is specified within another directive expression.
- Directive is used in macro type that does not support it, such as &DO or &GET in Word or Prefix macro models. This diagnostic is often displayed with others.

Action: Review the proper use of directive expressions and make appropriate corrections.

C17 UNDEFINED DIRECTIVE (ERROR)

Explanation: Occurs during macro loading. A macro model contains a word beginning with an ampersand (&). The word is not a tag, label, variable name, symbolic operand, or directive. Probable causes:

- Directive is misspelled.
- Invalid specification of word beginning with &.

Action: Correct the spelling of the directive, tag, label, variable, or symbolic operand, or remove the ampersand as a leading character.

C18 ILLEGAL ATTRIBUTE CODE *word* (ERROR)

Explanation: Occurs during macro loading. A macro model contains an attribute code which is not valid (i.e., not defined).

Action: Review the list of valid attribute codes and correct the erroneous code with the appropriate symbol.

C19 UNDEFINED VARIABLE *word* (ERROR)

Explanation: Occurs during macro loading. A variable name is referenced that was not previously defined as an external, global, or local variable.

Action: Define the variable before all references to it.

C20 ILLEGAL WORD IN PROTOTYPE *word* (ERROR)

Explanation: Occurs during macro loading. A macro prototype contains a word beginning with &T, &L, or &V, or contains a double or single quotation mark. Only constants and/or symbolic operands can be specified within a prototype. Also, the leading ampersand can only precede the integers 1-15 within String macro prototypes.

Action: Review the rules for macro prototype specification and correct the macro.

C21 & OR &0 USED IN PROTOTYPE (ERROR)

Explanation: Occurs during macro loading. The symbolic operand &0 is referenced in a String macro prototype. &0 is reserved for special use by &SCAN-type directive expressions and cannot be specified in the prototype.

Action: Replace &0 with any symbolic operand from &1 through &15.

C22 IMPROPER INITIALIZATION FOR VARIABLE (WARNING)

Explanation: Occurs during macro loading. A variable is encountered with an initial value that is not a literal or does not conform to the class of the variable (e.g., NULL can be specified only for non-numeric variables). The initial value defaults to a space or zero.

Action: Correct the variable definition causing the error.

C23 INVALID MACRO NAME *word* (ERROR)

Explanation: Occurs during macro loading. A macro name exceeds 30 characters, begins with an ampersand, takes the form of a numeric or non-numeric literal, or contains invalid characters.

Action: Review the rules for macro name formation and correct the macro name.

C24 ILLEGAL VARIABLE DEFINITION *word* (ERROR)

Explanation: Occurs during macro loading. An &EXTERN, &GLOBAL, or &LOCAL variable definition contains a picture which does not take the form X...X; X(n); 9...9; 9(n); or S. All valid characters up to the first invalid character in the definition are used as the assumed definition. If no valid characters are found, a definition of X is assumed.

Action: Correct the picture within the variable definition directive expression.

C25 NON-COMPATIBLE DUPLICATE DEFINITION *word* (WARNING)

Explanation: Occurs during macro loading. A variable defined by an &EXTERN, &GLOBAL, or &LOCAL directive expression has the same name as a previously defined variable but does not have the same picture or multiple occurrence value. The original definition is retained.

Action: Make sure that unique names are used in defining all external variables to be used during translation, all global variables within a region, and all local variables within each macro.

C26 VARIABLE DEFINITION TOO LONG (WARNING)

Explanation: A variable is defined which exceeds the maximum number of allowable characters. Numeric variables cannot exceed nine digits. Non-numeric variables cannot exceed 128 characters. The variable definition is truncated to the allowable maximum.

Action: Redefine the variable so that it does not exceed the allowable maximum.

C27 **UNMATCHED** *&word* (ERROR)

Explanation: Occurs during macro loading. &word is a directive associated with a directive construct that is not matched by the construct initiator. An example is an &ENDIF unmatched by a preceding &IF.

Action: Review the rules for directive constructs and correct the macro.

C28 **UNTERMINATED** *&word* (ERROR)

Explanation: Occurs during macro loading. &word is a directive construct initiator that is not matched by the appropriate terminator. For example, an &IF that is unmatched to an &ENDIF.

Action: Review the rules for directive constructs and correct the macro.

C29 **MIXED &AND**, **&OR** (ERROR)

Explanation: Occurs during macro loading. Complex conditional contains intermixed &AND and &OR operators.

Action: Correct the complex conditional so that only &AND or &OR are used.

C30 **STRUCTURE STACK OVERFLOW** (ERROR)

Explanation: Occurs during macro loading. Directive constructs of all types are nested beyond the capacity of the structure stack.

Action: Eliminate excessive levels of nesting by means of subordinate subroutines invoked by the &DO directive.

C31 **ILLEGAL USE OF BOOLEAN VAR** (ERROR)

Explanation: Occurs during macro loading. A Boolean variable is referenced incorrectly.

Action: Review the rules for Boolean variable definition and reference, and correct the error.

C32 **FOLLOWING TAGS ARE UNDEFINED** (ERROR)

Explanation: Occurs when a macro is loaded and one or more local tags are referenced but are not defined.

Action: Define the tags.

C33 FOLLOWING LABELS ARE UNDEFINED (ERROR)

Explanation: Occurs when an entire region has been loaded or at the end of macro loading. One or more global labels are referenced but are not defined.

Action: Define the labels.

C34 REPLACE STACK OVERFLOW (ERROR)

Explanation: The table for the REPLACING clause of the COPY statement is not large enough.

Action: Specify a larger size (in bytes) for the CRSTACK option and re-run the translation.

C35 TOO MANY *$REGION (ERROR)

Explanation: Occurs during macro loading. More than nine *$REGION Translator Directing statements are encountered. Action is taken on the first nine only; the rest are ignored.

Action: Limit *$REGION statements to nine.

C36 "NOT" IS INVALID IN THIS CONTEXT (ERROR)

Explanation: "NOT" was used with Format-1 of the &SELECT construct, where negation is not valid.

Action: Remove negation from the WHEN clause(s).

C39 TOO MANY NESTED SELECTS (ERROR)

Explanation: The maximum level of 50 nested &SELECTs was exceeded.

Action: Reduce the levels of nested &SELECTs.

C93 MACRO DIVISION ONLY (FATAL ERROR)

Explanation: Occurs during macro loading. No COBOL division header or standard division header abbreviation is in Area A of the source. This error also occurs when the COBOL program is being included by a *$COPY, *$LIBED, or *$LIBET which does not begin in column 7. CA-MetaCOBOL+ terminates immediately.

Action: For translation to occur, source must be preceded by a COBOL division header.

C94 LOCAL TABLE OVERFLOW (FATAL ERROR)

Explanation: Occurs during macro loading. A macro model contains more local variables and tag definitions than can be resolved and/or too many Prefix macros are defined. CA-MetaCOBOL+ terminates. The total number of tags and local variables which can be resolved per macro is approximately 120, in any combination, assuming that no Prefix macro names are stored.

Action: Do any combination of the following:

- Place all Prefix macro definitions at end of macro set.
- Define local variables as global variables.
- Define tags as labels.
- Place some macro logic in executable models of another macro, referred to by label name or through nested dummy Word macros.

F02 WORD TOO LONG *word* (ERROR)

Explanation: Occurs during translation. A macro attempts to substitute a word which exceeds 30 characters in length. The substituted word is truncated to 30 characters, and the macro continues to execute.

Action: Examine the output to verify the truncated word. Modify the macro to ensure generation of a valid word.

```
                                          ⎧ ⎡PRIVATE⎤ ⎡DIRECTORY ⎤ ⎫
                                          ⎪ ⎣PUBLIC ⎦ ⎣membername⎦ ⎪
G01   SYSLIB I/O ERROR - COPY LIBRARY     ⎨   LIB n membername     ⎬
                                          ⎪                        ⎪
                                          ⎩          (FATAL ERROR) ⎭
```

Explanation: Occurs during macro loading and translation. Source statement library resulted in an I/O error.

For DOS/VSE Release 1,
PRIVATE/PUBLIC identifies the library causing the error (the LIB *n* format does not occur).

DIRECTORY indicates that the error is doing a member search in that portion of the library. The public library is searched, if appropriate, or a "NOT FOUND" message is generated and an H*nn* diagnostic follows.

membername indicates that the member is in error; CA-MetaCOBOL+ terminates. The absence of either DIRECTORY or *membername* indicates that the error is in the system control area for the library. The library is not accessed for the remainder of the Basic Processor execution.

For DOS/VSE Release 2 and later releases,
LIB *n* identifies the level of the library causing the error (the PRIVATE/PUBLIC format does not occur). LIB 1 is the highest LIBDEF SL SEARCH level of the private library defined with an ASSGN. The value increases with succeeding lower level libraries.

membername indicates that the member is in error; CA-MetaCOBOL+ terminates.

Action: Use a system utility to examine the library/member in error. Make the necessary corrections.

G12 OPEN OR FIND ERROR - COPY LIBRARY (ERROR)

Explanation: The member requested cannot be found or the DD statement specified is not present.

Action: Correct the COPY statement or provide the correct DD statement and re-run.

G13 COPY RECURSIVE DDN={*member-name*|*DDname*} (ERROR)

Explanation: A nested COPY cannot recursively copy input.

Action: Correct the nested COPY and re-run.

G14 ALLOWED COPY NESTING LEVEL EXCEEDED (ERROR)

Explanation: The limit (256) for nested COPY statements has been exceeded.

Action: Correct the nested COPY statement and re-run.

G15 COPY I/O ERROR DDN={*member-name*|*DDname*} (ERROR)

Explanation: An I/O error has occurred reading from the specified member or data set.

Action: Correct the data set or member name in error and re-run.

H01 QUOTE LITERAL LONGER THAN 130 *word* (ERROR)

Explanation: Occurs during translation. A non-numeric literal is found in the input which exceeds 130 characters in length, including enclosing quotes. The literal is truncated to 128 characters, plus the enclosing quotes.

Action: IBM COBOL dialects limit non-numeric literals to 120 characters. Other COBOLs support longer literals. The literal must be altered.

H02 QUOTE LITERAL IMPROPERLY CONTINUED *word* (ERROR)

Explanation: Occurs during translation. A non-numeric literal does not end within the current input record, but the continuation column of the subsequent record does not contain a continuation code and/or a quote symbol as the first non-space character in Area B. The literal is truncated at the end of the last line before required continuation.

Action: Properly continue the source statement record containing the continued literal. (IBM dialects D, E, and F do not support continuation for words other than non-numeric literals, and ignore continuation conventions. Refer to the IGNORE Translator Option for the solution to this problem.)

H03 WORD LONGER THAN 30 BYTES *word* (ERROR)

Explanation: Occurs during translation. A source word exceeds 30 characters in length. The word is truncated to 30 characters.

Action: Review the SEPPAR/NOSEPPAR Translator Option. Define source words as 30 characters or less. This does not apply to non-numeric literals.

H04 COPY LIBRARY NOT FOUND [*module-name*] (ERROR)

Explanation: Occurs during translation. The module-name following a COBOL COPY or defined for an &COPY directive cannot be located in the source statement library -- or the library is not defined by job control statements. The *module-name* is provided in the diagnostic if the library request is from an &COPY directive.

Action: Define the source statement library with job control statements or correct *module-name*, the COBOL COPY statement, or the &COPY directive.

H05 FORMAT LEGAL ONLY IN MACRO DIVISION (ERROR)

Explanation: Occurs during translation. The source contains items which are legal only within macro definitions. The items are output unaltered.

Action: Do not specify any CA-MetaCOBOL+ terms or words after the first COBOL division header. Conversely, do not specify a COBOL division header within macro definitions.

H06 INVALID COPY STATEMENT (ERROR)

Explanation: Occurs during translation. The syntax of the COBOL COPY statement is invalid. The COPY statement is ignored.

Action: The COPY statement must be corrected to conform to IBM specifications, although a particular COBOL compiler may accept other undocumented formats.

H07 COPY STACK OVERFLOW (FATAL ERROR)

Explanation: Occurs during translation WHEN either a COPY statement or COPY text is encountered. If the error is on a COPY statement, the REPLACING function of the COPY statement exceeds its table space; therefore, no library text is processed. If the error is on COPY text, a COBOL comment or Area A indicator exceeds its table space; therefore, the comment or Area A indicator is dropped.

Action: Allocate additional REPLACING table space by the CRSTACK= Translator Option.

H08 ERROR *n* **ON FAIR** (ERROR)

Explanation: An I/O error has occurred reading from CA LIBRARIAN. *n* is a CA LIBRARIAN return code.

Action: Refer to the CA LIBRARIAN FAIR (File Access Interface Routine) documentation for the meaning and corrective action for *n*. Correct the error and re-run.

H09 ERROR *n* **ON** CA LIBRARIAN **READ** (FATAL ERROR)

Explanation: Occurs during macro loading/translation. An error occurs during a READ of an open CA LIBRARIAN file. The diagnostic follows a *$LIBE statement. The remainder of The CA LIBRARIAN module is ignored. If *n* = 2, CA-MetaCOBOL+ reads past the end of The CA LIBRARIAN module. If *n* = 9, an illogical condition (such as missing control records, index pointer errors, etc.) is found on the master file.

Action: Rerun to ensure that the error is not the result of a transient hardware failure. If the problem persists, contact your technical representative with supporting documentation.

H10 MODULE IS SOFTWARE LOCKED (ERROR)

Explanation: Occurs during macro loading/translation. The module named by the *$LIBED or *$LIBET statement is secured from reading by CA LIBRARIAN. The *$LIBE statement is ignored.

Action: Obtain the module for translation in a separate CA LIBRARIAN job step.

H11 INVALID DBCS LITERAL *(ERROR)*

Explanation: A double-byte character set (DBCS) literal is coded incorrectly.

Action: Correct the literal and re-run.

H12 **ERROR** *n* **ON** CA LIBRARIAN **OPEN** (FATAL ERROR)

Explanation: Occurs during macro loading/translation. An error occurs during an OPEN of a CA LIBRARIAN file. The diagnostic follows a *$LIBE statement. The CA LIBRARIAN copy is ignored.

If *n* = 1, the CA LIBRARIAN master or tape cycle control file is not defined by job control statements; or the CA LIBRARIAN master file may reside on an unsupported device.

If *n* = 2, the specified file does not conform to CA LIBRARIAN specifications because: 1) the master file assignments are incorrect, 2) the master or cycle file has been destroyed, or 3) the wrong tape master is mounted.

If *n* = 3, an error occurred before the end of the previous CA LIBRARIAN module copy CLOSEd.

If *n* = 4, the buffer size specified at installation of CA-MetaCOBOL+ or by the LCORE= Translator Option is too small.

If *n* = 9, an illogical condition (such as missing control records, index pointer errors, etc.) is found on the master file, or JCL is missing or incorrect.

Action: If *n* = 4, specify the proper buffer size by the LCORE Translator Option. In other cases, where the error cannot be determined, attempt to rerun to ensure that the error is not the result of a transient hardware failure. If the problem persists, contact your ADR support representative with supporting documentation.

H13 **MEMBER NOT FOUND - COPY BYPASSED** (ERROR)

Explanation: Occurs during macro loading/translation. The module requested by a *$LIBE statement cannot be found in a CA LIBRARIAN master file, or the module requested by the *$COPY statement cannot be found in the source statement library.

Action: Request the proper module-name.

H14 **NESTED COPY-REPLACING NOT ALLOWED** (ERROR)

Action: A nested COPY has been detected and the DIALECT Translator Option is not set to XO.

Explanation: Correct the COPY or DIALECT and re-run.

H16 LIBRARIAN NOT SUPPORTED (ERROR)

Explanation: Occurs during macro loading/translation. A *$LIBE statement is encountered on a CA-MetaCOBOL+ system which does not support CA LIBRARIAN. Request ignored.

Action: Remove *$LIBE statements or reinstall CA-MetaCOBOL+ to include CA LIBRARIAN support.

H17 INVALID *$LIBE/*$COPY STATEMENT (ERROR)

Explanation: Occurs during macro loading/translation. The syntax of a *$LIBE/*$COPY statement is incorrect.

Action: Correct the *$LIBE/*$COPY statement causing the error.

H18 INVALID -INC (WARNING)

Explanation: Occurs during macro loading/translation. Input obtained from a CA LIBRARIAN disk master contains the indicated -INC statement. The member does not exist or the statement appears at a nesting level greater than that supported by CA LIBRARIAN. The statement is ignored.

Action: Review the use of -INC to ensure proper usage.

H19 INVALID CA LIBRARIAN FUNCTION (WARNING)

Explanation: Occurs during macro loading/translation. Input obtained by a *$LIBET from a CA LIBRARIAN tape master contains the indicated -INC statement. -INC expansion for tape master is not supported by CA LIBRARIAN. The statement is ignored.

Action: -INC is incompatible with tape master files.

H21 INVALID HEX LITERAL (ERROR)

Explanation: Occurs during macro loading/translation. The definition of a hexadecimal literal is incorrect for one or more of the following reasons:

- Not bounded by pairs of quotes
- First quote not preceded by space or margin
- Last quote not followed by space, margin, or terminating punctuation.

Action: Correct the hex literal causing the error.

H22 LINE INVALID AFTER CBL STATEMENT (ERROR)

Explanation: The designated record is invalid following a COBOL statement and preceding a COBOL division header. It is ignored.

Action: Remove the designated statement from the input.

H23 INVALID *$LIBE PASSWORD (ERROR)

Explanation: Occurs during macro loading and translation. CA-MetaCOBOL+ is installed with CA LIBRARIAN password protection but the password is missing or incorrect in the *$LIBE statement. The *$LIBE statement is ignored.

Action: Provide the proper password on the *$LIBE statement.

H24 INVALID CONTENTS IN CONTINUATION COLUMN (ERROR)

Explanation: Occurs during macro loading and translation. CA-MetaCOBOL+ has encountered an invalid character (or bit configuration) in column 7. The input record is ignored.

Action: Correct the continuation column so that it contains a space, *, -, /, or uppercase alphabetic or numeric character.

H25 INVALID LIMIT FOR *$COL (ERROR)

Explanation: Occurs during macro loading and translation. The integer value for a *$COL statement is non-numeric or is a value outside the permitted range. A value of 80 is assumed for the statement in error.

Action: Correct the integer value on the *$COL statement.

H70 USER INPUT EXIT NOT DEFINED (ERROR)

Explanation: An &CALL or *$CALL input exit request has occurred, but no valid IXIT Translator Option has been specified. The request is ignored and Translator execution continues.

Action: Specify the proper module-name with the IXIT Translator Option, remove the *$CALL statement from the input stream, or delete the input code that invokes the &CALL directive.

H71 &CALL WITHIN *$CALL IS ILLEGAL (ERROR)

Explanation: An &CALL input exit request has occurred during execution of a *$CALL. The &CALL request is ignored.

Action: Delete the *$CALL Translator Directing statement or the source words that invoke the &CALL, as appropriate.

H72 INVALID *$CALL STATEMENT (ERROR)

Explanation: A *$CALL Translator Directing statement is missing a character-string operand or the character-string is syntactically incorrect. The *$CALL is ignored.

Action: Correct the *$CALL statement.

H73 INVALID PROCESS CODE FROM INPUT EXIT *code* (FATAL ERROR)

Explanation: The input exit subroutine has returned an undefined value in the second byte of the third parameter. The Translator terminates.

Action: Correct the input exit subroutine.

H78 *text* (FATAL ERROR)

Explanation: The input exit subroutine has returned the indicated fatal error. The Translator terminates.

Action: Specified by *text.*

H79 *text* NOTE

Explanation: The input exit subroutine has returned the advisory diagnostic indicated by *text*. (See diagnostics ADRX010E - ADRX050A below.)

Action: Specified by *text*. The input exit diagnostics begin with the following number:

ADRXnnnb

where:

ADRX	standard character abbreviation for the CA-MetaCOBOL+ input exit subroutine diagnostic.
nnn	unique 1-3 digit diagnostic number.
b	is the severity code:

- **A** is an advisory message.
- **E** is an error message. The source program requires at least one modification.

ADRX010E LOADER TABLE IS FULL.

Explanation: The input exit loader table is full; the maximum limit is ten input exits.

Action: Use the --CANCEL statement to open up the input exit loader table; this statement eliminates a current module of the table.

ADRX020E FAILED DELETING "*module-name*" RETURN-CODE =

Explanation: The input exit loader encountered an error while trying to delete specified module. The return code from the MVS delete SVC is listed.

Action: Either a request to delete was not issued or an attempt was made to delete a protected system module.

ADRX030E TABLE DOES NOT CONTAIN "*module-name*" FOR DELETION.

Explanation: The specified input exit is not loaded, and therefore can not be deleted.

Action: The specified module must be called before it can be deleted.

ADRX040E UNABLE TO LOAD "*module-name*"

Explanation: An error was encountered while trying to load the specified module.

Action: Check the return code for a system error message.

ADRX050A STAE REQUEST FAILED - RC: *return-code*

Explanation: The ADRXIXT Input Exit is unable to establish a STAE environment.

Action: None necessary, processing proceeds without STAE.

H91 READ PAST END OF INPUT (ERROR)

Explanation: Occurs during translation. The Translator attempts to read past the end-of-file marker on the Primary Input File. Probable cause is a logical macro error which acquires input by a closed &GET/&STORE/&STOW loop. CA-MetaCOBOL+ proceeds to the output merge phase.

Action: Determine the closed &GET/&STORE/&STOW loop and insert proper checks for loop termination. Review TRACE=G Translator Option for debugging aids to isolate the &GET/&STORE/&STOW loop, if necessary.

L01 INDETERMINATE USAGE (WARNING)

Explanation: Occurs during translation. An error is detected in a data description. A usage of `'U'` is assumed.

Action: Correct the item causing the error.

L02 NON-INTEGER FOLLOWING OCCURS OR TO (WARNING)

Explanation: Occurs during translation. A data-item definition contains an operand following an OCCURS or TO reserved word that is not an integer. A value of 0 is assumed.

Action: Correct the item causing the error. If in the Procedure Division, ensure the presence of a valid Procedure Division header.

L03 INVALID CD ENTRY (WARNING)

Explanation: Occurs during translation. An error is detected in a clause subordinate to a CD. The clause is ignored.

Action: Correct the item causing the error.

L04 MORE THAN 12 INDEX-NAMES (WARNING)

Explanation: Occurs during translation. More than 12 index-names are encountered in a single data item description. Action is taken on the first 12 only, the rest are ignored.

Action: Limit the number of index-names used to 12, as required by IBM COBOL dialects.

L05 PERIOD MAY BE MISSING IN PREVIOUS ITEM (WARNING)

Explanation: Occurs during translation. A possible level-number is encountered but no period terminates the previous item. Attributes are unpredictable.

Action: Correct the item causing the error.

L06 INVALID LEVEL NUMBER (WARNING)

Explanation: Occurs during translation. An invalid level number is detected. The level number and its associated data description are ignored. Processing continues with the next valid level number or FD.

Action: Correct the item causing the error.

M01 INVALID SUBSCRIPT NOTATION (ERROR)

Explanation: Occurs during translation. The subscripted or indexed current value of a symbolic operand is not terminated by a closed parenthesis. The diagnostic follows the next input period or Area A word. If the symbolic operand is specified in a macro prototype, the macro call fails. If the symbolic operand is acquired by an &STOW directive expression, execution of the macro model is terminated.

Action: Correct the source statement which caused the diagnostic.

M02 INVALID MACRO NESTING *word* (ERROR)

Explanation: Occurs during translation. The substituted "word" exceeds recursive nesting limits or represents invalid nesting for the original macro call. The last valid nesting level is substituted.

Action: Review the rules for valid macro nesting and correct the macro accordingly.

M03 STACK TABLE OVERFLOW (FATAL ERROR)

Explanation: Occurs during translation. The cumulative words of the current values of symbolic operands defined in a String macro prototype cannot be stored during an attempt to call a macro. CA-MetaCOBOL+ terminates.

Action: Reduce the number of source words comprising current values of symbolic operands or reduce the number of Prefix macros loaded. Obtain additional source words by &GET/&STOW/&STORE logic. Increase STACK=nnnn INSTALL OPTION.

M04 SYMBOLIC OPERAND TOO LONG (ERROR)

Explanation: Occurs during translation. The cumulative words of the current value of a symbolic operand cannot be stored when calling a String macro or acquiring the current value by &GET/&STOW directives. The current value is truncated and the macro continues to execute.

Action: Reduce the number of source words comprising the current value of the symbolic operand.

M05 MISSING &DO EXIT (ERROR)

Explanation: Occurs during translation. No &EXIT directive is executed within the model entered by an &DO directive expression. Execution of the macro model(s) terminates.

Action: Determine the performed model which terminated and include the &EXIT directive where logic dictates.

M06 STRING MISMATCH (WARNING)

Explanation: Occurs during translation. A nested Word or Prefix macro satisfying a String macro prototype has substituted a word of type N (`&n'T = 'N'`) before substituting a keyword, data-name, procedure-name, or verb.

Action: See the &SETR NOTE register section in the CA-MetaCOBOL+ *Macro Facility Reference Manual* for further information.

N01 UNDEFINED LABEL OR TAG (ERROR)

Explanation: Occurs during translation. A directive expression transfers control to an undefined tag or label. Execution of the macro model terminates.

Action: Determine the model statement which caused the diagnostic and correct macro logic. The TRACE=D Translator Option may be useful in determining the macro statement in error.

N02 MORE THAN 32 LEVELS OF &DO (ERROR)

Explanation: Occurs during translation. The number of nested &DO directives is exceeded. An immediate exit is taken to the model statement following the last active &DO, and execution continues.

Action: Determine the model statements which caused the diagnostic and correct the macro logic. The TRACE=D Translator Option may be helpful in determining the macro statement in error.

N03 INDEX FOR VARIABLE TABLE LT 1, **USING 1** (ADVISORY)

Explanation: The index for a macro variable is 0; however, 1 has been used in its place.

Action: None required, but a check of the macro logic is recommended.

N04 NON-NUMERIC DATA IN &SET OR &IF (ERROR)

Explanation: Occurs during translation. A directive expression references an item which must be numeric but contains non-numeric characters. Execution of the macro model terminates.

Action: Determine the model statement which caused the diagnostic and correct macro logic. The TRACE=D Translator Option may be helpful in determining the macro statement in error.

N05 INVALID INDEX FOR VARIABLE TABLE (ERROR)

Explanation: Occurs during translation. An indexed variable is referenced using an index which exceeds the defined maximum occurrences. Execution of the macro model terminates.

Action: Determine the model statement which caused the diagnostic and correct the macro logic. The TRACE=D Translator Option may be helpful in determining the macro statement in error.

N06 DATA-NAME UNDEFINED *word* (ERROR)

Explanation: Occurs during translation. A data attribute is referenced before the data item is defined. Execution of the macro model terminates.

Action: Determine the model statement which caused the diagnostic and correct macro logic. Logic may be conditioned by testing for the usage attribute "undefined" (i.e., &n'U = 'U'). The TRACE=D Translator Option may be helpful in determining the macro statement in error.

N07 INVALID &SCAN (ERROR)

Explanation: Occurs during translation for two reasons:

- &SCAN directive expression operands are specified so that the "thru" operand precedes the "from" operand within the Data Division. Execution of the macro model terminates.

- The consecutive execution of a different &SCAN directive before the first has terminated.

Action: Determine the model statement which caused the diagnostic and correct macro logic. The TRACE=D Translator Option may be helpful in determining the macro statement in error.

N08 &0 NOT DEFINED BY &SCAN OR &SCANF (ERROR)

Explanation: Occurs during translation. The symbolic operand &0 is referenced when &SCAN-type directive expression execution is not "open." Execution of the macro model terminates. Probable causes:

- &SCAN-type directive has not executed.

- "At end" branch or &SCANX directive has been executed.

- &0 has been specified rather than &1 through &15.

Action: Determine the model statement which caused the diagnostic and specify a numbered symbolic operand from &1 through &15, or correct the macro logic. The TRACE=D Translator Option may be helpful in determining the macro statement in error.

N09 CONCATENATION TOO LONG (ERROR)

Explanation: Occurs during translation. An operation on a non-numeric literal concatenation exceeding 128 characters, excluding bounding quotes, was attempted.

Action: Determine the model statement which caused the diagnostic and correct the concatenation construction. The TRACE=D Translator Option may be helpful in determining the macro statement in error.

N10 UNDEFINED SYMBOLIC OPERAND (ERROR)

Explanation: Occurs during translation. A symbolic operand containing no logical value is referenced. Execution of the macro model terminates. Probable causes:

- Numbered symbolic operand not defined in String macro prototype; or

- Not defined as object of &GET, &EQU, or &PIC directive expression.

Action: Determine the model statement causing the diagnostic and correct the number of the symbolic operand. The TRACE=D Translator Option may be helpful in determining the macro statement in error.

N11 IMPROPER USE OF &GET, &STOW AND &STORE (ERROR)

Explanation: Occurs during translation. Execution of the macro model terminates. Probable causes:

- Successive logical &GET directive expressions are executed without an intervening &STOW or &STORE in the same macro call.
- Successive logical &STOW and/or &STORE directives are executed without an intervening &GET.
- An &CALL is issued with an outstanding &GET.

Action: Determine the &GET, &STOW, &STORE, or &CALL directive causing the diagnostic and correct the macro logic. The TRACE=D Translator Option may be helpful in determining the macro line in error.

N12 ILLEGAL CONTENTS IN &JUMP VARIABLE (ERROR)

Explanation: Occurs during translation. The variable associated with the &JUMP directive expression contains a value which is less than one or greater than the number of logical destinations referenced in the directive expression. Execution of the macro model terminates.

Action: Determine the &JUMP directive expression causing the diagnostic and ensure valid variable contents. The TRACE=D Translator Option may be helpful in determining the macro statement in error.

N13 ZERO DIVISOR IN &SET (ERROR)

Explanation: Occurs during translation. An &SET directive expression is found where the divisor has a value of zero. The result is set to all 9's.

Action: Review the &SET and modify it so that division by zero cannot occur.

N91 SYSTEM ERROR (ERROR)

Explanation: Occurs during translation. A directive or directive expression cannot be executed. Execution of the current macro terminates.

Action: Report this error to your local technical representative with supporting documentation.

N98 *text* (FLAG)

Explanation: An &FLAG directive has been executed. The text of the message consists of the name of the macro executing the directive.

Action: Review macro set documentation to determine cause and corrective action.

N99 *text* (NOTE)

Explanation: An &NOTE directive has been executed. The text of the message should describe the error that has occurred.

Action: Review macro set documentation to determine cause and corrective action.

P01 **MACRO SET SUCCESSFULLY LOADED** (ADVISORY)

Explanation: Confirms that macros have been unloaded.

Action: None.

P04 **UNABLE TO OPEN UNLOAD MACRO FILE** (FATAL ERROR)

Explanation: A valid UNLOAD file must exist before pre-compiled macros may be unloaded.

Action: Check the translation JCL and make sure it correctly specifies the UNLOAD dataset.

R04 **UNABLE TO** OPEN RELOAD MACRO FILE (WARNING)

Explanation: A valid RELOAD file must exist before pre-compiled macros may be unloaded.

Action: Check the translation JCL and make sure it correctly specifies the RELOAD dataset.

R20 **HEADER RECORD MISSING**
R21 **BLOCK INFO. RECORD MISSING**
R22 **BLOCK HEAD RECORD MISSING**
R23 **ZDATA HEAD RECORD MISSING**
R24 **TRAILER RECORD MISSING** (FATAL ERROR)

Explanation: The RELOAD data set is corrupted.

Action: Recreate the RELOAD data set.

R40 **ADDRESS TRANSLATION ERROR** (FATAL ERROR)

Explanation: A malfunction in the address translation has occurred.

Action: Recreate the RELOAD data set.

T01 **MAXIMUM FOR TYPE ATTAINED** (WARNING)

Explanation: Applicable to MVS TERM= Translator Option only. Does not affect condition code. The maximum number specified for the particular type of display currently going to SYSTERM is reached. All further displays for this type are bypassed. Processing continues.

Action: None.

T02 **MAXIMUM FOR LIMIT ATTAINED** (WARNING)

Explanation: Applicable to MVS TERM=(...LIMIT=n) Translator Option only. Does not affect the condition-code. The maximum number specified for LIMIT is reached. All further displays going to SYSTERM are bypassed.

Action: None.

T03 **VALUE FOR CANCEL EXCEEDED** (FATAL ERROR)

Explanation: Applicable to MVS TERM=(...CANCEL=n) Translator Option only. CANCEL limit is exceeded for displays on SYSTERM. Translation terminates at this point.

Action: To translate further, delete the CANCEL option and rerun; increase the CANCEL limit and rerun; or reduce the number of lines qualifying for display on SYSTERM.

U91 **TOO MANY OUT-OF-LINE MARKERS** (FATAL ERROR)

Explanation: Out-of-line markers are nested more than 20 deep. May be caused by possible loop between &POINT and &MARKER.

Action: Limit the use of nested out-of-line markers so that stack is not more than 20 deep.

X01 **UNDEFINED PARAMETER**: *@word@* **(ERROR)**

Explanation: Occurs during translate-time parameter analysis. An undefined option is specified.

Action: Review the permissible Translator Options and abbreviations and correct the OPTION card or PARM list.

X02 INVALID {FORMAT|TERM} SUB-PARAMETER *@word@* (ERROR)

Explanation: Occurs during translate-time parameter analysis. A FORMAT name is specified that is not defined at installation time. If the time-member reference is zero, the installation default is a FORMAT name that was not specified at installation; the default becomes OPEN.

Action: Review the permissible translate-time FORMAT= or TERM= sub-parameters and correct the OPTION card or PARM list.

X03 INVALID FORMAT NAME (ERROR)

Explanation: Occurs during translate-time parameter analysis. A FORMAT name is specified that was not defined at installation. If the time-member reference is zero, the installation default is a FORMAT name that was not specified at installation; the default becomes OPEN.

Action: Review FORMAT= option names defined at installation. Probable cause of error is misspelling.

X04 INVALID FORMAT LIST (ERROR)

Explanation: Occurs during translate-time parameter analysis. More than 13 parameters are specified for a FORMAT= option. Action is taken on the first 13 parameters only; the rest are ignored.

Action: Review the FORMAT= in error and limit its parameters to 13.

X05 NO INPUT FILE(S) SPECIFIED (FATAL ERROR)

Explanation: (CMS only) No input is defined for the Translator execution. No input or work files are accessed by the Translator. (This diagnostic is exclusively written to the terminal, regardless of any TERM= Translator Option specified.)

Action: Define an input file and rerun.

X06 PRIME INPUT DOES NOT EXIST (FATAL ERROR)

Explanation: (CMS only) The last or only input file does not exist on any accessed disk. No output or work files have been accessed by the Translator. This diagnostic is written to the terminal only, and no processing takes place.

Action: Ensure that the last and/or only input file is specified properly.

X07 OUTPUT DISK IS *o* **WORK DISK IS** *w* (NOTE)

Explanation: (CMS only) This message is produced by the Translator during initialization, indicating the disks on which the output (*o*) and the work (*w*) files are written.

Action: None required.

X08 NO READ/WRITE DISK DEFINED (FATAL ERROR)

Explanation: (CMS only) All disks are currently defined READ/ONLY. At least one READ/WRITE disk is required to execute the CA-MetaCOBOL+ Translator. No output or work files are accessed by the Translator. (This diagnostic is written exclusively to the terminal, regardless of any TERM= Translator Option specified.)

Action: Define a READ/WRITE file and rerun.

X09 UNRECOGNIZED TARGET OR DIALECT (WARNING)

Explanation: Occurs at the conclusion of translate-time parameter analysis. Undefined TARGET or DIALECT was to be placed into effect for the execution and could have been specified as the installation default, as a PARM or OPTION parameter, or as a Translator assumption because of a previous parameter error. The line number reference is always zero. The Translator assumes a TARGET=A value or the highest level DIALECT for the operating system is used.

Action: Review DIALECT= parameter and options.

X12 INVALID PRESERVE OPTION (ERROR)

Explanation: An invalid PRESERVE option has been specified.

Action: Correct the OPTION card and re-run.

X13 OBSOLETE OPTION (ADVISORY)

Explanation: An obsolete Translator Option has been specified.

Action: Delete the option and re-run.

X14 LCORE IGNORED FOR CA LIBRARIAN (ADVISORY)

Explanation: LCORE is no longer required for CA LIBRARIAN.

Action: Delete the LCORE option and re-run.

X15 NO DD STATEMENT FOR CARDF

Explanation: The DD statement is missing from the CARDF File, a primary input file required for MVS.

Action: Insert a DD statement in the CARDF File.

X16 NO DD STATEMENT FOR LSTIN

Explanation: The DD statement is missing from the LSTIN File, the printer file for the Input, Output, Auxiliary, and Lost Text listings.

Action: Insert a DD statement in the LSTIN File.

X17 NO DD STATEMENT FOR FM

Explanation: The DD statement is missing from the FM File, the required in-line work file.

Action: Insert a DD statement in the FM File.

X18 NO DD STATEMENT FOR FE

Explanation: The DD statement is missing from the FE File, the required out-of-line work file.

Action: Insert a DD statement in the FE File.

X19 IDENTIFIER LENGTH MAX OF 8 EXCEEDED

Explanation: The identifier specified on the unload or reload option can be a maximum 8 characters in length.

Action: Correct the Option statement and rerun.

X20 BAD LOAD FOR XXXXXXXX SNNN-NN (MVS)
BAD CDLOAD FOR XXXXXXXX RC=NN (VSE)

Explanation: An IXIT=*membername* could not be loaded.

Action: Check *membername* spelling. Also, check whether *membername* exists in the load library available to CA-MetaCOBOL+ translation.

X20 BAD LOAD FOR *membername*, s806-*system-return-code* (ERROR)

Explanation: An IXIT=*membername* could not be loaded.

Action: Check *membername* spelling. Also, check whether *membername* exists in the load library available to CA-MetaCOBOL+ translation.

X91 MINIMUM CORE REQUESTED IS NOT AVAILABLE (FATAL ERROR)

Explanation: *CA-MetaCOBOL+* has requested core storage for tables and input/output buffers. The minimum core storage requirement for execution is not available. CA-MetaCOBOL+ terminates.

Action: Review core storage requirements as discussed in the CA-MetaCOBOL+ *User Guide* and allocate the appropriate partition or region.

Z01 WORK FILE ERROR - CODE *n* (FATAL ERROR)

Explanation: (CMS only) A system error occurred in reading from MCTFM or MCTFE, where *n* is the error code defined for an FSREAD.

Action: Check the appropriate CMS manual for the meaning of *n* and correct as indicated.

Z02 OUTPUT ERROR - CODE *n* (FATAL ERROR)

Explanation: (CMS only) A system error occurred in writing to MCTOUT or MCTAUX, where *n* is the error code defined for an FSWRITE.

Action: Check the appropriate CMS manual for the meaning of *n* and correct as indicated.

Z03 ACCT NOT SUPPORTED (ERROR)

Explanation: (CMS only) A system error occurred in reading from MCTFM or MCTFE, where *n* is the error code defined for an FSWRITE.

Action: Review the macro set(s) involved and remove all occurrences of the &ACCT directive.

Appendix E. Account Management Review

The CA-MetaCOBOL+ Account Management Review (AMR) accumulates data and produces management reports on CA-MetaCOBOL+ usage by both programmer and type of activity. Specifically, the report is used to perform the following:

- Evaluate the usage of CA-MetaCOBOL+ by the programming staff.
- Provide automatic status reports of systems under development or involved in conversion or upgrading projects.

AMR functions as follows. Each time the CA-MetaCOBOL+ Translator is executed, raw accounting data may be accumulated in the Accounting File with the &ACCT directive expression. Periodically, reports are derived from the data in the Accounting File with the CA-MetaCOBOL+ AMR module.

The Accounting File is a separate low-volume file for recording CA-MetaCOBOL+ usage. Fixed-format records containing the program-name, author-name, current date, and identifying information are written to this file.

E.1 &ACCT Directive Expression

A record is inserted into the Accounting File during CA-MetaCOBOL+ translation with the &ACCT directive expression. The format of the &ACCT directive expression is as follows.

```
        { word          }
&ACCT   { &Vname        }
        { &n            }
        { concatenation }
```

The first execution of an &ACCT directive expression places the current value of the word, variable (&Vname), symbolic operand (&n), or concatenation construction in the text portion of the Accounting Record. Subsequent executions of the same &ACCT directive are ignored.

The operand can contain a current value which is up to 38 characters in length. Excess characters are truncated. The format of the 80-character Accounting Record is:

Position	**Field**
02-05	ACCT
07-44	text
46-53	current date
55-64	program-name
66-80	author-name

For example, assume that several hundred programs are to be converted to ANSI COBOL in an MVS environment and that a progress report of programs translated by the conversion procedure must be obtained.

First, raw data for the report must be accumulated in the Accounting File. This is accomplished by specifying the &ACCT directive in the model of a macro that is called *each* time the conversion procedure is invoked. In the following example, the macro $PDX is invoked at the end of the source input. ($PDX is an *event-dependent* macro; that is, it is invoked each time the program is translated.)

```
   . . .
S     $PDX :
         &ACCT &( 'CONVERT TO OS COBOL' &)
   . . .
       IDENTIFICATION DIVISION.
       PROGRAM-ID.  PREDIT.
       AUTHOR.  JONES.
           . . .
       PROCEDURE DIVISION.
           . . .
```

As a result of this translation, the following record is written to the Accounting File:

```
ACCT CONVERTED TO OS COBOL  11/07/91  T JONES
```

Accounting data can, therefore, be accumulated and reported by category of translation (the text portion), by date, by program-name, and/or by author-name.

Note: Macro sets supplied by Computer Associates do not contain &ACCT directives. CA-MetaCOBOL+ users who wish to audit and report specific categories of translation must include the &ACCT directive and supporting logic in the appropriate macro sets.

E.2 AMR Conventions

By convention, the AMR interprets 1-7 numeric characters beginning in the text field as a separate item known as *exceptions*. Thus the text field can contain two logical items: exceptions (the leading numeric characters, if any) and category.

For example, the &ACCT directive can be used to concatenate a variable containing the number of NOTE diagnostics displayed during translation and a literal indicating the category of translation. The following &ACCT directive places the exceptions and category in the text field as follows:

```
&ACCT &( &VERRORS 'CONVERSION TO OS COBOL' &)
```

A series of conversion translations may then accumulate the following accounting records in the Accounting File:

```
ACCT 0CONVERSION TO OS COBOL  10/01/91 INVCTL01  JONES
ACCT 5CONVERSION TO OS COBOL  10/01/91 INVCTL02  SMITH
ACCT 25CONVERSION TO OS COBOL 10/01/91 INVCTL03  DOE
```

Note that the category, CONVERSION TO OS COBOL, is preceded by the number of NOTE diagnostics.

E.3 Accounting File Characteristics

Under MVS, the Accounting File (ACCT) is an extendable data set. Therefore, accounting records must be reported periodically to clear the data set, thereby allowing additional raw accounting information to accumulate.

Under VSE, the Accounting File (SYS010) is not extendable. VSE CA-MetaCOBOL+ users must provide external procedures to ensure the accumulation and reporting of all Accounting Records.

E.4 Accounting Data Report Facility

The AMR module in the macro library supplied by Computer Associates is a CA-MetaCOBOL+ source program which edits and validates Accounting File input, sorts the accounting data into several sequences, updates an Accounting Master File, prints the accounting reports, and, under MVS, clears the Accounting File for subsequent accumulation.

The AMR module contains both macros and a basic source program. AMR therefore must be translated by the CA-MetaCOBOL+ Translator to produce an ANSI COBOL source program. Then it must be compiled, link-edited, and placed on the core-image library (VSE) or load library (MVS).

The AMR COBOL program operates in three logical segments: data preparation, sort, and master file update and reporting.

E.4.1 Data Preparation Segment

The first segment opens the Accounting File as input and reads each record sequentially. Records that do not contain ACCT in columns 02-05 are ignored. Leading numeric characters from the text field (columns 07-44) of valid Accounting Records are extracted as a separate field (exceptions). The remainder of the text field is left-aligned (category).

Each Accounting Record releases two records to the SORT: one for a report by category and the second for a report by author-name. Category records contain a '1' in the first position, followed by category, author-name, program-name, date (ascending sequence of input), a numeric field containing a 1 (becomes number of RUNS), and exceptions. The sort key is position 1 through the sequence-generated field. Author records contain a '2' in the first position, followed by author-name, program-name, category, date-generated, sequence-generated, RUNS, and exceptions. RUNS is a numeric field that originally contains a 1 and later stores the number of RUNS. The sort key is again position 1 through the postion of the sequence-generated field.

When all input is read, the Accounting File is closed and the input procedure is terminated.

E.4.2 Sort Segment

The second segment of the AMR program sorts the category and author records to the output procedure. All category records precede all author-name records for reporting by category before reporting by author-name.

E.4.3 Master File Update and Reporting Segment

The third segment of the AMR program opens an input Old Master File and an output New Master File. Sorted category and author-name records are matched/merged with the Old Master File in order to create a New Master File. Duplicate category records (category, author-name, and program-name) and duplicate author-name records (author-name, program-name, and category) are summarized into single records containing a count of duplicates (number of RUNS) and the data and exceptions count of the most recent translation. Each record written to the New Master File becomes a line item of the category or author report, as appropriate.

When all processing and reporting are complete, all files are closed. Under MVS, the Accounting File is then opened as output, a single blank record is written to reinitialize the file, and the file is closed. The AMR program then terminates.

Note: Under MVS, if the Accounting File is obtained by a system READER procedure, the logic required to reinitialize the file must be removed from the AMR program. If this logic is not removed, the AMR program attempts to open a spooling file as output--with unpredictable results.

E.5 MVS Operating Considerations

The MVS AMR COBOL program contains the following file assignments:

File-name	Assignment	Records/Block*	Record Length
ACCT-FILE	UT-S-SYSACCT	0	80
TRANS-FILE	UT-S-SYSSORT	N/A	84
OLD-FILE	UT-S-SYSOLD	0	80
NEW-FILE	UT-S-NEW	0	80
REPORT-FILE	UT-S-SYSPRINT	0	121

* Files must be blocked in multiples of Record Length.

It is recommended that the Accounting File (SYSACCT) be initialized on a direct-access device as a cataloged data set. STEP4 of the CA-MetaCOBOL+ *MVS Installation Guide* describes the initialization process. Reinitialization is accomplished by the AMR program. The summarized Master File can be maintained on disk or tape, possibly as a generation data set. Initialization of the Master File can be accomplished by defining SYSOLD as DD DUMMY during the first update cycle.

The following JCL statements execute the AMR program under MVS using generation data sets:

```
 //jobname  JOB ...
 //stepname EXEC PGM=AMRname
[//STEPLIB  DD DSN=user.loadlib,DISP=SHR]
 //SORTLIB  DD DSN=SYS1.SORTLIB,DISP=SHR
 //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)
 //SYSOUT   DD SYSOUT=A
 //SYSACCT  DD DSN=user.acct,DISP=(OLD,KEEP),DCB=BLKSIZE=80
 //SYSOLD   DD DSN=dsname(+0),DISP=OLD,DCB=catlgdcb
 //DD1      DD DSN=dsname(-n),DISP=OLD,
 //            DCB=catlgdcb,UNIT=(,,DEFER)
 //SYSNEW   DD DSN=dsname(+1),DISP=(NEW,CATLG,DELETE),
 //            DCB=catlgdcb,VOL=REF=*.DD1,UNIT=unit
 //SYSPRINT DD SYSOUT=A,DCB=BLKSIZE=121
```

E.6 VSE Operating Considerations

The VSE AMR COBOL program contains the following file assignments:

File-name	Assignment	Records/Block*	Record Length
ACCT-FILE	SYS006-UR-2540R-S-ACCT	unblocked	80
TRANS-FILE	SYS001-DA-2314-S-SORTWK1	N/A	84
OLD-FILE	SYS010-UT-2400-S-OLDMAST	10	80
NEW-FILE	SYS011-UT-2400-S-NEWMAST	10	80
REPORT-FILE	SYS008-UR-1403-S-REPORT	unblocked	121

Initialization of the tape Master File can be accomplished by assigning SYS010 to ignore (IGN) during the first update cycle. The following sample JCL statements are required to execute the AMR program under VSE:

```
 // JOB AMR UPDATE AND REPORT
 // ASSGN SYS006,X'cuu'           Card reader
 // ASSGN SYS008,X'cuu'           Printer
 // ASSGN SYS001,X'cuu'
 // ASSGN SYS002,X'cuu'           Sort work files
 // ASSGN SYS003,X'cuu'}
 // DLBL SORTWK1,,68/001,SD
 // EXTENT SYS001,volserno,1,0,20,100
 // DLBL SORTWK2,,68/001,SD
 // EXTENT SYS002,volserno,1,0,120,100
 // DLBL SORTWK3,,68/001/SD
 // EXTENT SYS003,volserno,1,0,220,100
 // ASSGN SYS010,X'cuu'
 // TLBL OLDMAST,'AMR MASTER',yy/ddd,volserno
 // ASSGN SYS011,X'cuu'
 // TLBL NEWMAST,'AMR MASTER',yy/ddd,volserno
 // EXEC AMRname
    (Raw Accounting Record input)
 /*
 /&
C
```

E.7 AMR Reports

This section contains an example of an AMR report by category and an AMR report by author-name.

E.7.1 Category Reports

The following category report, entitled CONVERSION TO OS COBOL, can be used as a progress report of a system conversion effort. The EXCEPTIONS listed in the right-hand column also indicate the magnitude of manual intervention required to complete the conversion.

```
12/18/91 CA-MetaCOBOL+ USAGE MANAGEMENT REVIEW REPORT  PAGE 1
ACTIVITY BY CATEGORY CONVERSION TO OS COBOL
```

AUTHOR	PROGRAM	RUNS	LAST RUN	EXCEPTIONS
MONAHAN	INVCTL03	1	09/10/91	18
	INVCTL06	1	09/05/91	20
	INVCTL09	1	09/05/91	16
	INVCTL13	1	09/08/91	11
	INVCTL16	1	09/08/91	25
	INVCTL19	1	09/10/91	4
WOODS	INVCTL02	1	09/01/91	9
	INVCTL05	1	09/01/91	5
	INVCTL08	1	08/05/91	26
	INVCTL12	1	09/08/91	5
	INVCTL15	1	09/10/91	23
	INVCTL18	1	09/01/91	4
	INVCTL22	1	09/18/91	22
WILSON	INVCTL01	1	09/01/91	27
	INVCTL04	1	09/01/91	27
	INVCTL07	1	09/05/91	1
	INVCTL11	1	09/05/91	24
	INVCTL17	1	09/15/91	31

E.7.2 Author Reports

The following is a report by author-name:

```
12/18/91 CA-MetaCOBOL+ USAGE MANAGEMENT REVIEW REPORT PAGE 1
ACTIVITY BY AUTHOR WILSON
```

PROGRAM	CATEGORY	RUNS	LAST RUN	EXCEPTIONS

DBMGMT03	NEW PROGRAM DEVELOPMENT	1	09/01/91	
DBMGMT06	NEW PROGRAM DEVELOPMENT	1	09/18/91	
DBMGMT09	NEW PROGRAM DEVELOPMENT	1	09/22/91	
DBMGMT13	NEW PROGRAM DEVELOPMENT	1	09/03/91	
DBMGMT16	NEW PROGRAM DEVELOPMENT	1	09/28/91	
DBMGMT19	NEW PROGRAM DEVELOPMENT	1	09/18/91	
INVCTL01	CONVERSION TO OS COBOL	1	09/01/91	27
INVCTL04	CONVERSION TO OS COBOL	1	09/01/91	27
INVCTL07	CONVERSION TO OS COBOL	1	09/05/91	1
INVCTL11	CONVERSION TO OS COBOL	1	09/05/91	24

Index

&

*

-

A

C

U

V

W

X