CA-MetaCOBOL™ +

Online Programming Language Reference

Release 1.1

R203M+11DRP

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

Contents

1. About This Manual

1.1 Purpose

This manual provides an introduction to the CA-MetaCOBOL+ Online Programming Language (OPL) and a detailed description of all the OPL headers and statements. Familiarity with COBOL, CICS, and BMS maps is assumed.

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 introduction to OPL.
3	Describes the Identification Division OPL statements.
4	Describes MAP SECTION OPL statements that may be coded in either the Data Division or the Environment Division.
5	Describes the Data Division OPL statements.
6	Describes the Procedure Division OPL statements.
Appendix A	Lists the OPL reserved words.
Appendix B	Lists the OPL generated data-names.
Appendix C	Lists the OPL messages and corresponding corrective action.
Appendix D	Describes the CA-MetaCOBOL+ OPL macro sets. Additionally, MVS and VSE JCL examples for OPL program translation are provided.
Appendix E	Describes the OPL macro sets, JCL for performing OPL program translation, and JCL file requirements.

1.3 Publications

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, and the Online Programming 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.
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.

Title	Contents
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 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 Guide	Provides further instruction for using Online Programming Language statements.
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 Related Publications

The following publication relates to OPL and is supplied by Computer Associates:

Title
CA DATACOM/DB Programming Guide and Reference

The following publications relate to OPL and are supplied by IBM.

Title
CICS Application Programmer's Reference Manual
VS COBOL II Application Programming Guide
VS COBOL II Application Programming Reference
OS/VS COBOL Compiler and Library Programmer's Guide

1.5 Notation Conventions

This manual uses the following rules and special characters in syntax illustrations.

Enter the following exactly as they appear in header and statement descriptions:

UPPERCASE	Identifies optional, reserved keywords that must be entered exactly as shown (if included).
<u>Underlined</u>	Identifies reserved keywords that must be entered exactly as shown.
symbols	All special characters such as parentheses, periods, apostrophes, 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.
Ellipses, ...	Indicate that the preceding word or clause can be repeated.

2. Introduction

This chapter provides a brief overview of OPL, including its benefits and its relationship with other CA-MetaCOBOL+ facilities.

2.1 What is 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 that 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.

2.2 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.

2.3 Processing Overview

At the foundation of CA-MetaCOBOL+ is the Translator. The Translator is a general purpose, COBOL-oriented macro processor that can be customized, by means of its macro language, to solve a broad range of problems. The macro language enables the creation of specialized macro sets, designed to solve specific COBOL programming problems. These macro sets can be user written or supplied by Computer Associates. CA-MetaCOBOL+ is distributed with two OPL 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 statements, as described in Chapters 3 through 6.

To understand the development of an OPL program, review the steps involved in creating, translating, compiling, and link editing an OPL program.

First, the OPL program must be created and stored in a file. An editor, such as the CA ROSCOE, CA VOLLIE, or ISPF editor, is used to create the program and save it the desired file (e.g., an MVS partitioned data set, a CA LIBRARIAN master file, a CA ROSCOE library member, a CA VOLLIE library member, or a VSE source statement library). For detailed information on writing an OPL program, refer to the *Online Programming Language Guide*. Note that the creation of any necessary physical and symbolic BMS maps is a co-requisite of this step. These maps can be created with the CA-MetaCOBOL+ Panel Definition Facility (PDF/CICS) or created independently of PDF/CICS. Refer to the *Panel Definition Facility Command Reference Manual* and the *Panel Definition Facility User Guide* for more information on PDF/CICS.

Programs can also be created using CA-MetaCOBOL+/PC. Refer to the CA-MetaCOBOL+/PC *User Guide* for more information. Use the CA-MetaCOBOL+ *CA DATACOM/PC Programming Guide* to use CA-MetaCOBOL+/PC in a CA DATACOM/PC environment.

Second, the program must be translated, compiled, and link edited. This is accomplished with a batch job which performs the following steps:

1 Executes the CA-MetaCOBOL+ Translator specifying the OPLPS macro set.

2 Executes the CA-MetaCOBOL+ Translator specifying the OPL macro set. This step translates the high-level OPL statements into COBOL and command-level CICS statements.

3 Executes the IBM CICS precompiler, copying in any copybooks, or BMS symbolic maps, from the copybook source library (generally SYSLIB).

Step 3A, for CA-MetaCOBOL+ DML programs containing SQL statements, executes the CA DATACOM/DB precompiler. Refer to the CA-MetaCOBOL+ *Program Development Guide - CA DATACOM/DB* for more information on DL programs.

4 Executes the IBM COBOL compiler, producing object code for the next step.

5 Executes the IBM linkage editor, producing an executable load module (MVS) or phase (VSE) and storing it in a CICS accessible load library (MVS) or core image library (VSE).

Finally, the CICS Program Control Table (PCT) and Program Properties Table (PPT) are updated with information associating the program and mapset to a CICS transaction.

See Appendix D for detailed OPL translation JCL examples.

If you are developing CICS applications using CA-MetaCOBOL+/PC, you must use a CICS simulation product such as CA REALCICS to perform steps equivalent to those described in this processing overview. Refer to the CA-MetaCOBOL+/PC *User Guide* for more information.

2.4 OPL and Other CA-MetaCOBOL+ Services

OPL is one component in the CA-MetaCOBOL+ collection of COBOL programming tools. These tools can be used independently of one another, however, many of them provide complementary services. The OPL programmer can benefit from the following CA-MetaCOBOL+ services:

The CA DATACOM/DB Facility

The CA DATACOM/DB Facility provides a simple but powerful Data Manipulation Language (DML) to access CA DATACOM/DB from COBOL programs. OPL is fully compatible with DML (i.e., a COBOL program can contain both OPL and DML statements). See the CA-MetaCOBOL+ *Program Development Guide - CA DATACOM/DB* for information on DL programs.

PDF/CICS

CA-MetaCOBOL+'s Panel Definition Facility (PDF/CICS) is a panel painter and BMS map source code generator. The OPL programmer can further simplify the task of creating a psuedo-conversational CICS program by using PDF/CICS generated BMS maps. See the CA-MetaCOBOL+ *Panel Definition Facility User Guide* or *Panel Definition Facility Command Reference Manual* for information on PDF/CICS.

Work Bench

The CA-MetaCOBOL+ Work Bench is an online dialog facility that combines the features of a site's editor and the CA-MetaCOBOL+ program development facilities. An OPL programmer can take advantage of program shell generation, file and data definition, procedure code generation, program compilation, panel definition, and more. See the CA-MetaCOBOL+ *User Guide* for more information on the Work Bench.

Keyword Expansion

Keyword expansion is a feature of CA-MetaCOBOL+ that generates a template of a data description or a procedure statement. The template is generated by entering a keyword at the beginning of a program line and pressing a PF key. After the template is generated, it is completed by typing the appropriate information. From the CA-MetaCOBOL+ Work Bench, several keywords are available to create OPL templates. Refer to Appendix B, "Keywords," for a complete list.

Note: For more information on keyword expansion, refer to the CA-MetaCOBOL+ *User Guide*.

3. Identification Division

This section describes the statements coded in the Identification Division of an OPL program. The Identification Division header, the TRANSACTION-ID statement, and the PROGRAM-ID statement are required for CA-MetaCOBOL+ translation.

PROGRAM-ID Statement

The PROGRAM-ID statement is required by OPL.

Format:

```
      PROGRAM-ID. program-name.
```

program-name
Specifies a 1- to 8-character valid COBOL program-id.

TRANSACTION-ID Statement

The TRANSACTION-ID statement is used to identify the CICS transaction-id. OPL requires the CICS transaction-id in order to generate the CICS statements necessary for crossing transaction boundaries. The TRANSACTION-ID statement is required.

Format:

```
TRANSACTION-ID. cics-transaction-id.
```

cics-transaction-id
: Specifies the 1- to 4-character CICS transaction id. It must be defined as a transaction to CICS.

Example:

```
IDENTIFICATION DIVISION.
PROGRAM-ID.
     TESTPGM.
TRANSACTION-ID.
     TS01.
```

The PROGRAM-ID and CICS TRANSACTION-ID must correspond to definitions in the CICS Program Control Table (PCT) or to the equivalent definition using Resource Definition Online (RDO).

4. Map Section

This section describes the statements that may be coded in *either* the Data Division or Environment Division of an OPL program. The map section names each panel to be transmitted by an OPL program and defines responses for those panels.

MAP SECTION Header

The MAP SECTION header is an OPL extension of ANSI COBOL. It is not required; its purpose is to document the program. The MAP SECTION statement may be placed in the Environment Division or the Data Division of the program.

Format:

```
MAP SECTION.
```

MAP DEFINITION Statement

MAP DEFINITION statements are used to name each panel to be transmitted by a program. At least one MAP DEFINITION statement is required if any map is to be transmitted by the program. The MAP DEFINITION statement may be placed in the Environment Division or the Data Division of the program.

OPL will automatically copy the copybook for the BMS map from the SYSLIB into WORKING-STORAGE; therefore, a COPY statement for the BMS map should not be coded.

Use of the BMS BASE parameter is not supported by OPL. OPL assumes that storage is allocated automatically by WORKING-STORAGE.

The MAP DEFINITION also provides a way to associate a help map to a map or a field in a map. A help map must be named by the MAP DEFINITION statement for its corresponding map (i.e., a help map must not be defined with its own MAP DEFINITION statement). No copybook is used by OPL for help maps, and no data can be returned from a help map into an OPL program. HELP maps can be smaller than the screen, allowing help maps to be used as pop-up help windows.

By default, a help map will overlay the currently displayed map. Any portion of the current map not overlayed by the help map will remain displayed on the screen. To clear the current map and display only the help map, use the ERASE operand of the MAP DEFINITION statement.

You can use the Keyword Expansion keyword .MAPDEF, .MAP, or .MD for this statement.

Format:

```
MAP DEFINITION
    NAME IS mapname-1 [MAPSET IS mapset-name-1]
   [HELP IS mapname-2 [MAPSET IS mapset-name-2] ERASE]
   [VALIDATION PROCEDURE IS procedure-name-1]
   [MESSAGE-FIELD IS field name-1]
   [FIELD field name-2
        [HELP IS mapname-3 [MAPSET IS mapset-name-3] ERASE]
        [VALIDATION PROCEDURE IS procedure-name-2]
        [MESSAGE-FIELD IS field name-3]...].
```

mapname-1
mapname-2
mapname-3

Specifies a 1- to 7-character unique mapname. The map name is required and must be specified prior to any other clause of the MAP DEFINITION statement.

mapset-name-1
mapset-name-2
mapset-name-3

Specifies the 1- to 7-character mapset name. The mapset must contain the associated mapname. The mapset name is required only if it differs from the map name.

procedure-name-1
procedure-name-2

Specifies the 1- to 30-character unique, valid COBOL procedure name defined in the Procedure Division.

Note: A procedure can not be used in a validation procedure for more than one field or map.

field name-1
field name-2
field name-3

Specifies the 1- to 7-character BMS field name defined for the associated mapname. A field name is required only to assign field-level help, field-level messages, or field-level validation procedures.

Example 1:

```
MAP SECTION.
MAP DEFINITION
    NAME IS MAPAAAA.
```

In this example, the map and mapset name are assumed to be the same. Additionally, no OPL automatic processing is desired for help maps, field or map validation, or the display of error messages.

Example 2:

```
MAP SECTION.
MAP DEFINITION
    NAME IS AAAA MAPSET IS MAPAAAA
    HELP IS HLPAAAA
    MESSAGE-FIELD IS MSGAAAA
    VALIDATION PROCEDURE IS EDIT-AAAA
    FIELD FLD1
       HELP IS HLPFLD1
       VALIDATION PROCEDURE IS EDIT-FLD1
       MESSAGE-FIELD IS MSG1
    FIELD FLD2
       HELP IS HLPFLD2
       VALIDATION PROCEDURE IS EDIT-FLD2
       MESSAGE-FIELD IS MSG2.
```

This example uses several mapsets. Each of the three help maps is in its own mapset, which has the same name as the help map.

Example 3:

```
MAP SECTION.
MAP DEFINITION
    NAME IS MAINMAP MAPSET IS MULTMAP
    HELP IS MAINHLP MAPSET IS MULTMAP
    MESSAGE-FIELD IS MAINMSG
    VALIDATION PROCEDURE IS MAIN-EDIT
    FIELD KEY01
        HELP IS KEY1HLP MAPSET IS MULTMAP
        VALIDATION PROCEDURE IS KEY01-EDIT
```

This example uses a single mapset named MULTMAP, which contains all the maps used by this program.

RESPONSE DEFINITION Statement

The RESPONSE DEFINITION statement is used once in an OPL program to specify common responses to terminal attention and function keys for all maps transmitted by the program. The RESPONSE DEFINITION statement is optional. Depending on the processes specified in the RESPONSE DEFINITION statement, OPL will generate the appropriate logic for processing map events as part of both TRANSMIT statement code generation and HELP map processing.

Unless the RESPONSE DEFINITION statement indicates otherwise, CLEAR results in a return to CICS, and PA keys will refresh the screen image.

The RESPONSE DEFINITION statement applies to every TRANSMIT statement in the program. The RESPONSE DEFINITION statement may be placed in the Environment Division or the Data Division of the program.

You can use the Keyword Expansion keyword .RESPONSEDEF, .RESPDEF, or .RD for this statement.

Format:

```
RESPONSE DEFINITION
 WHEN condition-1 [OR condition-2...]
     process-1
[WHEN condition-3 [OR condition-4...]
     process-2...]
[WHEN OTHER
     process-3].
```

condition-1
condition-2
condition-3
condition-4

Must be one of the following conditions: PF1 through PF24, PA1 through PA3, CLEAR, or ENTER. The logical operator, OR, may be used to connect conditions PA1-PA3 and CLEAR, or any of PF1-PF24. OTHER, if specified, must be coded as the last WHEN clause.

process-1
process-2
process-3
Must be one of the following:

CONTINUE
Processing continues with a CICS RECEIVE MAP function and proceeds to the statement following the TRANSMIT statement.

DISPLAY *'message'*
The specified message is displayed in the map-level MESSAGE-FIELD defined in the RESPONSE DEFINITION statement and the panel is redisplayed. Any data entered in the panel is retained. The 'message' text must not exceed the length of the MESSAGE-FIELD defined in the RESPONSE DEFINITION.

ERASE
The screen is refreshed with the original image of the map.

GOBACK
Control is returned to CICS. A termination message is displayed on an unstructured screen.

HELP [RETURN-KEY IS KEY]
Processing continues with a TRANSMIT of the appropriate help map as named in a MAP DEFINITION statement. If the cursor is positioned anywhere on a field for which a help map is named, then that help panel is displayed. If a help map is not specified for the field, or the cursor is not positioned on a field, the help map named for the current map is transmitted. If a help map is not specified for either the field or the map, the current screen is redisplayed.

Note: When a help map is displayed as a result of the HELP process being specified in the RESPONSE DEFINITION statement, a RETURN event is required to return control to the current map. That is, a user viewing a help map must depress a key defined as RETURN-KEY to return to the original display. Any data typed into that map prior to HELP will be retained by the OPL program.

If this optional definition is omitted, the key defined for GOBACK will be used to return from help.

REFRESH
The screen is refreshed to its state prior to the TRANSMIT.

VALIDATE

Processing continues with a CICS RECEIVE MAP function and proceeds as follows:

- The MAP VALID condition is set.
- If specified for fields in the map definition, validation procedures are performed.
- If specified for the map definition, validation procedures are performed.
- If the validation procedure(s) detect an invalid condition, the map is retransmitted.
- If no validation procedures have been named for the current map, or an invalid condition is not detected, processing continues with the statement following the TRANSMIT statement.

Validation can also be programmed in the Procedure Division with the VALIDATE statement.

In summary, GOBACK exits the OPL program. CONTINUE and VALIDATE, with no invalid condition detected, continue with the statement following the TRANSMIT statement. ERASE, HELP, REFRESH, and VALIDATE, with an invalid condition, and DISPLAY continues according to the RESPONSE DEFINITION until a GOBACK, a CONTINUE, or an accepted VALIDATE cause the end of the TRANSMIT.

Notes:

1. Explicit specification of REFRESH, CONTINUE, or GOBACK for emergency situations is strongly recommended when VALIDATE is coded for one or more conditions. An inadvertent or incorrect MAP INVALID condition can cause an infinite loop unless an escape is coded with a REFRESH, a CONTINUE, or a GOBACK.

2. Only the RETURN-KEY definition is valid for help maps. When a help map is displayed, pressing this key erases the help map and redisplays the current map. Any other key, regardless of how it is defined, redisplays the help map.

Example 1:

If no RESPONSE DEFINITION statement is coded, the following is generated.

```
RESPONSE DEFINITION
WHEN CLEAR
    GOBACK
WHEN PA1 OR PA2 OR PA3
    REFRESH
WHEN OTHER
    CONTINUE
```

Example 2:

```
RESPONSE DEFINITION
WHEN PF3 OR PF15
    GOBACK
WHEN CLEAR
    ERASE
WHEN PA1 OR PA2 OR PA3
    REFRESH
WHEN ENTER
    VALIDATE
WHEN OTHER
    DISPLAY 'INVALID KEY WAS DEPRESSED'.
```

Notes:

1. Regardless of how many maps are defined, only one RESPONSE DEFINITION can be coded per program.

2. In order to define a PF key to perform different functions for different maps, it is recommended that the PF key be defined as CONTINUE in the RESPONSE DEFINITION and that the program use conventional methods to test for specific PF keys following the relevant TRANSMIT statements. The following sample OPL program demonstrates such a case. For map 'ABCD', PF3 is defined as GOBACK, for map 'QRST', PF3 is defined as a TRANSFER.

```
RESPONSE DEFINITION
WHEN PF1 OR PF13
    HELP
WHEN PF3
    CONTINUE.
       .
       .
       .
TRANSMIT MAP 'ABCD'
IF PF3
    GOBACK.
       .
       .
TRANSMIT MAP 'QRST'
IF PF3
    TRANSFER CONTROL TO TRANSACTION-ID BLTR.
```

5. Data Division

This section describes the statements coded in the Data Division of an OPL program. OPL provides for transaction storage, terminal storage, and temporary storage. Storage areas are saved and restored automatically.

Note that the OPL statements described in Chapter 4, "Map Section," can also be coded in the Data Division.

WORKING-STORAGE SECTION Header

WORKING-STORAGE in an OPL program functions as it does in a batch COBOL program. Any values placed in WORKING-STORAGE are automatically saved in main storage and automatically restored for future access, even across transaction boundaries.

Working storage is implemented by OPL as a temporary storage queue defined only to the current program. The OPL program manages the storage and retrieval of WORKING-STORAGE so it is always available for use. The WORKING-STORAGE SECTION header is not required.

Format:

```
    WORKING-STORAGE SECTION.
    01  data...
```

Valid level numbers are 01 through 48, 66, 77, and 88.

Note: All WORKING-STORAGE data is combined into a single temporary MAIN storage record with a key of term-id followed by 'MCTW'. The combined WORKING-STORAGE data can not exceed 32,767 bytes. WORKING-STORAGE is written before the TRANSMIT and retrieved after the TRANSMIT. It is deleted by a TRANSFER CONTROL or GOBACK.

GLOBAL-STORAGE SECTION Header

GLOBAL-STORAGE is based on CICS temporary storage. GLOBAL-STORAGE is accessible to every program and transaction executed at the same terminal. It is retained until a DELETE GLOBAL-STORAGE statement is executed. Data in this area may be passed between programs or accessed by the same program on subsequent calls. Each GLOBAL-STORAGE area has a name. A program may access more than one global area.

The GLOBAL-STORAGE SECTION header is required only if GLOBAL-STORAGE is desired. Each 01 level in the GLOBAL-STORAGE section represents a separate CICS temporary storage record identified by the storage name and terminal ID (a 4 character user-id and 4 character terminal id). The record remains accessible to other OPL programs until a DELETE GLOBAL-STORAGE statement is executed. The STORAGE NAME IS clause specifies a name used to generate the temporary storage record name. If a storage type is not specified, it is saved in MAIN storage.

Notes: Each GLOBAL-STORAGE record is read from temporary storage when the OPL program first receives control from CICS. All GLOBAL-STORAGE data is then combined into a single temporary storage record. When a TRANSMIT occurs, this combined record is written to MAIN storage (regardless of STORAGE TYPE) prior to sending the associated map. It is retrieved from MAIN storage when the map is received. The queue name for the TRANSMIT global record is the term-id followed by 'MCTG'. The combined GLOBAL-STORAGE data can not exceed 32,767 bytes.

When a TRANSFER CONTROL or GOBACK is encountered, each GLOBAL-STORAGE 01-level record is written to MAIN or DISK storage (as specified by STORAGE TYPE) with a queue name of 'stor-name' (as specified) followed by term-id. These records may be read by other OPL programs to which control is transferred, simply by matching the 'stor-names' specified by STORAGE NAME.

GLOBAL-STORAGE persists across GOBACK and TRANSFER CONTROL; therefore, care should be taken to use global data only for situations when such transfers are anticipated.

GLOBAL-STORAGE is deleted from temporary storage queues only by an explicit DELETE GLOBAL-STORAGE statement in the Procedure Division. WORKING-STORAGE is deleted automatically by OPL when the program executes an OPL TRANSFER CONTROL or GOBACK.

Format:

```
GLOBAL-STORAGE SECTION.
01  identifier      STORAGE NAME IS stor-name
                   [STORAGE TYPE IS MAIN/DISK].
    02  data...
```

identifier
Specify the 1- to 30-character COBOL name.

stor-name
Specify the 1- to 4-character name used to generate the temporary storage record name.

Example

```
GLOBAL-STORAGE SECTION.
01    XY01-TRANSACTION-PARM  STORAGE NAME IS 'XY01'
      05    XY01-PARM-MMDDYY      PIC 9(6)
      05    XY01-PARM-CODE        PIC X(2)
```

In the above example, a transaction plans to transfer control to one of several transactions (e.g., XY01, XY02, etc.). Transaction XY01 expects to receive a date and a product code. When the program performs the transfer control statement, OPL will write out any global storage records, thereby making data available to other transactions.

CONSTANT-STORAGE SECTION Header

The CONSTANT-STORAGE SECTION header contains definitions of constant data which will not be altered by the program. This statement is optional. Data items coded following the CONSTANT-STORAGE SECTION header are placed in the generated WORKING-STORAGE SECTION. Storage for the CONSTANT-STORAGE section is not saved during a transmit. Data may be moved into CONSTANT-STORAGE, but the values will be lost when a transmit occurs. After a transmit, all values in CONSTANT-STORAGE are set to their initial values.

Format:

```
CONSTANT-STORAGE SECTION.
01  data... VALUE 'value-literal'.
```

6. Procedure Division

The high-level terminal I/O and program control statements of OPL correspond to BMS functions and command-level CICS services. They may be coded anywhere in the Procedure Division, including within conditional statements. These statements are compatible with CA-MetaCOBOL+ COBOL/DL statements.

Note: The use of SECTIONs is prohibited in an OPL program. The Procedure Division header must be immediately followed by a paragraph name.

The statements in this section are listed in alphabetic order.

Condition-Names

The Procedure Division of an OPL program can contain OPL condition-names for function keys. These condition-names are translated by the OPL macros into equivalent EIBAID EQUAL DFHxxx' statements. The following is a list of the condition-names and their corresponding COBOL translation:

Condition-name	**Translation**
PF1 through PF24	EIBAID EQUAL DFHPFnn
PA1 through PA3	EIBAID EQUAL DFHPAn
CLEAR	EIBAID EQUAL DFHCLEAR
ENTER	EIBAID EQUAL DFHENTER

Example:

```
Input:          IF PF3
               TRANSFER CONTROL TO TRANSACTION-ID 'ABCD'

Output:    IF EIBAID EQUAL DFHPF3
               TRANSFER CONTROL TO TRANSACTION-ID 'ABCD'
```

Note that, for simplicity, the translation of the TRANSFER CONTROL statement is not shown here.

DISPLAY Statement

The DISPLAY statement is used to initialize a message field prior to the next map transmission. There are two formats of the DISPLAY statement.

The first format must be coded in a VALIDATION PROCEDURE. The message text is moved into the MESSAGE-FIELD associated with the field or map for which the VALIDATION PROCEDURE is named. A MESSAGE-FIELD must be specified for the map or field in the MAP DEFINITION statement.

The second format must be coded in the Procedure Division. This format requires the map name or field name for which the message is to be displayed.

You can use the Keyword Expansion keyword .DISPLAY or .DIS for this statement.

Format 1:

```
DISPLAY 'message-text'/message-identifier
```

Format 2

```
DISPLAY 'message-text'/message-identifier
   FOR mapname/field name
```

'message-text'
: Any alphanumeric literal enclosed by quotes. If the message text is longer than the destination message field, it will be truncated from the right.

message-identifier
: Any COBOL dataname. Must be defined in the WORKING-STORAGE, GLOBAL-STORAGE, or the CONSTANT-STORAGE SECTION. If the message identifier is longer than the destination message field, it will be truncated from the right.

mapname
: Specify the 1- to 7-character map named in a MAP DEFINITION statement.

field name
: Specify either a 1- to 7-character BMS field name or a COBOL dataname which corresponds to a BMS field name defined for an associated map. If duplicate field names occur in multiple maps, the field name may be qualified by the BMS map name.

Example of Format 1:

```
DATA DIVISION
MAP DEFINITION
    NAME IS AAAA
    VALIDATION PROCEDURE IS CHECK-MAP
    MESSAGE-FIELD IS MSG AAA.
         .
         .
         .
PROCEDURE DIVISION.
         .
         .
         .
CHECK-MAP.
    IF FLD-1 EQUAL FLD-2
        SET MAP INVALID
        DISPLAY 'FIELD-1 MUST NOT EQUAL FIELD-2'.
```

Example of Format 2:

```
DATA DIVISION.
MAP DEFINITION
    NAME IS AAAA
    MESSAGE-FIELD IS MSGAAAA.
        .
        .
        .
PROCEDURE DIVISION.
MAIN-LINE-ROUTINE.
    DISPLAY 'Enter password' FOR AAAA.
    TRANSMIT 'AAAA'
    GOBACK.
```

DELETE GLOBAL-STORAGE Statement

The DELETE GLOBAL-STORAGE statement is used to delete an area previously specified in the GLOBAL-STORAGE section of the program.

You can use the Keyword Expansion keyword .DELETEGLOBAL, .DELETEGBL, or .DELGBL for this statement.

Format:

DELETE GLOBAL-STORAGE [NAME IS *stor-name*]

stor-name
: A 1- to 4-character name as identified in the GLOBAL-STORAGE SECTION of the program. Required only when more than one 01-level record is named in the GLOBAL-STORAGE SECTION.

SET ALARM Statement

The SET ALARM statement is used to set the terminal to sound the audible alarm when the next map is transmitted. SET ALARM OFF is the initial default and will stay in effect until a SET ALARM ON statement is executed. Conversely, a SET ALARM ON statement will remain in effect for all subsequent TRANSMIT statements until a SET ALARM OFF statement executes.

The Keyword Expansion keyword for this statement is .SETALARM.

Format:

$$\underline{\text{SET}}\ \underline{\text{ALARM}}\quad \begin{bmatrix} \underline{\text{ON}} \\ \text{OFF} \end{bmatrix}$$

SET ATTRIBUTE Statement

The SET ATTRIBUTE statement is used to alter the attributes of a field or fields for the next transmission of the map.

You can use the Keyword Expansion keyword .SETATTRIBUTES, .SETATTRS, or .SETATTR for this statement.

Format:

```
SET ATTRIBUTE[S] attribute-1 [attribute-2...]
   ON field name-1 [field name-2...]
```

attribute-1
attribute-2...

Specify one or more of the following:

PROTECTED (or PROTECT)
UNPROTECTED (or UNPROTECT)
SKIP
NUMERIC
LOW (or NORMAL)
HIGH (or BRIGHT)
INVISIBLE (or DARK)
MDT (or RECEIVE)

field name-1
field name-2...

Specify a 1- to 7-character BMS field name or a COBOL dataname which corresponds to a BMS field name defined for an associated map. If duplicate field names occur in multiple maps, the field name may be qualified by the BMS map name.

Example:

```
PROCEDURE DIVISION.MAIN-LINE-ROUTINE.
    SET ATTRIBUTE PROTECT
        ON FLD1 OF AAAA
    TRANSMIT 'AAAA'
    GOBACK.
```

Note: If conflicting attributes are specified (e.g., SET ATTRIBUTE PROTECT UNPROTECT), then the last one specified is used. Conflicting attributes are PROTECT/UNPROTECT or HIGH/LOW/INVISIBLE.

SET COLOR Statement

The SET COLOR statement is used to alter the COLOR attribute of a field or fields for the next transmission of the map.

The Keyword Expansion keyword for this statement is .SETCOLOR.

Format:

<u>SET</u> <u>COLOR</u> *color*
  <u>ON</u> *field name-1* [*field name-2...*]

color

Specify one of the following:

BLUE
RED
GREEN
PINK
TURQUOISE
YELLOW
WHITE
DEFAULT (or NEUTRAL)

field name-1
field name-2...

Either a 1- to 7-character BMS field name or a COBOL dataname which corresponds to a BMS field name defined for an associated map. If duplicate field names occur in multiple maps, the field name may be qualified by the BMS map name.

Example:

```
PROCEDURE DIVISION.
MAIN-LINE-ROUTINE.
    SET COLOR BLUE
        ON FLD1 OF AAAA
    TRANSMIT 'AAAA'
    GOBACK.
```

SET CURSOR Statement

The SET CURSOR statement is used to position the cursor for the next transmission of a map. The SET CURSOR statement manages a program flag which determines, at the point of transmit, if cursor positioning is required. The flag is cleared after each transmit. Therefore, another SET CURSOR statement is required if explicit cursor control is desired prior to the next transmit. If used, the SET CURSOR statement will override the initial cursor attribute defined in the map, as well as symbolic cursor positioning.

You can use the Keyword Expansion keyword .SETCURSOR, .SETCSR, or .SETCUR for this statement.

Format:

```
                     ┌                ┐
                     │ HOME           │
                     │ field name     │
SET CURSOR AT        │ SAME POSITION  │
                     │ SAME FIELD     │
                     └                ┘
```

HOME

Specifies a position of row 0, column 0 on the terminal.

field name

Either a 1- to 7-character BMS field name or a COBOL dataname which corresponds to a BMS field name defined for an associated map. If duplicate field names occur in multiple maps, the field name may be qualified by the BMS map name.

SAME POSITION

Specifies that the cursor should remain in the same position for the next transmission of the map.

SAME FIELD

Specifies that the cursor should be placed at the beginning of the field the cursor is currently positioned on for the next transmission of the map.

Example:

```
PROCEDURE DIVISION.
MAIN-LINE-ROUTINE.
    SET CURSOR AT FLD1 OF AAAA
    TRANSMIT 'AAAA'
    GOBACK.
```

SET ERASE Statement

The SET ERASE statement is used to set internal parameters for the next TRANSMIT. SET ERASE ON is the initial default and will stay in effect until a SET ERASE OFF statement is executed.

The Keyword Expansion keyword for this statement is .SETERASE.

Format:

$$\underline{\texttt{SET}}\ \underline{\texttt{ERASE}}\quad \begin{bmatrix} \underline{\texttt{ON}} \\ \texttt{OFF} \end{bmatrix}$$

SET HIGHLIGHT Statement
The SET HIGHLIGHT statement is used to alter the extended attributes of a field or fields for the next transmission of the map.

You can use the Keyword Expansion keyword .SETHIGHLIGHT, .SETHIGH, or .SETHI for this statement.

Format:

```
SET HIGHLIGHT attribute-1 [attribute-2...]
    ON field name-1 [field name-2...]
```

attribute-1
attribute-2...
Specify one or more of the following:

BLINK
REVERSE
UNDERSCORE (or UNDERLINE)

field name-1
field name-2...
Either a 1- to 7-character BMS field name or a COBOL dataname which corresponds to a BMS field name defined for an associated map. If duplicate field names occur in multiple maps, the field name may be qualified by the BMS map name.

Example:

```
PROCEDURE DIVISION.
MAIN-LINE-ROUTINE.
    SET HIGHLIGHT UNDERLINE
        ON FLD1 OF AAAA
    TRANSMIT 'AAAA'
    GOBACK.
```

SET MAP Statement

The SET MAP statement is used in conjunction with the map or field VALIDATION PROCEDUREs to initialize and set the flag controlling the error state of the map last transmitted. SET MAP VALID makes the MAP-VALID condition-name TRUE. SET MAP INVALID makes the MAP-VALID condition-name FALSE.

Note that SET MAP applies to all maps in a program.

The Keyword Expansion keyword for this statement is .SETMAP.

Format:

$$\underline{\texttt{SET}}\ \underline{\texttt{MAP}}\ \left[\begin{array}{l}\underline{\texttt{INVALID}}\\ \underline{\texttt{VALID}}\end{array}\right]$$

Example:

```
VALIDATE-FLD1.
    IF FLD1 EQUAL '1' OR '2' OR '3' OR 'X'
        SET MAP VALID
        DISPLAY SPACES
    ELSE
        SET MAP INVALID
        DISPLAY 'VALID ENTRIES ARE 1, 2, 3, OR X'.
```

Note: When VALIDATE is specified in the RESPONSE DEFINITION statement, SET MAP VALID is implicitly executed prior to the execution of any field or map validation procedure.

When the VALIDATE statement is specified in the PROCEDURE DIVISION, the program must set and test the MAP-VALID condition-name to determine whether the map data is valid, and whether to retransmit the map.

TRANSFER CONTROL Statement

The TRANSFER CONTROL statement may be used to transfer to another program or transaction, or to transfer control back to CICS (synonymous with GOBACK).

The Keyword Expansion keyword for this statement is .TRANSFER.

Format:

```
                           [ program-name               ]
TRANSFER CONTROL TO        [ CICS                       ]
                           [ TRANSACTION-ID trans id    ]
```

program-name
: Any 1- to 8-character program name. The transferring program terminates and execution continues with the specified program.

trans-id
: Any 1- to 4-character valid CICS transaction-id. The transferring program terminates and execution continues with the program associated with the specified transaction.

CICS
: The transferring program terminates and sends a completion message on an unformatted screen. Control is returned to CICS. Transferring control to CICS is equivalent to GOBACK.

Example 1:

```
PROCEDURE DIVISION.
MAIN-LINE-ROUTINE.
    DISPLAY 'Enter password' FOR AAAA.
    TRANSMIT 'AAAA'
         .
         .
         .
    TRANSFER CONTROL TO CICS.
```

Example 2:

```
PROCEDURE DIVISION.
MAIN-LINE-ROUTINE
    TRANSMIT MENU
    IF CHOICE EQUAL '1'
       TRANSFER CONTROL TO TRANSACTION-ID ABCD.
    IF CHOICE EQUAL '2'
       TRANSFER CONTROL TO TRANSACTION-ID QRST.
    IF CHOICE EQUAL '3'
       TRANSFER CONTROL TO PROG14B.
    IF CHOICE EQUAL 'X'
       TRANSFER CONTROL TO CICS.
    GOBACK.
```

Note: GLOBAL-STORAGE is written to CICS temporary storage queues. The working data temporary storage queue is deleted.

TRANSMIT Statement

The TRANSMIT statement is used to send and receive the screen. At least one map must be identified with a MAP DEFINITION statement coded in the ENVIRONMENT or Data Division. If more than one MAP DEFINITION statement exists in the program, the map to be transmitted must be specified on the TRANSMIT statement.

During the TRANSMIT process, the PFn, PAn, CLEAR, and ENTER keys perform whatever process was assigned to them by the program's RESPONSE DEFINITION statement.

You can use the Keyword Expansion keyword .TRANSMIT or .TR for this statement.

Format:

$$\underline{\text{TRANSMIT}}\ \text{MAP} \begin{bmatrix} \text{'}\textit{mapname}\text{'} \\ \textit{map-identifier} \end{bmatrix}$$

'mapname'
: Required only when more than one MAP DEFINITION statement exists in the Data Division. Specify the 1- to 7-character map, named in a MAP DEFINITION statement, enclosed in quotes.

map-identifier
: Specifies any valid COBOL dataname defined in the WORKING-STORAGE SECTION, GLOBAL-STORAGE SECTION, or CONSTANT-STORAGE SECTION. Must be assigned the value of a map name defined on a MAP DEFINITION statement included in the program. If the map identifier contains an invalid value at the time of transmit, the transaction terminates with an error message on an unformatted screen. These messages are documented in Appendix D, "Diagnostics."

Example 1:

```
PROCEDURE DIVISION.
MAIN-LINE-ROUTINE.
    TRANSMIT 'AAAA'
    GOBACK.
```

Example 2:

```
PROCEDURE DIVISION.
MAIN-LINE-ROUTINE.
    MOVE 'AAAA' TO MAP-NAME
    TRANSMIT MAP-NAME
    GOBACK.
```

Notes:

1. The TRANSMIT statement may not be coded in a VALIDATION procedure.

2. The action taken during TRANSMIT is defined in the RESPONSE DEFINITION statement.

3. GLOBAL-STORAGE and WORKING-STORAGE are written to temporary storage and automatically restored.

VALIDATE Statement

The VALIDATE statement is used to invoke the procedures named in the MAP DEFINITION statement. At least one VALIDATION PROCEDURE must be named in the MAP DEFINITION statement for the desired map. This procedural form of the VALIDATE statement is provided as an alternative to the non-procedural form coded in the RESPONSE DEFINITION statement. This form allows the program complete control over which data is edited and the order in which the edit is performed.

The VALIDATE statement should be coded in conjunction with SET MAP VALID and SET MAP INVALID statements. Validation procedures can set the INVALID flag or reset it to VALID as necessary. Unlike VALIDATE processing in RESPONSE DEFINITION, invalid maps will not be retransmitted until validated. Instead, the program must test the MAP VALID/MAP INVALID condition and explicitly execute a transmit.

There are two formats of the VALIDATE statement. The first, VALIDATE MAP, invokes the validation procedure(s) named for a specific map. The second, VALIDATE FIELD, invokes validation procedure(s) for one or more specific fields. If the VALIDATE MAP statement is coded, procedures for fields are invoked in the order of their occurrence in the MAP DEFINITION statement, followed by the VALIDATION PROCEDURE named for the map. If the VALIDATE FIELD statement is coded, only procedures named in the MAP DEFINITION statement for the specified fields are invoked.

The Keyword Expansion keywords for the VALIDATE FIELD statement are .VALIDATEFIELD, .VALIDATEF and .VALFLD. The keywords for the VALIDATE MAP statement are .VALIDATEMAP, .VALIDATEM, and .VALMAP.

Format 1:

```
VALIDATE MAP 'mapname'
```

'mapname'
: Specify the 1- to 7-character map name, named in a MAP DEFINITION statement, enclosed in quotes. Required only when more than one MAP DEFINITION statement exists in the Data Division.

Format 2:

```
VALIDATE FIELD[S] field name-1 [field name-2...]
```

field name-1
field name-2
: Specify either a 1- to 7-character BMS field name or a COBOL dataname which corresponds to a BMS field name defined for an associated map. The associated MAP DEFINITION statement must contain a VALIDATION PROCEDURE for the specified field name. If duplicate field names occur in multiple maps, the field name may be qualified by the BMS map name.

Example 1:

```
PROCEDURE DIVISION.
MAIN-LINE-ROUTINE.
    TRANSMIT 'AAAA'
    VALIDATE 'AAAA'
    GOBACK.
```

Example 2:

```
PERFORM WITH TEST AFTER
UNTIL MAP-VALID
    TRANSMIT
    SET MAP VALID
    VALIDATE FLD-1
    IF STATE = 'TX'
        VALIDATE FLD-2
    ELSE
        VALIDATE FLD-3
    END-IF
    VALIDATE FLD-4
END-PERFORM
```

Note: The VALIDATE statement may not be coded in a VALIDATION procedure.

Appendix A. Reserved Words

In addition to the words reserved by COBOL, the following are reserved words in OPL.

ALARM
ATTRIBUTE
ATTRIBUTES
BLINK
BLUE
BRIGHT
CICS
CLEAR
COLOR
CONSTANT-STORAGE
CURSOR
DARK
DEFINITION
DISK
END-FOR
FIELD
FIELDS
GLOBAL-STORAGE
GREEN
HELP
HIGHLIGHT
HOME
INVISIBLE
LOW
MAIN
MAP
MAPSET
MDT
MESSAGE-FIELD
NAME
NEUTRAL
NORMAL
PA1, ... PA3
PF1, ... PF24
PINK
PROTECT
READ
REFRESH
REQUIRED
RESPONSE
REVERSE
SKIP
STORAGE
TRANSACTION-ID
TRANSFER
TRANSMIT
TURQUOISE
UNDERLINE
UNDERSCORE
UNPROTECT
UNPROTECTED
VALIDATION
WHITE
YELLOW

Appendix B. Keywords

Data Division

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

.DISPLAY	.MAP	.RESPDEF
.DIS	.MD	.RD
.MAPDEF	.RESPONSEDEF	

Procedure Division

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

Appendix C. Generated Data-Names

The following data-names are generated by OPL and are therefore reserved.

M--ALARM-OPTION

M--ATTRIBUTE-CONSTANTS

M--BLINK

M--BLUE

M--COLOR-CONSTANTS

M--COMMAREA-LEN

M--COMMAREA-LENGTH

M--CURRENT-DEPTH

M--CURSOR-FLAG

M--CURSOR-POSITION

M--DEFAULT

M--DFHCOMMAREA

M--DFHCOMMAREA-LEN

M--DFHCOMMAREA-LENGTH

M--DUMMY

M--ERASE-OPTION

M--ERROR-MSG-AREA

M--ERROR-MSG-AREA-2

M--ERROR-MSG-ID

M--ERROR-MSG-TEXT

M--ERROR-PGM-NAME

M--ERROR-RETURN-CODE

M--EVALUATE-SUBJECT-nnnn-nnn

M--fieldname-END-POSITION

M--fieldname-START-POSITION

M--FIELD-SUBSCRIPT

M--FILLER

M--FILLER-L

M--GLOBAL-STORAGE

M--GLOBAL-QUEUE-LENGTH

M--globalname-GBL-QUEUE-LEN

M--globalname-GBL-QUEUE-LENGTH

M--globalname-GBL-QUEUE-NAME

M--globalname-GBL-QUEUE-TERMID

M--GREEN

M--HELP-OPTION

M--HELPOPT-fieldname

M--HELPSET-fieldname

M--HELP-CONTROL

M--HELP-fieldname

M--HELP-MAP

M--HELP-MAPSET

M--HIGHLIGHT-CONSTANTS

M--INDEX-nn

M--MAP-ADDRESS

M--MAP-FIELD-PARAMETERS

M--MAP-NAME

M--MAP-NAMES

M--MAPNAME

M--mapname-FIELD-HELP-OPTION

M--mapname-HELP-OPTION

M--mapname-CURSOR-POSITION

M--mapname-FIELDS

M--mapname-FIELD-COUNT

M--mapname-FIELD-END

M--mapname-FIELD-HELP

M--mapname-FIELD-HELP-INFO

M--mapname-FIELD-HELP-MAP

M--mapname-FIELD-HELP-MAPSET

M--mapname-FIELD-HELP-TABLE

M--mapname-FIELD-INFO

M--mapname-FIELD-NAME

M--mapname-FIELD-POSITIONS

M--mapname-FIELD-START

M--mapname-FIELD-TABLE

M--mapname-HELP

M--mapname-HELP-INFO
M--mapname-HELP-MAPSET
M--mapname-QUEUE-LEN
M--mapname-QUEUE-LENGTH
M--MAPSET-NAME
M--MODULE-NAME
M--MSG-AREA
M--MSG-INFORMATION
M--MSG-LEN
M--OPL-MAP-INFO
M--PERFORM-SUBSCRIPT-nn
M--PERFORM-DEPTH
M--PERFORM-STATE
M--PGM-NAME
M--PINK
M--PROT
M--PROT-BRT
M--PROT-BRT-MDT
M--PROT-DRK
M--PROT-DRK-MDT
M--PROT-MDT
M--QUEUE-LENGTHS
M--RED
M--RETURN-CODE
M--REVERSE
M--SBA-ADDR
M--SBA-BYTE
M--SBA-NUMERIC
M--SBA-ORDER
M--SBA-ORDER-CODE
M--SCREEN-CONTROL
M--SCREEN-QUEUE-NAME
M--SCREEN-QUEUE-TERMID
M--SKIP
M--SKIP-BRT
M--SKIP-BRT-MDT
M--SKIP-DRK
M--SKIP-DRK-MDT
M--TRANSACTION-MSG-AREA
M--TURQUOISE

M--UNDERSCORE
M--UNPROT
M--UNPROT-BRT
M--UNPROT-BRT-MDT
M--UNPROT-DRK
M--UNPROT-DRK-MDT
M--UNPROT-MDT
M--UNPROT-NUM
M--UNPROT-NUM-BRT
M--UNPROT-NUM-BRT-MDT
M--UNPROT-NUM-DRK
M--UNPROT-NUM-DRK-MDT
M--UNPROT-NUM-MDT
M--USER-DFHCOMMAREA
M--VALID
M--WHITE
M--WORKING-QUEUE-LEN
M--WORKING-QUEUE-LENGTH
M--WORKING-QUEUE-NAME
M--WORKING-QUEUE-TERMID
M--WORKING-STORAGE-RECORD
M--WS-COMMAREA
M--WS-MAPNAME
M--WS-PGM-NAME
M--YELLOW
MAP-INVALID
MAP-VALID

Appendix D. Diagnostic Messages

This appendix lists and describes the messages that may be generated during OPL program translation or OPL program execution.

The format of the OPL messages is:

`OPL`*nnnc*

`where`:

OPL	is the prefix for diagnostics issued by OPL.
nnn	is the number of the diagnostic.
c	is the severity code:

A	is an advisory message.
E	is an error message. The source program requires at least one modification.
F	is a fatal error message. Processing ends.
W	is a warning message. Some automatic change must be reviewed.

D.1 OPLPS and DLMPS Macro Diagnostics

The following messages may be generated during a translation with either the OPLPS or DLMPS macro set.

Refer to the MetaCOBOL+ *Program Development Reference - CA DATACOM/DB* for a description of diagnostics specific to the DML language statements.

OPL001E - **"*identifier*" IS INVALID IN THIS CONTEXT.**

Corrective Action: The specified exception clause (AT END, ON SIZE ERROR, etc.) is not valid after the immediately preceding verb. Review the syntax for the preceding VERB and correct the source.

OPL002E - **STRUCTURE STACK FULL.**

Corrective Action: More than 99 levels of nested constructs are specified. Place constructs which are beyond this nesting capacity in separate paragraphs and PERFORM them.

OPL003W - **COBOL VERB EXPECTED,
FOUND "*identifier*".**

Corrective Action: A COBOL verb was expected, such as after an ELSE, and none was found. Review the syntax for the current construct and correct the source.

OPL004W - **INVALID 'NEXT SENTENCE'.**

Corrective Action: NEXT SENTENCE is only valid in the IF and SEARCH constructs and must not be part of a compound statement. Consider using CONTINUE rather than NEXT SENTENCE. Otherwise, review the syntax of the current construct and correct the source.

OPL005E - **INVALID STRUCTURE.**

Corrective Action: A clause or END-verb statement is found that does not correspond to any outstanding construct. Review the syntax for the current construct and correct the source.

OPL006W - **UNTERMINATED STRUCTURE.**

Corrective Action: A structure with no END-verb or period is found at the end of a paragraph or at the end of the program. A period is assumed. Supply the proper END-verb(s) or a period to terminate the structure.

OPL007E - **INVALID PERFORM SYNTAX.**

Corrective Action: Invalid syntax in a PERFORM statement is found. Review the syntax for the PERFORM statement and correct the source.

OPL008E - **MORE THAN 9 NESTED EVALUATES.**

Corrective Action: OPL can only handle 9 nested EVALUATE statements. Place the EVALUATE(s), which are beyond the nesting capacity of OPL in separate paragraphs and PERFORM them.

OPL009E - **MORE THAN 9 EVALUATE SELECTION SUBJECTS SPECIFIED.**

Corrective Action: OPL can only handle 9 EVALUATE selection subjects. Change the program logic so that a maximum of 9 EVALUATE selection subjects are specified.

OPL010E - **PARAGRAPH NAME MUST FOLLOW PROCEDURE DIVISION HEADER.**

Corrective Action: Insert a paragraph name for your mainline routine immediately following the PROCEDURE DIVISION header.

OPL011E - **INVALID ARITHMETIC EXPRESSION.**

Corrective Action: An invalid arithmetic expression is found as an EVALUATE selection subject or object. Review the syntax for an arithmetic expression and correct the source.

OPL012E - **INVALID EVALUATE WHEN CLAUSE.**

Corrective Action: An EVALUATE WHEN clause either has invalid syntax for one of the selection objects or the number of selection subjects and selection objects do not match. Review the syntax for an EVALUATE WHEN clause and correct the source.

OPL013E - **INVALID EVALUATE SYNTAX.**

Corrective Action: A WHEN clause does not immediately follow the last EVALUATE selection subject. Review the syntax for the EVALUATE statement and correct the source.

OPL014A - A SELECTION SUBJECT ARITHMETIC EXPRESSION MAY LEAD TO UNEQUAL COMPARE IF THE RESULT REQUIRES MORE THAN 5 PLACES OF DECIMAL PRECISION.

Corrective Action: To implement an arithmetic expression as an EVALUATE selection subject, OPL computes the expression into a temporary data item with PICTURE S9(12)V9(5). If this is not sufficient to hold the resulting values, unpredictable results may occur. If the size is not sufficient to hold the result, COMPUTE the expression into a temporary data item prior to the EVALUATE statement then specify the EVALUATE with the temporary data item as a selection subject rather than the expression.

OPL015E - IMPERATIVE STATEMENT MUST FOLLOW WHEN.

Corrective Action: Review the syntax for the EVALUATE statement and correct the source.

OPL016E - GO TO STATEMENTS ARE PROHIBITED.

Corrective Action: Structure the code with modules which can be PERFORMed. Eliminate all "GO TO" statements.

OPL017E - MARKERS REQUIRED TO COMPLETE TRANSLATION EXCEEDS MAXIMUM.

Corrective Action: Due to the number of PERFORM and SEARCH statements in the program, the marker limit has been exceeded. Reduce the number of these statements in your program.

OPL018E - THE USE OF SECTIONS IS PROHIBITED IN AN OPL PROGRAM.

Corrective Action: Structure the code with modules that can be PERFORMed. Remove all SECTION headers in the PROCEDURE DIVISION.

OPL019A - DATA DIVISION HEADER MISSING.

Corrective Action: Code a Data Division header.

OPL020A - **UNMATCHED "*scope-terminator*".**

Corrective Action: Verify the syntax of the preceding structures and eliminate the unnecessary terminator.

D.2 OPL Macro Diagnostics

The following messages may be generated during a translation with the OPL macros.

NO OPL ERRORS FOUND

Corrective Action: None.

OPL101W - **LEVEL NUMBER EXCEEDS MAXIMUM OF 48.**

Corrective Action: OPL increments level numbers coded in the WORKING-STORAGE SECTION of the program. Therefore the highest level number should not exceed 48. Decrement items at level 49 to level 48 or lower.

OPL102E - **MARKERS REQUIRED TO COMPLETE OPL TRANSLATION EXCEEDS MAXIMUM.**

Corrective Action: Contact CA-MetaCOBOL+ support.

OPL103E - **"*identifier-1*" IS INVALID SYNTAX IN "*identifier-2*" SKIPPING TO NEXT RECOGNIZABLE PHRASE, CLAUSE, OR STATEMENT.**

Corrective Action: Check the syntax for the statement indicated by identifier-2 and correct program source.

OPL105E - **EXPECTING "*identifier-1*" IDENTIFIER, FOUND "*identifier-2*".**

Corrective Action: Review the syntax for the preceding verb or statement and correct program source.

OPL106E - **EXPECTING "*identifier*" IDENTIFIER, BUT THE OPL STATEMENT PREMATURELY ENDED.**

Corrective Action: Review the syntax for the preceding verb or statement and correct program source.

OPL107E - DUPLICATE "*identifier*" CLAUSES ARE INVALID.

Corrective Action: Remove the incorrect duplicate clause.

OPL108E - "*identifier-1*" GREATER THAN MAXIMUM OF "*identifier-2*" CHARACTERS.

Corrective Action: Correct the specification of identifier-1 so that it is not longer than the length indicated by identifier-2.

OPL109E - "*mapname*" COPYBOOK DOES NOT CONTAIN DATANAME "*mapnameI*".

Corrective Action: Verify the copybook for the indicated map. If necessary, rerun the BMS utility for generating a DSECT.

OPL110A - NO MAPS WERE DEFINED FOR THIS PROGRAM.

Corrective Action: If a map is to be transmitted by the program via the TRANSMIT statement, then a map must be identified with the MAP DEFINITION statement. Otherwise, no action is required.

OPL111E - "*identifier-1*" CANNOT BE TRANSLATED BECAUSE THE PREVIOUS MAP DEFINITION STATEMENT DID NOT END PROPERLY WITH A PERIOD.

Corrective Action: End the preceding identifier-1 statement with a period.

OPL113E - "*mapname*" MAP IS NOT DEFINED IN THE PROGRAM.

Corrective Action: The indicated map has not been identified in the program. Include a MAP DEFINITION statement for the indicated map.

OPL114E - "*fieldname*" FIELD IS NOT DEFINED.

Corrective Action: The indicated fieldname is not found in the copybook for any map defined for the program. Either correct the spelling of the fieldname or name the field in the appropriate map via PDF/CICS.

OPL115E - **"*identifier*" DATANAME IS NOT DEFINED IN THE PROGRAM.**

Corrective Action: The indicated dataname is not found in the copybook for any map defined for the program. Either correct the spelling of the dataname, provide the dataname for a FIELD via PDF/CICS and regenerate the copybook, or otherwise include the dataname in the copybook for the map.

OPL116E - **"*identifier*" FIELD IS NOT UNIQUE AND MUST BE QUALIFIED.**

Corrective Action: The same field name is defined for more than one map for the program. Qualify the field name with the mapname. The mapname may be 1 to 7 characters.

If you have two fields named FIELD A, you can qualify them as FIELD A OF MAP A and FIELD A OF MAP B.

OPL117E - **"*identifier*" DATANAME HAS NO CORRESPONDING MAP FIELD.**

Corrective Action: The indicated dataname does not correspond by start position and length with any field defined in a map record. Review the copybook containing the dataname, verify the dataname, and correct the record layout.

OPL118E - **A MAP MUST BE SPECIFIED BECAUSE MORE THAN ONE MAP IS DEFINED FOR THE PROGRAM.**

Corrective Action: Specify a 1- to 7-character mapname for all TRANSMIT, VALIDATE, and SET CURSOR statements.

OPL119E - **"*identifier*" DATANAME IS NOT DEFINED.**

Corrective Action: The indicated dataname is not found in the corresponding map copybook. Correct the options specified for map attributes in PDF/CICS and regenerate the map.

OPL120E - **HELP PANEL NAME HAS NOT BEEN SPECIFIED.**

Corrective Action: In order to use HELP in the RESPONSE DEFINITION statement, help maps must be named in the MAP DEFINITION statement. Include at least one HELP clause in the map definition statement.

OPL121E - **MAP NAME MUST BE UNIQUE.**

Corrective Action: Provide a unique map name for each map definition statement.

OPL122E - **MAP NAME MUST PRECEDE identifier.**

Corrective Action: Place the NAME IS clause of the MAP DEFINITION statement so that it precedes the indicated clause.

OPL123E - **MAP NAME MUST BE SPECIFIED.**

Corrective Action: Include a NAME IS clause on each MAP DEFINITION statement.

OPL124E - **NUMBER OF MAP DEFINITION STATEMENTS EXCEEDS MAXIMUM OF 20.**

Corrective Action: Either limit the number of MAP DEFINITION statements in you program to 20, or contact CA-MetaCOBOL+ support for instructions on how to extend the limit in the OPL macros.

OPL125E - **DUPLICATE RESPONSE DEFINITION STATEMENTS ARE INVALID.**

Corrective Action: Remove all but one RESPONSE DEFINITION statement in the program.

OPL126E - **"OTHER" CANNOT BE USED IN A COMPOUND CONDITION.**

Corrective Action: WHEN OTHER cannot be used in conjunction with any other condition. Separate other conditions in a separate WHEN clause.

OPL127E - **"*identifier*" WAS PREVIOUSLY CODED IN A RESPONSE CONDITION.**

Corrective Action: Remove the duplicate specification for the indicated condition in the RESPONSE DEFINITION statement.

OPL128E - **SPECIFIC CONDITIONS MUST PRECEDE THE "WHEN OTHER" CLAUSE.**

Corrective Action: Move the WHEN OTHER clause to the end of the RESPONSE DEFINITION statement.

OPL129E - **PA KEY AND CLEAR CANNOT BE USED IN A COMPOUND CONDITION WITH PF KEYS OR ENTER.**

Corrective Action: Separate the conditions PA1 through PA3 and CLEAR from PF1 through PF24 and ENTER into different WHEN clauses of the RESPONSE DEFINITION statement.

OPL130E - **ONLY ONE RESPONSE PROCESS MAY BE SPECIFIED.**

Corrective Action: Remove all but one process from the WHEN clause on the indicated line of the RESPONSE DEFINITION statement.

OPL131A - THE RESPONSE DEFINITION STATEMENT IS INCOMPLETE AND WILL BE IGNORED.

Corrective Action: Review the syntax of the RESPONSE DEFINITION statement and correct the program source.

OPL132E - HELP MAPS ARE NOT DEFINED FOR MAP "*mapname*".

Corrective Action: In order to use HELP in the RESPONSE DEFINITION statement, you must name the help maps in the map definition statement. Include at least one HELP clause in the MAP DEFINITION statement for the indicated map.

OPL133E - THE RESPONSE DEFINITION MUST INCLUDE A DESIGNATED KEY FOR RETURN.

Corrective Action: In order to use HELP in the RESPONSE DEFINITION statement, a condition must be includeD for the RETURN process. Include this specification in the RESPONSE DEFINITION statement.

OPL134W - THE PRECEDING VALIDATION PROCEDURE "*identifier*" DOES NOT CONTAIN A SET MAP INVALID STATEMENT.

Corrective Action: Include a SET MAP INVALID statement somewhere in the indicated procedure paragraph.

OPL135W - THE VALIDATION PROCEDURE "*identifier*" IS NOT DEFINED.

Corrective Action: Include a paragraph for the indicated validation procedure, or remove the corresponding VALIDATION PROCEDURE clause from the MAP DEFINITION statement.

OPL136E - THE NUMBER OF VALIDATION PROCEDURES EXCEEDS THE MAXIMUM OF 99.

Corrective Action: Limit the number of VALIDATION procedures to 99 or fewer.

OPL137E - A VALIDATION PROCEDURE MUST NOT CONTAIN A TRANSMIT STATEMENT.

Corrective Action: Remove the TRANSMIT statement from the scope of the VALIDATION procedure paragraph.

OPL138E - A VALIDATION PROCEDURE MUST NOT CONTAIN A VALIDATE STATEMENT.

Corrective Action: Remove the VALIDATE statement from the scope of the VALIDATION procedure paragraph.

OPL139W - A VALIDATION PROCEDURE WAS NOT SPECIFIED FOR FIELD "*fieldname*".

Corrective Action: A VALIDATE FIELD statement is coded for the indicated field and a VALIDATION PROCEDURE has not been specified for it. Include, in the MAP DEFINITION statement, a FIELD definition which includes a VALIDATION PROCEDURE clause.

OPL140E - A VALIDATION PROCEDURE MUST BE UNIQUE.

Corrective Action: Provide a unique name for each VALIDATION PROCEDURE specified in the MAP DEFINITION statements.

OPL141A - MESSAGE TEXT FOR FIELD "*fieldname*" WILL BE TRUNCATED ON THE RIGHT.

Corrective Action: Either decrease the length of the message text or increase the size of the message field.

OPL142E - DESTINATION FOR MESSAGE, "*identifier*", IS NOT A VALID FIELD OF MAP "*mapname*".

Corrective Action: Verify the destination fieldname and correct the program source.

OPL143E - DESTINATION FOR MESSAGE, "*identifier*", HAS NO DESIGNATED MESSAGE FIELD.

Corrective Action: Include in the MAP DEFINITION statement a MESSAGE-FIELD clause for the indicated field or map.

OPL144E - MESSAGE IDENTIFIER "*identifier*" IS UNDEFINED.

Corrective Action: Include somewhere in the WORKING-STORAGE, GLOBAL-STORAGE, or CONSTANT-STORAGE SECTION, a definition of the indicated data item.

OPL145E - INVALID CONTEXT FOR THIS FORMAT OF THE DISPLAY STATEMENT.

Corrective Action: Either include a designation of the field or map for which the message is intended, or move the DISPLAY statement to a VALIDATION PROCEDURE paragraph.

OPL146E - **THE MAXIMUM NUMBER OF GLOBAL-STORAGE ITEMS HAS BEEN EXCEEDED.**

Corrective Action: Either limit the number of GLOBAL-STORAGE areas in your program to 20, or contact CA-MetaCOBOL+ support for instructions on how to extend the limit in the OPL macros.

OPL147E - **THE RECORD NAME IS MISSING FOR THIS GLOBAL-STORAGE AREA.**

Corrective Action: Follow each 01 level number in the GLOBAL-STORAGE section with a valid COBOL dataname up to 30 characters.

OPL148E - **STORAGE NAME IS INVALID.**

Corrective Action: Review the syntax for the STORAGE NAME IS clause of the GLOBAL-STORAGE section and correct program source.

OPL149E - **STORAGE TYPE IS INVALID.**

Corrective Action: Review the syntax for the STORAGE TYPE IS clause of the GLOBAL-STORAGE section and correct program source.

OPL150E - **STORAGE NAME IS MISSING FOR "*identifier*".**

Corrective Action: Include a STORAGE NAME IS clause for each 01 level item in the GLOBAL-STORAGE section.

OPL151E - **STORAGE NAME IS REQUIRED.**

Corrective Action: Include a NAME IS clause for each DELETE GLOBAL-STORAGE statement.

OPL152E - **GLOBAL-STORAGE HAS NOT BEEN DEFINED WITH THE NAME "*identifier*".**

Corrective Action: Verify the STORAGE NAMEs of each 01 level item in the GLOBAL-STORAGE SECTION. Correct the name specified on the DELETE GLOBAL-STORAGE statement in the program source.

OPL153E - **A "*identifier*" STATEMENT MUST BE CODED IN THE IDENTIFICATION DIVISION.**

Corrective Action: Review the syntax for the identified statement and correct program source.

OPL154A - **MULTIPLE "*identifier*" STATEMENTS ARE INVALID, STATEMENT IGNORED.**

Corrective Action: Remove the identified duplicate statement in the program source.

OPL155W - **PROCEDURE DIVISION HEADER IS MISSING; NO PROCEDURE DEFINED FOR THIS PROGRAM.**

Corrective Action: Code a Procedure Division.

OPL156E - **"*identifier*" EXCEEDS MAXIMUM ALLOWABLE LENGTH OF 32767 BYTES.**

Corrective Action: Reduce the size of the indicated area.

OPL157A - **NO MESSAGE FIELD WAS SPECIFIED FOR MAP "*map name*".**

Corrective Action: If desired, specify the MESSAGE-FIELD clause on each MAP DEFINTION statement.

OPL158E - **CONDITION SPECIFIED IS TOO LONG TO PROCESS BEGINNING WITH "*identifier*".**

Corrective Action: Code conditions beginning with the indicated key on a separate WHEN clause in the RESPONSE DEFINTION statement.

OPL159A - **NO VALIDATION PROCEDURES HAVE BEEN SPECIFIED FOR MAP "*map name*".**

Corrective Action: Include specification of one or more VALIDATION PROCEDURES in the indicated MAP DEFINITION statement.

OPL160E - **STORAGE NAME "*identifier*" IS NOT UNIQUE.**

Corrective Action: Provide a unique name in each STORAGE NAME clause on every GLOBAL-STORAGE record.

OPL161E - **"*process*" IS RESTRICTED FOR "*identifier*" PROCESSING.**

Corrective Action: Do not specify HELP or VALIDATE for the CLEAR key process, and do not specify VALIDATE for any PA key process. Review the RESPONSE DEFINITION statement syntax.

OPL162W - **"CONTINUE" PROCESS SPECIFIED FOR "*identifier*" PROCESS, BUT NO MAP WILL BE AVAILABLE FOR PROCESSING.**

Corrective Action: None required.

D.3 Run-time Diagnostics

The following messages may be generated by an OPL program at run-time.

***transid* transaction completed**

Corrective Action: None. Indicates the normal completion of the executed transaction.

***transid* transaction msg: OPL901E map *mapname* not defined to program**

Corrective Action: A TRANSMIT statement has been executed but does not refer to a valid map as named on a MAP DEFINITION statement. Verify and assign a valid map name to the map identifier specified in the TRANSMIT statement.

***transid* transaction msg: OPL902E *mapset-name* unavailable**

Corrective Action: The indicated mapset is either not defined or is disabled in the CICS system definition file. Define, enable, and install the mapset so that the mapset information is stored in the CICS PPT.

***transid* transaction msg: OPL903E module *mapset-name* not found**

Corrective Action: Assemble and linkedit the BMS macros for the indicated mapset. Include the load library in the DFHRPL DD in the active CICS system.

***transid* transaction msg: OPL904E module *mapset-name* does not contain requested mapset**

Corrective Action: Verify the load module with the indicated name. If necessary, reassemble and linkedit the BMS map macros for the indicated mapset.

***transsid* transaction msg: OPL905E module *mapset-name* does not contain the requested map**

Corrective Action: Verify the load module with the indicated name. If necessary, reassemble and linkedit the BMS map macros for the indicated mapset.

***transid* transaction msg: OPL906E MAP *mapname* out of sync, too few fields**

Corrective Action: Verify the copybook for the indicated map. If necessary, regenerate the copybook from the BMS macros and retranslate the program.

***transid* transaction msg: OPL907E MAP *mapname* out of sync, too many fields**

Corrective Action: Verify the copybook for the indicated map. If necessary, regenerate the copybook from the BMS macros and retranslate the program.

Appendix E. Translation Requirements

This section describes the CA-MetaCOBOL+ OPL macro sets, JCL for performing OPL program translation, and the JCL file requirements. For information on translate-time options, source file concatenation, condition codes, and memory considerations, refer to the CA-MetaCOBOL+ *User Guide*.

CA-MetaCOBOL+ translates OPL programs into standard COBOL. The output source program is then translated by the CICS Command-Level Processor and compiled by a standard IBM COBOL compiler. Input to CA-MetaCOBOL+ consists of:

- The translate-time options that modify CA-MetaCOBOL+ translation
- The macro set desired for a translation
- The copybooks which represent the maps transmitted by a program
- The OPL source program

Input may be provided directly from the Primary Input File, from source libraries or CA LIBRARIAN or CA PANVALET master files.

E.1 The OPL Macro Sets

An OPL program must be translated twice. Two macros sets (OPLPS and OPL) are provided with CA-MetaCOBOL+ for OPL translation. They are described below.

OPLPS restructures the input program's source code, as required for pseudo-conversational source code generation.

OPL translates the high-level statements in the OPL language as described in Chapters 3 through 6.

Depending on whether the OPL program also contains DML statements, the DLM macro set is specified along with OPLPS for the first execution of the CA-MetaCOBOL+ Translator. OPLPS should be specified before DLM. If the Structured Programming Language is going to be used as well, then the SP macro should be specified first. The OPL macro set is always specified for the second execution of the Translator.

E.2 Translation Under MVS

Note: If you are using CA-MetaCOBOL+/PC, refer to the CA-MetaCOBOL+/PC *User Guide* for equivalent file requirements and SET Statements required for translation.

E.2.1 MVS File Requirements

The following list contains the applicable MVS DDnames with a brief description of their use.

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 translate-time 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.

E.2.2 MVS JCL Example

The following JCL procedure example is provided 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. Optional JCL statements appear in brackets.

```
//METACBL PROC META=mctpgm,
CA-MetaCOBOL+ Prog. Name
//              STEPLIB='mct.load.lib',             Load
Library Name
//              COBOL=NULLFILE,                     COBOL
Output
//              AUX=NULLFILE,
Auxiliary Output
//              COPYLIB='copy.libr',                COPY
Library
//              LIBMSTR='mct.lib.mast',      Disk CA
LIBRARIAN Master
//              TERM=NULLFILE,                      Terminal
Data Set
//MCT     EXEC PGM=&META
//STEPLIB   DD DSN=&STEPLIB,DISP=SHR
//LSTIN     DD SYSOUT=A,
//             DCB=(RECFM=FBA,LRECL=121,BLKSIZE=605)
//PUNCHF    DD UNIT=SYSDA,
//             SPACE=(TRK,100),
//             DCB=BLKSIZE=400,
//             DISP=(,PASS)
//AUX       DD DSN=&AUX,
//             UNIT=SYSDA,
//             SPACE=(TRK,20),
//             DCB=BLKSIZE=400,
//             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)
 //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),     ]
[//             DISP=(KEEP,DELETE)                        ]
[//RELOAD    DD DSN=user.maclib,DISP=(SHR)                ]
 //     PEND
 //STEP1  EXEC MCTCBL[,PARM='options']
 //CARDF DD *[,DCB=BLKSIZE=80]
  [OPTION CA-MetaCOBOL+ translator options]
       *$COPY OPLPS
   source program
         .
         .
         .
 //STEP2 EXEC METACBL[,PARM='options']
 //CARDF DD *[,DCB=BLKSIZE=80]
  [OPTION CA-MetaCOBOL+ translator options]
       *$COPY OPL
 //      DD DSN=*.STEP1.MCT.PUNCHF,DISP=(OLD)
 /*
```

E.3 Translation Under VSE

Note: If you are using CA-MetaCOBOL+/PC, refer to the CA-MetaCOBOL+/PC *User Guide* for equivalent file requirements and SET Statements required for translation.

E.3.1 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: If the SYSIPT installation parameter has been specified, SYSIPT is the Primary Input File. No concatenation is permitted, and you will be unable to translate OPL in a single job. The SYSIPT translate option is required to perform a two-step translation in a single job. Therefore, if you have specified the SYSIPT translate option but have not specified the SYSIPT installation parameter, you will be able to perform a two-step translation in a single job.

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 translate-time 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 translate-time 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.

E.3.2 VSE JCL Example

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  CA-MetaCOBOL+ TRANSLATOR OPTIONS
        *$COPY OPLPS
     SOURCE PROGRAM
          .
          .
          .

  // DLBL IJSYS06
  // EXTENT SYS006,
  // ASSGN SYS006, DISK, VOL=volserno, SHR
  // EXEC METACOB
     OPTION  SYSIPT ,  [CA-MetaCOBOL+ OPTIONS]
        *$COPY OPL
  /*
     CLOSE SYSIPT, SYSRDR
  /*
  /*
```

Index