CA-MetaCOBOL™ +

Introduction

Release 1.1

R203M+11DRP

Release 1.1, January, 1992
Updated March, 1992

Contents

1. About this Manual

This manual gives a basic introduction to CA-MetaCOBOL+ and its uses. The major features of CA-MetaCOBOL+, including the Macro Facility and the Quality Assurance Facility, are described in detail.

1.1 Purpose

This document provides an overview of how each CA-MetaCOBOL+ component can be used to meet today's programming needs. The major features of each component and their relationship to each other are illustrated. This document is meant to give you a basic understanding of how CA-MetaCOBOL+ can enhance your efficiency and productivity.

1.2 Organization

This manual is organized as follows:

Chapter	Description
1	Introduces the contents and organization of this manual. Additional reference materials used with this manual are also listed.
2	Provides an overview of how CA-MetaCOBOL+ can help your organization maximize its use of the COBOL language.
3	Describes the major components of CA-MetaCOBOL+, including the Structured Programming Facility, Quality Assurance Facility, and Macro Facility.

1.3 Publications

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

Title	Contents
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.
User Guide	Explains how to customize, get started, and use CA-MetaCOBOL+. Includes information on keyword expansion, the CA-MetaCOBOL+ translator, and CA macro sets and programs.
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-DATACOM/DB environment.

Title	Contents
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 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.
Online Programming Language Guide	Contains further instruction for using the Online Programming Language.
PC User Guide	Explains how to use CA-MetaCOBOL+/PC. Includes information on the CA-MetaCOBOL+ translator and CA macro sets and programs. Also describes the relationship between CA-MetaCOBOL+ and CA-MetaCOBOL+/PC.
Program Development Guide CA-DATACOM/PC	Describes how to develop programs that use the CA-DATACOM/PC environment.
String Manipulation Language Guide	Introduces the String Manipulation Language, which provides string handling and inspection capabilities unavailable in COBOL.

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

1.4 Notation Conventions

This manual uses the following rules and special characters in syntax illustrations. Enter the following exactly as they appear in command descriptions:

UPPERCASE	Identifies commands, keywords, and keyword values which must be entered exactly as shown or replaced by an authorized abbreviation.
symbols	All special characters such as parentheses and quotation marks (but not ellipses, brackets, and braces) must be entered as shown.

The following notations clarify command syntax; do not enter them as shown.

lower case *italics*	Represent a value you must supply.
Brackets, [ ]	Identify optional keywords or clauses, or a group of options from which, if included at all, a choice of one must be made.
Braces, {}	Indicate that one of the keywords or clauses must be entered.
<u>Underlining</u>	Indicates a MetaCOBOL+ default that cannot be changed with a SET command, and therefore need not be specified.
Ellipses, ...	Indicate that the preceding word or clause can be repeated.

2. Overview

The COBOL language was designed to produce programs that are easy to read and easy to maintain. After more than 20 years of use, it remains the language of choice for large-scale application programming. During the life of any application program, changing technologies and user requirements mean that active programs must be modified and maintained. However, circumstances can make an old and valuable program hard to fix:

- Programmers change jobs. Because the average programmer stays with a company for less than four years, the person or persons who wrote the original program may be gone.
- Even a programmer can forget. Since a programmer can work on many projects over time, it's not unlikely that he or she will forget some of the program logic after a few months.
- The application programmer may not be the maintenance programmer. In this case, the person who maintains programs must learn someone else's approach to coding before doing any work.
- One program may require many programmers. Different approaches to designing and coding program modules may yield unexpected conflicts when one module is modified.

These are just some of the problems that can beset an organization with a long-term commitment to COBOL. The programmers who create and maintain applications must use the same sets of tools and concepts to provide consistent code that can be easily modified to meet new demands. The tools must be adaptable to new technologies without sacrificing current needs.

CA-MetaCOBOL+ provides that environment and maximizes your organization's investment in COBOL.

2.1 Maximizing Your Organization's Investment in COBOL

CA-MetaCOBOL+ provides extensions to the COBOL language that allow applications to grow. This growth retains past investment while allowing these same applications to interact with the dictionaries and databases of today in their current languages. CA-MetaCOBOL+ enables one systems programmer to create and implement an environment in which you can build or maintain COBOL programs in accordance with your requirements and desires. It enables you to set and enforce site standards, provides toolkits to extend the limits of the COBOL language, introduces editor extensions for TSO/ISPF, CA-ROSCOE, and CA-VOLLIE, and performs continual quality assurance as code is created or maintained. With CA-MetaCOBOL+/PC, you can develop, maintain, and execute CA-MetaCOBOL+ programs on the PC under MS-DOS or PC-DOS. For instructions to use CA-MetaCOBOL+/PC, refer to the CA-MetaCOBOL+/PC *User Guide*.

CA-MetaCOBOL+ simplifies the development and maintenance of CA-DATACOM/DB, DB2, CICS, VSAM, and conventional COBOL programs. This shortens the amount of time spent on initial program development, testing, debugging, maintenance, and documentation. Its flexibility enables you to modify and maintain existing COBOL code in any manner you choose; for example, you can use one set of standards for new programs, and another set for old programs.

2.1.1 Developing and Maintaining COBOL Programs

CA-MetaCOBOL+ makes it easy to develop and maintain CA-DATACOM/DB, DB2, CICS, VSAM, and conventional COBOL programs. The CA-MetaCOBOL+ development and maintenance system provides:

- Complete, high-level support of all structured programming concepts. Coding, testing, debugging, maintaining, quality assurance, and documenting are simplified.
- Quality assurance and formatting tools to ensure program consistency, clarity, and standardization. These facilities enable you to customize and tailor your site's COBOL programs.
- Easy access to CA-DATACOM/DB databases.

2.1.2 SQL Support for COBOL Programs

CA-MetaCOBOL+ provides a means for COBOL applications to evolve and interact with the dictionaries and databases of today's technology. The key is being able to extend COBOL to support SQL and being able to define rules for how to handle these extensions. CA-MetaCOBOL+ does just that.

CA-MetaCOBOL+ supports the embedding of SQL statements in COBOL programs. Additionally, editor extensions are provided for SQL constructs to assist in the accurate coding of SQL constructs for both DB2 and CA-DATACOM/DB 8.0. See Section 3.1.1 for details.

CA-MetaCOBOL+ also recognizes when the need for a DB2 precompile or CA-DATACOM/DB precompile is present. CA-MetaCOBOL+ automatically generates the necessary JCL step to satisfy that need. See Section 3.1.4 for details.

Finally, using the macro writing capabilities of CA-MetaCOBOL+ allows SQL constructs, as well as any complex-repetitive code, to be generated from macros as the result of an application programmer specifying a high-level verb. See Section 3.5 for details.

CA-MetaCOBOL+ protects your COBOL investment past, present, and future.

3. CA-MetaCOBOL+ Facilities and Features

CA-MetaCOBOL+ is designed to help you anticipate, forestall, or solve problems you may encounter when you build and maintain large-scale COBOL programs. It has a number of facilities to help you develop, code, test, debug, maintain, and document all of your organization's COBOL programs:

- The Macro Facility is the foundation for all of the CA-MetaCOBOL+ program development and maintenance tools. It enables you to manipulate the COBOL language and CA-MetaCOBOL+ environment to suit the unique requirements of your site.
- The CA-MetaCOBOL+ Structured Programming Facility implements all of the data structures and methodologies of the structured programming discipline to produce efficient code.
- The CA-MetaCOBOL+ Quality Assurance Facility enables you to implement and enforce site standards, control language use, improve readability through automatic formatting procedures, and document your program.
- The CA-DATACOM/DB Facility is fully integrated with CA-DATADICTIONARY, provides a simple but powerful Data Manipulation Language (DML) to access CA-DATACOM/DB from COBOL programs, and completely supports set-at-a-time or record-at-a-time processing.
- The Panel Definition Facility (PDF/CICS) provides a total environment for creating and maintaining panel definitions and generating BMS map source.
- The Online Programming Language (OPL) is an extension of the COBOL programming language. OPL provides high-level statements for CICS transaction processing in a syntax that is concise, easy to read, and easy to code.
- The CA-MetaCOBOL+ Work Bench incorporates many of the CA-MetaCOBOL+ facilities and features to provide an online system in which you can code, test, debug, modify, maintain, and document new or old COBOL programs in one session. You can use the Work Bench in tandem with any of the other CA-MetaCOBOL+ facilities.
-

Each facility stands alone, but you can use any or all of them during a single online session.

3.1 The CA-MetaCOBOL+ Work Bench

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

The Work Bench combines the features of your programming editor with the CA-MetaCOBOL+ Program Development and Quality Assurance Facilities to form an effective COBOL workstation. You can perform CA-MetaCOBOL+ tasks, and you can review listings, error messages, and test results in a single online session.

The CA-MetaCOBOL+ Dialog Main Menu groups services under the tasks listed on the menu, as shown in the figure below.

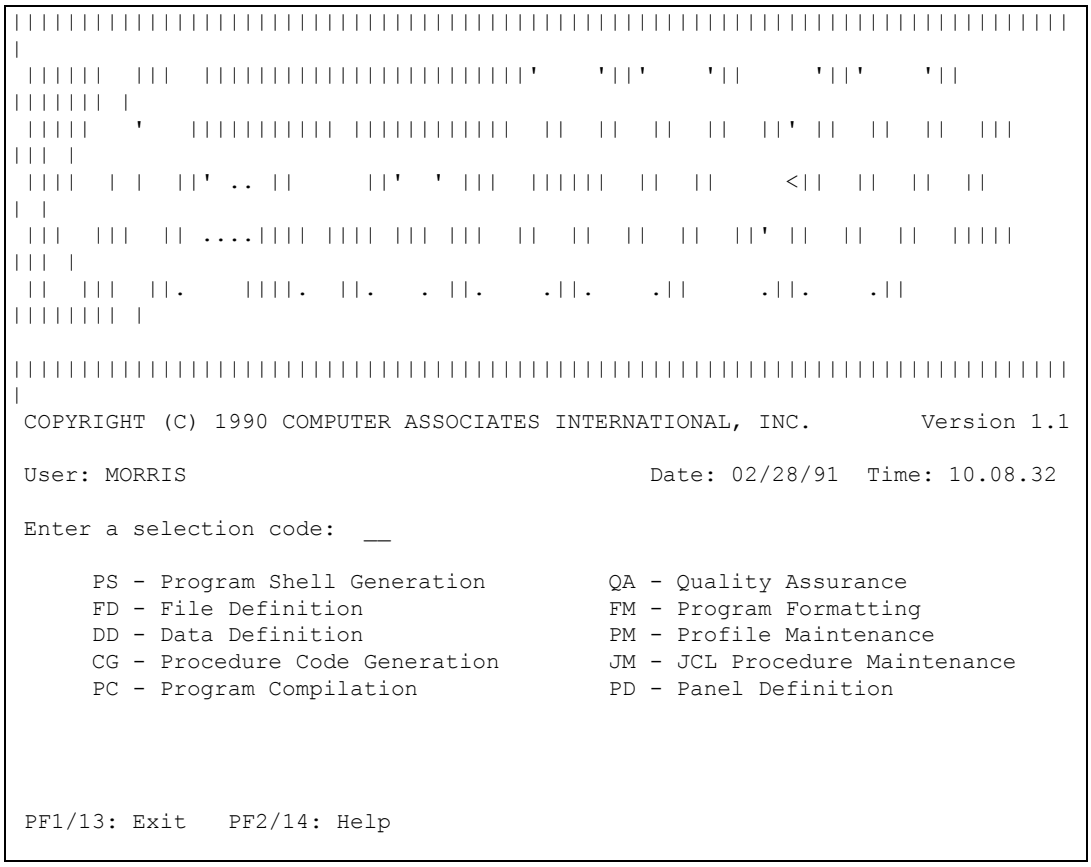

Figure1 1. The Dialog Facility Main Menu

CA-MetaCOBOL+ guides you through each facility. It edits and verifies user responses directly to minimize errors that may occur in subsequent steps. There is a context-sensitive help facility. The following sections describe each option on the Main Menu.

3.1.1 Program Shell Generation

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 (COBOL or SPP), and whether the program will use SQL statements for accessing CA-DATACOM/DB or DB2. The program shell is stored as a member in your library.

As an example, below is a sample program shell for a program that will access CA-DATACOM/DB. The lowercase 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
```

3.1.2 File Definition

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, or SORT/MERGE files.

3.1.3 Data Definition and Procedure Code Generation

These 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, SQL for CA-DATACOM/DB, and SQL for DB2. 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
```

In addition to keyword expansion, these services provide panels to define CA-DATACOM/DB dataviews.

3.1.4 Program Compilation

The Program Compilation service generates JCL to translate, precompile, compile, and link-edit the program. The precompile step may be required if the program contains CICS statements or SQL statements. The Job Submission 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.

3.1.5 Quality Assurance

The Quality Assurance Facility ensures that structured programming concepts are implemented in your COBOL programs. This facility:

- enables customization of the 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.

These services are fully described in Section 3.3.

3.1.6 Program Formatting

The Program Formatting Facility 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. 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
```

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

3.1.7 Profile Maintenance

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. Specifically, you can specify job card information, JCL procedures, CA-MetaCOBOL+ translate-time options, stub generator macros, formatting macros, and optimization macros.

For example, to change your compiler option, access the Dialog Facility Main Menu, select Profile Maintenance (PM) which puts you in a submenu, select Job Control Information on the submenu, 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.

3.1.8 JCL Procedure Maintenance

The JCL Procedure Maintenance services enable you to modify and update the JCL procedures used by the Job Submission services. You can save these procedures in your library.

3.1.9 Panel Definition

The Panel Definition Facility (PDF/CICS) is a complete environment for the creation and maintenance of CICS panels and BMS maps. PDF/CICS provides:

- a panel painter, which enables you to create a CICS panel via the panel layout process.
- a BMS map generator, which uses the CICS panel as input to generate BMS map source. The resulting map is stored in a member.
- a powerful, full screen editor, which enables you to edit both panel definitions and members.
- easy to use menus, command prompters, and fill-ins for the less experienced user.
- a comprehensive command language for the more experienced user.
- a print facility, for printing, viewing, and routing PDF/CICS output.

Panel definitions and members can be displayed, edited, printed, duplicated, renamed, indexed, and deleted with PDF/CICS.

3.2 The CA-MetaCOBOL+ Structured Programming (SP) Facility

Structured Programming is a philosophy that has become both a discipline and methodology. It emphasizes top-down design of programs and program modules with one entry point and one exit to eliminate confusion. It also emphasizes consistent formatting to make programs easy to read. For the same reason, sequence, repetition, and selection data constructs are frequently used. Structured programs are easy to code, test, debug, maintain, document, and modify.

The CA-MetaCOBOL+ Structured Programming (SP) Facility implements all of the concepts of structured programming, and gives you the tools to design and create an environment in which structured programs are easily built, tested, and maintained. Among other things, it enables you to:

- implement and support all structured programming constructs.
- encourage top-down development.
- include definitions of data structures local to a module inside that module.
- use an efficient method to define, test and modify control variables.
- provide true selection and repetition constructs.

▪ 3.2.1 Implements All Structured Programming Constructs

The following table lists the characteristics of structured programming and the constructs used to support these characteristics. It compares the support offered by the CA-MetaCOBOL+ SP Facility, ANSI 68 or ANSI 74 COBOL, and VS COBOL II:

Feature of Structured Programming	**Type of Support**	**CA SP Facility**	**ANSI 68 or 74 COBOL**	**VS COBOL II**
Enforces Adherence to Structured Programming	Mandatory single-entry single-exit modules	YES	NO	NO
	Scope terminators required	YES	NO	NO
	Provides for formatting of programs	YES	NO	NO
Top-Down Development	Explicit control variable definition	YES	NO	NO
	"Local" data definition in Procedure Division	YES	NO	NO
	Stub Generation	YES	NO	NO
	Documentation of module hierarchy and invocation	YES	NO	NO
Repetition Structure	DO-WHILE	YES	NO	YES
	DO-UNTIL	YES	NO	YES
	Process before and after test	YES	NO	NO
Selection (case) Structure	SELECT FIRST	YES	NO	YES
	SELECT EVERY	YES	NO	NO
	SELECT LEADING	YES	NO	NO

As you can see, the CA-MetaCOBOL+ SP Facility promotes strict adherence to the philosophy and discipline of structured programming.

By contrast, 1974 and 1968 ANSI COBOL do not support structured programming at all, and since VS COBOL II (1985 ANSI COBOL) merely provides two variants of the repetition construct (PERFORM...UNTIL) and a restricted form of the selection construct (EVALUATE), its support of structured programming is severely limited.

VS COBOL II has other limitations. It does not have facilities for top-down module development or for program formatting and documenting, both of which CA-MetaCOBOL+ provides. Unlike CA-MetaCOBOL+, VS COBOL II programs do not allow programmers to define data locally within a module; additionally, programmers cannot explicitly define control variables.

VS COBOL II does not enforce single-entry/single-exit modules: Programmers can still use the GO TO and PERFORM...THRU statements. This lack of enforcement means that programmers can undermine structured programming standards. The SP Facility, on the other hand, enables you to promote strict adherence to structured programming.

3.2.2 Encourages Top-Down Development

The Structured Programming Facility ensures that COBOL programs have well-defined segments and a high degree of structure by providing for single-entry/single-exit modules. This makes programs easier to develop, debug, and maintain.

COBOL does not provide for single-entry/single-exit modules. COBOL paragraphs are not modules because they allow control to "fall through" from one paragraph to the next and can be exited arbitrarily with a GO TO statement. At best, COBOL's lack of a well-defined module can make a program's logic difficult to follow. At worst, it can permit "spaghetti code" and programs that are impossible to maintain.

The CA-MetaCOBOL+ SP Facility enhances COBOL by encouraging top-down development. It redefines COBOL paragraphs as modules. Its structured programming statements invoke subordinate modules and return control to the invoking module directly. Thus control does not fall through to the next paragraph. It also prohibits GO TO and PERFORM...THRU statements, which inhibit top-down modular development. The SP Facility can help you create code that is truly modular.

3.2.3 Enables Local Data Definition

Programs are easier to read and maintain if definitions of data structures local to one module are included in that module instead of in the Data Division. The CA-MetaCOBOL+ START-DATA/END-DATA statement enables you to define data in the Procedure Division, so you can define data referenced by a single module within the module itself. For example:

```
. . .
PROCEDURE DIVISION
. . .
POSSIBLE-CONTROL-BREAK
    START DATA
    77  PREVIOUS-CUSTOMER PIC X(7) LOW-VALUES.
    01  CUSTOMER-AREA.
        02 CUSTOMER       PIC X(7).
        02 INVOICE-NO     PIC X(7).
        02 INVOICE-DATE   PIC X(6).
        02 INVOICE-AMT    PIC S9(9)V99 COMP-3.
    END DATA
    IF CUSTOMER = PREVIOUS-CUSTOMER . . .
. . .
```

The POSSIBLE-CONTROL-BREAK module contains the START DATA/END DATA statement, which defines data that will only be used inside the module. Using this construct means that maintenance programmers do not have to go to the Data Division to check local data definitions.

3.2.4 Makes Efficient Use of Control Variables

The Structured Programming Facility provides a precise, consistent method for defining, testing, and modifying control variables. Using this method, you can clearly distinguish between operations on control variables and operations on data. For example:

```
. . . .
GET-NEXT-INVOICE.
    START DATA
    77  FLAG RECORDS-LEFT IS TRUE.
    END DATA
    READ INVOICE-RECORD
    IF END
       SET-FALSE RECORDS-LEFT
    ENDIF
. . . .
```

The START DATA/END DATA statement sets a flag used inside the GET-NEXT-INVOICE module. Using this flag means that programmers can set control variables without having to specify them as level 88 entries in the Data Division. It also helps maintenance programmers check program flow.

3.2.5 Provides True Repetition and Selection Constructs

The SP Facility's repetition and selection constructs implement two characteristics of structured programming: repeating an operation and choosing one correct option among many. These constructs enable you to develop clear and concise COBOL programs. The repetition construct is the LOOP statement, and the selection construct is the SELECT statement.

The LOOP Statement

The LOOP statement enables a program to repeat an action a number of times, for example:

```
CLEAR-TABLE.
    SET X1 TO 1
    LOOP 10 TIMES
        MOVE SPACES TO TABLE-ENTRY (X1)
        MOVE ZEROS TO TABLE-AMOUNT (X1)
        SET X2 TO 1
        LOOP 5 TIMES
            MOVE ZEROS TO TABLE-UNITS (X1 X2)
            SET X2 UP BY 1
        ENDLOOP
    SET X1 UP BY 1
    ENDLOOP
```

In this example, nested loops are used to clear a table. In the outer loop, the first ten occurrences of TABLE-ENTRY are reset, including the elementary numeric item TABLE-AMOUNT.

The SELECT Statement

The SELECT statement enables a program to choose one action from a number of options, for example:

```
SELECT FIRST ACTION FOR TRANSACTION-CODE
  WHEN 'A'
    DO ADD-TRANSACTION
  WHEN 'C'
    DO CHANGE-TRANSACTION
  WHEN 'D'
    DO DELETE-TRANSACTION
  WHEN NONE ARE SELECTED
    DO ERROR-TRANSACTION
ENDSELECT
```

In this SELECT example, the identifier TRANSACTION-CODE is tested:

- If either the A, C, or D conditions is true, its subordinate process is executed, and control passes to the statement that follows ENDSELECT. If more than one condition is true, only the first true condition's subordinate process is executed.

- If these three conditions are false, then the WHEN NONE ARE SELECTED condition is true: its subordinate process, ERROR-TRANSACTION, is executed, then control passes to the statement that follows ENDSELECT.

The SP Facility also has SELECT EVERY and SELECT LEADING constructs for flexibility and power.

3.2.6 Documents Your Program

The CA-MetaCOBOL+ Structured Programming Facility creates reports to show relationships among modules in an application program. The Module Hierarchy Report and the Module Where Invoked Report are described in the following subsections.

The Module Hierarchy Report

The SP Facility creates a Module Hierarchy Report to show the relationships among modules in a program. For example:

```
01  AGED-TRIAL
  02  BEGIN-AGED-TRIAL
  02  GET-NEXT-INVOICE
  02  PROCESS-INVOICE
    03  POSSIBLE-CONTROL-BREAK
      04  CUSTOMER-BREAK
    03  AGED-INVOICE
      04  POSSIBLE-PAGE-BREAK
  02  END-AGED-TRIAL
    03  CUSTOMER-BREAK
    03  FINAL-BREAK
```

The Module Where Invoked Report

The SP Facility creates the Module Where Invoked Report, a cross-referenced listing that identifies each module in a program, the line number on which the module is defined, and line numbers on which a reference to the module occurs. For example:

```
                          DEFN    REFERENCES

AGED-INVOICE              001630  001450
AGED-TRIAL                001190
BEGIN-AGED-TRIAL          001320   001230
CUSTOMER-BREAK            001920   001590   001490
END-AGED-TRIAL            001480   001290
FINAL-BREAK               002110   001500
GET-NEXT-INVOICE          001370   001250
POSSIBLE-CONTROL-BREAK    001530   001440
POSSIBLE-PAGE-BREAK       002210   001700
PROCESS-INVOICE           001430   001270
REPORT-AGED-INVOICE       001680   001460
```

3.3 The CA-MetaCOBOL+ Quality Assurance (QA) Facility

The CA-MetaCOBOL+ Quality Assurance (QA) Facility provides toolkits to help an administrator or system programmer define the type of code all of the programmers at your site will create. Using these toolkits will help you produce code to meet your requirements, and decrease the amount of time spent testing, debugging, and maintaining COBOL programs. The QA Facility enables you to:

- define the rules for your organization's COBOL programming standard.
- enforce those rules.
- promote good programming practices.
- guarantee consistent language use.
- audit COBOL source to improve readability.
- produce program documentation to help maintenance programmers quickly determine the logic and structure of the program.

3.3.1 Enables You to Customize Your Environment

The CA-MetaCOBOL+ Quality Assurance Facility helps you establish the appropriate parameters for your organization. If further customization is required, the quality assurance macros are delivered in source form to allow easy modification or extension using the CA-MetaCOBOL+ Macro Facility. This enables you to tailor the way your organization uses COBOL. The Macro Facility is described in Section 3.5.

3.3.2 Enforces the Standards You Create

The Quality Assurance Facility audits and, in some cases, automatically modifies COBOL programs to make sure that they conform to your organization's COBOL programming standards. When automatic source modification will not provide clear and maintainable program logic, CA-MetaCOBOL+ issues a diagnostic to identify the violation. Programs that violate the site standards you specified do not enter production.

CA-MetaCOBOL+ provides a summary of the quality assurance and standards audit process, which identifies standards violations and language use information helpful to both managers and programmers. For example:

```
** COBOL QUALITY ASSURANCE DIAGNOSTIC SUMMARY
**
** CQA30A-GROUP ITEMS USED IN COMPARISON       5
** CQA31A-INCONSISTENT DATA ITEM USAGES        2
** CQA33A-INCONSISTENT DATA ITEM LENGTHS       1
** CQA34A-DISPLAY ITEMS USED IN ARITHMETIC     5
** CQA38A-INEFFICIENT SUBSCRIPT TYPES          4
** CQA60A-SHORT DATA-NAMES OR PROCEDURE-NAMES  3
```

3.3.3 Promotes Good Programming Practices

The QA Facility flags inefficient coding. For example, it can flag statements requiring operator intervention (STOP) and subscripts defined as COMP-3 or DISPLAY. It can also flag the ON SIZE ERROR clause as inefficient and as an indication of poor editing.

CA-MetaCOBOL+ helps prevent maintenance problems. For example, it flags language elements that tend to obscure program logic (ALTER, RENAMES, MOVE CORRESPONDING, GO TO DEPENDING). It helps provides meaningful data and procedure names by requiring these names to have a minimum number of characters. It signs COMP, COMP-3, and COMP-4 data items, and inserts VALUE clauses where they are missing.

3.3.4 Controls Language Use

Since COBOL is an evolving language, some language elements that exist today may become obsolete in a future revision to the COBOL standard. In between these revisions, various dialects of COBOL and vendor extensions appear. For example, IBM has provided 13 dialects of COBOL in the last 20 years. If your applications rely heavily on nearly-obsolete or non-standard syntax, your programs may not be upwardly compatible with future COBOL standards.

The CA-MetaCOBOL+ Quality Assurance Facility lets you control COBOL usage in two ways: first, it checks programs for non-standard or soon-to-be-obsolete language elements; second, it makes sure that syntax is specified consistently.

3.3.5 Improves Readability

The QA Facility enables an administrator or system programmer to specify that all programs created at your site are consistently formatted to help programmers read and debug COBOL programs. For example, it can automatically indent continued sentences and conditional statements, and it can make sure that each verb begins on a new line. In this respect, it replicates the function of the Program Formatting Facility, which is described in Section 3.1.6, "Program Formatting."

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
```

3.3.6 Documents Your Program

The CA-MetaCOBOL+ Quality Assurance Facility creates the Data Division Map to display information about a program's data structures. For example:

```
LEVEL  NAME             USAGE  BYTES  POSN OCCUR OCCLV REDEF  VALUE  SIGN  SYNC DIGIT DECML

       WORKING-STORAGE
 01    WORK-AREA         GRP   80     1                       VALUE
   02  TRANSACTION-CODE  A/N   1      11                      VALUE
   02  PLANT             NUM   3      20                      VALUE             3    0
   02  PRODUCT-CODE      A/N   1      35                      VALUE
   02  MAKE-SHIP-CODE    A/N   1      56                      VALUE
   02  GROSS             NUM   5      66                      VALUE             5    0
 01    COUNTERS          GRP   30     1                       VALUE
   02  PLANT-ADDITIONS   PACK  3      1                       VALUE
   02  PLANT-CHANGES     PACK  3      4                       VALUE  SIGN       5    0
   02  PLANT-DELETES     PACK  3      7                       VALUE  SIGN       5    0
   02  PLANT-GROSS-MADE  PACK  3      10                      VALUE  SIGN       5    0
   02  PLANT-GROSS-SHIP  PACK  3      13                      VALUE  SIGN       5    0
   02  FINAL-ADDITIONS   PACK  3      16                      VALUE  SIGN       5    0
   02  FINAL-CHANGES     PACK  3      19                      VALUE  SIGN       5    0
   02  FINAL-DELETES     PACK  3      22                      VALUE  SIGN       5    0
   02  FINAL-GROSS-MADE  PACK  3      25                      VALUE  SIGN       5    0
   02  FINAL-GROSS-SHIP  PACK  3      28                      VALUE  SIGN       5    0
 01    REPORT-RECORD     GRP   120    1                       VALUE
   02  REPORT-PLANT      A/N     3    6                       VALUE
   02  REPORT-ADDITIONS  NUM     5    14                      VALUE             5    0
   02  REPORT-DELETES    NUM     5    34                      VALUE             5    0
   02  REPORT-GROSS-SHI  NUM     5    54                      VALUE             5    0
   02  REPORT-FINAL      A/N     5    64                      VALUE
```

3.4 The CA-MetaCOBOL+ CA-DATACOM/DB Facility

The CA-MetaCOBOL+ CA-DATACOM/DB Facility is fully integrated with CA-DATADICTIONARY and includes the high-level language constructs used by CA-IDEAL, CA's advanced programming environment. It includes:

- a simple but powerful Data Manipulation Language (DML) to access CA-DATACOM/DB from COBOL programs.
- complete support for set-at-a-time and record-at-a-time processing.
- CA-DATADICTIONARY integration.

These features are described in the following subsections.

3.4.1 The Data Manipulation Language (DML)

The CA-MetaCOBOL+ CA-DATACOM/DB Data Manipulation Language (DML) consists of COBOL-like statements that support the full range of CA-DATACOM/DB services. The statements can perform any function that CA-DATACOM/DB commands perform: set-at-a-time processing, record-at-a-time processing, obtaining and releasing exclusive control, logging, and diagnostic assistance.

CA-MetaCOBOL+ statements are easier to use than CA-DATACOM/DB commands because their function is more direct and self-evident. For example, from the CA-DATACOM/DB CALL level, to retrieve record number 100 from an employee database, you would have to code the following:

```
MOVE 'PMF' TO EMPLOYEE-RA-FILE
MOVE 'EMPNO' TO EMPLOYEE-RA-KEY-NAME
MOVE 100 TO EMPLOYEE-EMPNO-RA-KEY
MOVE 'REDKX' TO EMPLOYEE-RA-FUNCTION
CALL 'DBNTRY' USING USER-INFORMATION-AREA
                    EMPLOYEE-REQUEST-AREA
                    EMPLOYEE-WORKAREA
                    EMPLOYEE-ELEMENT-LIST
```

From the CA-MetaCOBOL+ CA-DATACOM/DB Facility, you could code a single line:

```
READ EMPLOYEE WHERE ID-NUMBER = 100
```

CA-MetaCOBOL+ generates the code necessary to perform a CALL-level access to CA-DATACOM/DB. In addition, all of the program's database access statements are verified within a single CA-MetaCOBOL+ translation, which reduces testing time for the program.

3.4.2 Support for Set-at-a-time and Record-at-a-time Processing

The CA-MetaCOBOL+ CA-DATACOM/DB Facility provides statements for set-at-a-time and record-at-a-time processing, CA-DATACOM/DB's two primary access techniques. Set-at-a-time processing is the relational technique, and record-at-a-time processing is conventional key access.

In CA-MetaCOBOL+, relational access is realized with the FOR construct, which takes full advantage of CA-DATACOM/DB's set-at-a-time processing. The CA-MetaCOBOL+ FOR construct has a logical and English-like structure that makes it easy to implement database access from your program. It also automatically formats and generates the request qualification area, a complex task that is required when coding at the CALL level. For example:

```
FOR EACH EMPLOYEE RECORD
    WHERE (STATE-ADDRESS EQUAL 'MA' OR 'NY' OR 'TX')
    ORDER RECORDS ON SOC-SEC-NUM
    DO PRINT-REPORT
    WHEN ERROR
         DO EMP-ERROR-ROUTINE
    WHEN NONE
         DO NO-EMP-RECORDS
ENDFOR
```

LOCATE and READ are examples of the CA-DATACOM/DB Facility's record-at-a-time statements. LOCATE is an index processing statement, and READ is a data retrieval statement. You can use LOCATE and READ to access data sequentially or randomly.

LOCATE searches the index to quickly determine whether a record exists. It supports the full range of CA-DATACOM/DB index processing, including specific and generic searches. You can use the LOCATE statement to locate a record with a key that is equal to, less than, or greater than a specified value. You can implement scrolling by locating the next record, previous record, or all records within a range of key values.

The READ statement retrieves records that are currently located or whose key value has been specified explicitly. You can use the READ statement to read records with or without exclusive control over the record. As with LOCATE, READ also lets you scroll through the database by retrieving the next or previous record or all records within a range of key values.

3.4.3 Integration with CA-DATADICTIONARY

CA-DATADICTIONARY is a central repository of data descriptions for CA-DATACOM/DB. One type of data description is the dataview.

The dataview describes the data that can be requested in a single access from an application program. The dataview can be changed separately from the application program. This allows greater data independence than can be achieved when data descriptions are maintained within programs.

CA-MetaCOBOL+ is fully integrated with CA-DATADICTIONARY. This integration is accomplished in two ways:

1. The DATAVIEW statement identifies the CA-DATADICTIONARY dataviews required by the program.

2. During the program's translation, CA-MetaCOBOL+ retrieves the COBOL data names associated with the requested dataview from CA-DATADICTIONARY and generates all required data definitions necessary to support the database requests.

If a dataview requested by a program does not exist, or if the program is not authorized to access it, CA-MetaCOBOL+ does not allow the program to be translated. In this way, CA-MetaCOBOL+ ensures that a program accesses valid dataviews before the program is even compiled. This avoids execution errors caused by invalid CA-DATADICTIONARY dataviews.

So, you would not need to maintain a redundant COBOL copy library for data definitions. Instead, you could use CA-DATADICTIONARY as a central location of data definitions for CA-MetaCOBOL+, CA-IDEAL, and CA-DATAQUERY, and take full advantage of CA-DATADICTIONARY's security features.

3.5 The CA-MetaCOBOL+ Macro Facility

The Macro Facility is the foundation for all of the CA-MetaCOBOL+ program development and maintenance tools. The tasks you perform with CA-MetaCOBOL+ are implemented all or in part with the Macro Facility. With the Macro Facility, you can manipulate the COBOL language to suit the unique requirements of your organization. For example, you can:

- make programs portable across operating environments.
- create high-level interfaces to database and data communication systems not provided by Computer Associates.
- define specialized programming standards to enforce program efficiency, clarity, and portability.
- perform complex program conversions if the conversion task is too great to be effectively accomplished manually or by off-the-shelf conversion aids.
- develop an end-user application language, for example, an online banking language or a specialized report generator.
- customize other CA-MetaCOBOL+ facilities, such as the Quality Assurance Facility.

The Macro Facility features a COBOL-oriented macro programming language that manipulates COBOL source code. It provides great flexibility in processing COBOL input and output.

Basically, the macro language lets you replace a sequence of input COBOL words with another sequence of words. The macro language does not require you to specify an input sequence word for word; you can specify the input symbolically. CA-MetaCOBOL+ recognizes the attribute of an input word, that is, it can distinguish among data names, literals, figurative constants, procedure-names, and so on. In addition, the macro language provides selection, and repetition constructs.

By default, CA-MetaCOBOL+ places output words in the same location of the program as the input word they are replacing. A macro can override this rule by placing output, such as JCL, before or after the program, or in another COBOL division or section. Additionally, the macro can send its output to another file.

The following three-stage example shows how an effective macro can enhance COBOL and save time. Suppose printing a record requires you to code the following statements:

```
ADD literal TO LINE-COUNT
IF LINE-COUNT IS GREATER THAN +56
  PERFORM PAGE-HEAD-ROUTINE.
WRITE PRINT-RECORD FROM record-name
  AFTER ADVANCING literal LINES.
```

You can create the following macro:

```
SP   PRINT &1(S) BY &2(L) :
     ADD &2 TO LINE-COUNT.
     IF LINE-COUNT IS GREATER THAN +56
       PERFORM PAGE-HEAD-ROUTINE.
     WRITE PRINT-RECORD FROM &1
       AFTER ADVANCING &2 LINES.
```

Then, to print a record, you only have to code:

```
 PRINT record-name BY literal.
```

Macros and their COBOL input are processed by the CA-MetaCOBOL+ Translator. Macros can be input to the Translator as records preceding your COBOL source or they can be loaded separately from a library as one or more macro sets.

3.6 CA-MetaCOBOL+ Online Programming Language (OPL)

The Online Programming Language (OPL) is an extension to the COBOL programming language. OPL consists of statements for pseudo-conversational CICS transactions. OPL statements are coded in the Identification Division, Environment Division, Data Division, and Procedure Division of a COBOL program. The following example shows the overall structure of an OPL program.

OPL example

```
 IDENTIFICATION DIVISION.
 PROGRAM-ID.
    program-name.
 TRANSACTION-ID.
    CICS-transaction-id.
 ENVIRONMENT DIVISION.
        ...
 DATA DIVISION.
[MAP SECTION.]
[MAP DEFINITION statements...]
[RESPONSE DEFINITION statement]
[GLOBAL-STORAGE SECTION.]
[WORKING-STORAGE SECTION.]
[CONSTANT-STORAGE SECTION.]
        ...
 PROCEDURE DIVISION.
        ...
    OPL data-manipulation-statements...
       ...
```

OPL provides high-level statements for CICS transaction processing in a syntax that is concise, easy-to-read, and easy-to-code. CA-MetaCOBOL+ translates these statements into equivalent BMS panel I/O areas and command-level CICS statements. The result, or output, of the translation is conventional COBOL which is passed to the CICS precompiler.

An OPL programmer should be familiar with CICS, and have a working knowledge of the COBOL programming language.

Notes:

1. The use of SECTIONs is prohibited in an OPL program. The PROCEDURE DIVISION header must be immediately followed by a paragraph name.

2. The use of nested programs in an OPL application is prohibited.

3. OPL does not support BMS partitions or BMS paging.

Benefits

While CICS applications abound, CICS application programming is nonetheless an acquired skill. Creating an effective, efficient COBOL and command-level CICS program is a complex, error-prone, and time consuming task. Much of the inherent complexity of such a program lies in its need to manipulate CICS. The CICS programmer must give careful consideration to terminal I/O, storage management, database access, and transaction boundaries.

OPL offers a simple solution to this complex problem. By creating an OPL program consisting of conventional COBOL and OPL statements, both the program and the time required to create it are greatly reduced. An OPL program is translated by the CA-MetaCOBOL+ translator via a batch job execution. During translation, an OPL program generates all the COBOL and command-level CICS statements necessary to perform the following tasks:

- terminal I/O processing
- storage management
- database access
- transaction boundary crossing

In short, OPL frees the programmer from many CICS environmental concerns. OPL enables the experienced programmer to avoid some of the complications of COBOL and command-level CICS programming. The less experienced programmer also benefits from OPL, not only as a productivity tool, but as a learning tool. By using OPL, the programmer is able to program using proper structured design and makes fewer costly mistakes. The OPL programmer works at a higher conceptual level. Consequently, the program logic matches the perceived job to be done.

Index

W