CA-MetaCOBOL™ +

Online Programming Language Guide

Release 1.1

R203M+11DRP

Release 1.1, January, 1992

Contents

Figures

1. About this Manual

1.1 Purpose

This guide presents the basic OPL functions and describes their use by building an OPL program. This guide should be used in conjunction with the *OPL Reference* manual. Familiarity with COBOL, CICS, and BMS 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 overview of the OPL functions, a brief description of CICS transactions, and the relationship between OPL and CA-METACOBOL+ PDF.
3	Provides information on beginning an OPL program.
4	Provides information on sending and receiving BMS maps.
5	Provides information on invoking a user-written validation procedure and the two methods of OPL validation.
6	Provides information on managing OPL program storage.
7	Provides information on the OPL help facility.

1.3 Publications

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

Title	Contents
Introduction to CA-METACOBOL+	Introduces the CA-METACOBOL+ Work Bench, Structured Programming Facility, Quality Assurance Facility, CA DATACOM/DB Facility, Macro Facility, Panel Definition Facility, 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
Title	**Contents**
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.
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.

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

2. Introduction

CA-METACOBOL+ Online Programming Language (OPL) is an extension to COBOL that greatly simplifies designing and programming for CICS. OPL shields the programmer from the complexities of pseudo-conversational CICS programming.

OPL is a series of high-level, COBOL-like statements. OPL statements are coded in the Identification, Environment, Data, and Procedure Divisions of a program. CA-METACOBOL+ translates these statements into equivalent BMS panel I/O areas and command-level CICS statements. The result of the translation is conventional COBOL. An OPL program can be designed in a fully conversational style. However, the generated COBOL will not incur the processing overhead associated with conversational program execution.

The OPL statements enable you to:

- Define BMS maps. OPL uses conventional BMS maps. The MAP DEFINITION statement defines each map used in the program. CA-METACOBOL+ copies the map definition into the program and uses it to identify map fields. Optionally, an error message field, the name of a help panel, and/or an editing routine can be defined for the entire map and individual fields.

- Send and receive BMS maps using the TRANSMIT statement. An OPL program enables you to treat a terminal like any other I/O device. OPL handles terminal errors, temporary storage control, and other CICS problems related to sending and receiving BMS maps.

  The TRANSMIT statement also simplifies sending and receiving help maps. Since help maps are defined with the MAP DEFINITION statement, OPL can transmit the application panel, detect PF key responses, save screen data, display HELP, return to the application, and redisplay the original panel. OPL can display field help or panel help. All of this is accomplished through OPL definitions and does not require any programming.

  After executing a TRANSMIT statement, the program simply proceeds to the next statement.

- Define user-response handling. The RESPONSE DEFINITION statement specifies what actions take place when specific keys are pressed. When the RESPONSE DEFINITION statement is used in conjunction with the TRANSMIT statement, code is generated to write to the terminal and receive the response.

- Invoke user-written editing procedures using the VALIDATE statement. Validation procedures can be defined for individual fields and the entire map. If panel data is found invalid, OPL can display an error message at the terminal. The program stays within the TRANSMIT statement processing until all panel data is found valid, at which point the program continues. A map-level validation procedure lets the program cross-check data between fields or perform other special processing.

- Manage program storage. OPL provides the WORKING-STORAGE, CONSTANT-STORAGE, and GLOBAL-STORAGE statements, which automatically save and restore program data across transaction boundaries.

Because OPL is an extension to COBOL and command level CICS, your program is not limited by OPL. An OPL program uses native COBOL and CICS services. OPL simply generates COBOL; it leaves your code intact and in its original context.

2.1 CICS Transactions

When a user signs on to a native CICS session, CICS presents a welcome message. The user enters a transaction identifier to start an application. CICS services then starts the program associated with this transaction id. When the program gets control, it does its processing and returns control to CICS. The program can tell CICS what transaction to invoke next. All CICS processing is based on the concept of transactions.

Other CICS facilities enable a program to send a BMS map to the screen, obtain the user's response, and receive data from the screen. A simple CICS program may do little more than send the map and receive data. A more complex CICS program can send the map, receive and validate the user's response, display error messages, transfer control to another application, and more. This manual develops a CICS program that uses OPL statements to accomplish these tasks.

2.2 OPL and CA-METACOBOL+ PDF

The CA-METACOBOL+ Panel Definition Facility (PDF) is a CICS panel painter and BMS map generator. The PDF user designs a panel at his or her terminal and creates a panel definition. PDF uses the panel definition to generate BMS map source, which is then assembled to produce a BMS map. OPL provides a high-level extension to COBOL that enables you to define, send, validate, and receive BMS maps. Thus, PDF and OPL offer a complete CICS programming environment by providing complementary services.

Of course, OPL also supports BMS maps that are not generated by PDF.

3. Getting Started

To demonstrate how an OPL program is created, this manual builds a relatively simple OPL program by presenting a progression of program segments. In each segment, the newly added OPL statements appear in **bold** type for clarity. Only the last program sample will be a complete OPL program. This manual discusses each OPL statement in the context of the sample program. For a complete description of all OPL statements, refer to the *OPL Reference*.

We will start our sample program with the Identification Division. As shown in Figure 1, the program id is SIGNON and the transaction id is `SON1'.

```
     IDENTIFICATION DIVISION.
     PROGRAM-ID.  SIGNON.
     TRANSACTION-ID.  SON1
```

Figure 1. The Identification Division

Next, let's define a BMS map to our program by adding a MAP SECTION. The MAP SECTION, MAP DEFINITION, and NAME IS statements declare that a map is used in this program. The MAP SECTION can be coded in the Data or Environment Division. The map name in this program is MAPB. For simplicity, this example shows the minimal statements required to define a map. The MAP DEFINITION statement has additional operands for defining optional help maps, validation procedures, and message fields. These operands will be discussed later in this guide.

```
     IDENTIFICATION DIVISION.
     PROGRAM-ID.  SIGNON.
     TRANSACTION-ID.  SON1
     *
     ENVIRONMENT DIVISION.
     *
     DATA  DIVISION.
     MAP   SECTION.
     MAP   DEFINITION
           NAME IS MAPB.
```

Figure 2. Defining a Map with the MAP SECTION Statements

Note that OPL will automatically copy the map copy book from the SYSLIB into working storage; therefore a COBOL COPY statement should not be coded.

4. Sending and Receiving Panel Data

Now that we have defined MAPB to our program, we will send and receive it using the TRANSMIT statement. The TRANSMIT statement is an OPL procedural statement. CA-METACOBOL+ will generate the code necessary to present the map, identify the next transaction to CICS, and read the map from the terminal on the next transaction. GOBACK will generate COBOL which, as a default action, clears the terminal and sends the default message `SON1 transaction has completed'. This message may be eliminated or changed to another message if desired.

In our sample, we have defined only one map. If the program defined more than one map, the TRANSMIT statement would include the map name (for example, TRANSMIT MAPC).

```
 IDENTIFICATION DIVISION.
 PROGRAM-ID.  SIGNON.
 TRANSACTION-ID.  SON1
*
ENVIRONMENT DIVISION.
*
 DATA  DIVISION.
 MAP   SECTION.
 MAP   DEFINITION
       NAME IS MAPB.
*
 PROCEDURE DIVISION.
       TRANSMIT.
       GOBACK.
```

Figure 3. Sending and Receiving a Map with the TRANSMIT Statement

OPL will generate the logic to determine whether this is the first pass through the program. On the first pass, any statements preceding the TRANSMIT are executed, and the TRANSMIT sends the map to the terminal. On the next pass, OPL resumes processing the TRANSMIT statement, receives the map, and passes control to the statement after the TRANSMIT.

OPL also generates the COBOL and CICS statements necessary to handle terminal processing and transaction boundary crossing. Much of the complexity of a COBOL and command-level CICS program is required to handle CICS. OPL removes this burden from the programmer.

4.1 Defining Responses to Attention Identifier Keys

Panel data is only sent from a terminal when a user presses an Attention Identifier key, or AID key. The AID keys are ENTER, CLEAR, any PF key, and any PA key. The ENTER and PF keys send back all modified data fields, an indication of which AID key was used, and the cursor position. The CLEAR and PA keys send back only an indication of which key was used.

CICS returns the AID key indicator in the EIBAID Field of the EXEC Interface block. This field can be checked by your program. OPL provides condition-names that correspond to each AID key. When the OPL program is translated, the condition-name generates code to check for a certain AID key. For example, if you code:

```
IF PF3
  statement-1.
```

OPL will generate:

```
IF EIBAID EQUAL DFHPF3
  statement-1.
```

For a list of OPL condition-names and their translations, refer the *OPL Reference*.

The RESPONSE DEFINITION Statement

The RESPONSE DEFINITION statement defines common responses to AID keys used for all TRANSMIT statements in a program. Instead of explicitly coding tests for AID keys, OPL detects which AID key is to be tested and generates COBOL logic to test the appropriate keys in the best location. The RESPONSE DEFINITION actions are executed as though they were embedded within the TRANSMIT. The conditions that can be defined by the RESPONSE DEFINITION are CONTINUE, DISPLAY '*message*', ERASE, GOBACK, HELP, REFRESH, and VALIDATE.

Let's take the previous program sample and add a RESPONSE DEFINITION STATEMENT. In this sample, if the user presses the ENTER key, processing continues with a CICS RECEIVE MAP function and proceeds to the statement following the TRANSMIT. If PF3 is pressed, the program immediately returns control to CICS. Any statements following the TRANSMIT will NOT be executed. If either of the PA keys are pressed, the screen will be refreshed to its state prior to the TRANSMIT.

```
      IDENTIFICATION DIVISION.
      PROGRAM-ID.  SIGNON.
      TRANSACTION-ID.  SON1.
     *
      ENVIRONMENT DIVISION.
     *
      DATA  DIVISION.
      MAP   SECTION.
      MAP   DEFINITION
            NAME IS MAPB.
      RESPONSE DEFINITION
            WHEN ENTER
              CONTINUE
            WHEN PF3
              GOBACK
            WHEN PA1 OR PA2
              REFRESH.
     *
      PROCEDURE DIVISION.
            TRANSMIT.
            GOBACK.
```

Figure 4. The RESPONSE DEFINITION Statement

The RESPONSE DEFINITION statement is optional. If it is omitted, responses default to the following:

Key	**Function**
CLEAR key	returns to CICS
PA keys	refresh the screen
PF keys	initiate the CONTINUE function

The RESPONSE DEFINITION can also be used to simplify validation and help processing. These topics will be discussed later in this guide.

4.2 BMS Maps and Copy Books

CICS provides Basic Mapping Services (BMS) to present data at a terminal, and to move data between a program data area and the terminal display. BMS and the program pass data to each other by using a data area whose format is known to both. When a BMS map definition is assembled, BMS generates both a physical map module and a COBOL copy book. The map copy book defines input and output data areas for each field shown on the screen and contains all the information passed between a program and BMS. A program request for BMS refers to this copy book to insure COBOL and BMS expect the same data layout.

In a conventional CICS program, a COPY statement makes the copy book and, therefore, the data names associated with a BMS map, available to the COBOL compiler. In an OPL program, the MAP DEFINITION automatically makes the copy book available to the program. The copy book is used for both input and output to the terminal. Data placed in this area by the program will be merged with an external map module by BMS when a map is sent. The data returned from a BMS will be placed in this area by BMS when a map is received.

Our program sample uses a map named MAPB. Map MAPB consists of two panel fields, a name and a password. The following figure is the copy book for MAPB.

```
    *COPY MAPB.
     01    MAPBI.
           02    FILLER           PIC   X(12).
           02    NAMEL      PIC  S9(04)      COMP.
           02    NAMEF      PIC   X(01).
           02    FILLER REDEFINES NAMEF.
                 03    NAMEA PIC   X(01).
           02    FILLER           PIC   X(04).
           02    NAMEI      PIC   X(08).
    *
           02    PSWL             PIC  S9(04)      COMP.
           02    PSWF             PIC   X(01).
           02    FILLER REDEFINES PSWF.
                 03    PSWA       PIC   X(01).
           02    FILLER           PIC   X(04).
           02    PSWI             PIC   X(08).
    *
     01    MAPBO REDEFINES MAPBI.
           02    FILLER           PIC   X(12).
           02    FILLER           PIC   X(3).
           02    NAMEC      PIC   X.
           02    NAMEP      PIC   X.
           02    NAMEH      PIC   X.
           02    NAMEV      PIC   X.
           02    NAMEO      PIC   X(08).
           02    FILLER           PIC   X(3)
           02    PSWC             PIC   X.
           02    PSWP             PIC   X.
           02    PSWH             PIC   X.
           02    PSWV             PIC   X.
           02    PSWO             PIC   X(08).
```

Figure 5. Copy Book of MAPB

The map is defined twice in the copy book. The first definition is used when reading the map and is referenced by the name MAPBI (which is the map name suffixed with an 'I'). The second definition is used when writing the map and is referenced by the name MAPBO (which is the map name suffixed with an 'O').

The entire structure can be referenced as either MAPBI, for input from the terminal, or MAPBO, for output. For clarity, however, when referring to screen contents, it is better to refer to fields suffixed with 'I' when using data received from the map. Fields should be suffixed with 'O' when moving data into the map.

Each subordinate data item provides a separate piece of information about the field and its characteristics. For each field, the map provides a series of names based on the BMS field name. BMS restricts field names to 7 characters so it can append one character to the name and still stay within the assembler's 8-character name limit.

4.2.1 BMS Control Fields

BMS provides the COBOL program with the means to control the attribute bytes, cursor placement, and the content of data fields. The location of fields on the terminal is completely controlled in the static map definition and cannot be changed by the program. For each data item that can be changed by the program, BMS provides a COBOL name by using the data item name and adding a 1-character suffix. These names are found in the map copy book.

Attribute Bytes

Copy book names suffixed with 'A' represent hardware attribute bytes (in Figure 5, NAMEA and PSWA). With field data, BMS can obtain the attribute byte from either the program data or the physical map definition. If the program wishes to control the attribute byte, it places a value into the 'A'-suffixed data item. A value of binary zeros tells BMS to use the attribute from the physical map definition instead of the program data. Therefore, an attribute can always be reset to its original value (as specified in the BMS macro) by moving LOW-VALUES into it.

OPL provides the SET ATTRIBUTE statement to make attribute manipulation easier. The SET ATTRIBUTE statement can be used to specify an attribute byte of protected, unprotected, skip, numeric, low, high, invisible, or mdt. We have added two SET ATTRIBUTE statements to the following program sample. When the map is sent to the screen, the name field will be highlighted and the password field will be invisible.

```
     PROCEDURE DIVISION.
          SET ATTRIBUTE HIGH ON NAME.
          SET ATTRIBUTE INVISIBLE ON PSW.
          TRANSMIT.
                    GOBACK.
```

Figure 6. The SET ATTRIBUTE Statement

OPL supports several other SET statements that are not used in our sample program. Refer to the *OPL Reference* for a detailed description of them.

Modified Data Tag

One of the most important attributes is also possibly the most confusing. When the user types data into a field, a hardware flag is set on. This flag is called the Modified Data Tag, or MDT. The MDT is part of the attribute byte. A program can also turn on the MDT by specifying the MDT attribute value, and letting BMS send that attribute to the terminal. Note that ONLY FIELDS WITH MDT SET ON WILL RETURN DATA WHEN A MAP IS RECEIVED.

Since BMS sets the "field" data items to LOW-VALUES before mapping data from the screen, it is possible to lose the data placed in a field. For instance, this could occur on a variable label field where an initial value is moved before a TRANSMIT. Since this is not a field that the user would type over, it would be marked as not modified when the map is received. If the map is sent again, the value for this field has to be moved in again. A program may turn on the MDT to insure that data is always returned for a field, even if the user does not type anything into it. This can be easily accomplished with the 'SET ATTRIBUTE MDT ON *field*' statement.

Since attribute bytes are not returned in the input data stream, these fields are not needed to describe the input data. Instead, BMS redefines the attribute byte fields to be flags. These fields are suffixed with 'F' in the symbolic map definition. BMS will set flags in these copy book fields to let the program test which fields were received.

Determining Field Lengths

Copy book names suffixed with 'L' contain the length of the data stream. After a TRANSMIT, the 'L' field contains the length of data received for that field. If the screen field on the terminal has not been modified by the user (and the modified data tag is off), this field will contain zero to indicate that no data was received. This field will also contain zero if the user erased all data from the field. Otherwise, it contains the number of bytes of data received in that field. All characters, including trailing spaces, are counted in the total.

Controlling Cursor Position

Since the length field is not used when a map is sent, BMS uses a length with the special value of -1 (negative one) as a convention to place the cursor on that field. OPL provides a SET CURSOR statement to control positioning the cursor. We have added a SET CURSOR statement to the following program sample. When MAPB is sent, the cursor will be positioned at the start of the NAME field.

```
     PROCEDURE DIVISION.
          SET ATTRIBUTE HIGH ON NAME.
          SET ATTRIBUTE INVISIBLE ON PSW.
          SET CURSOR ON NAME.
          TRANSMIT.
          GOBACK.
```

Figure 7. The SET CURSOR Statement

5. Validating Panel Data

A validation procedure validates panel data. The actual type of validation is determined by the programmer. For example, the validation procedure might check the type or size of the data, or it might execute a table search. If the panel data is valid, the program continues. If the panel data is invalid, the program might display an error message on the screen. OPL provides statements for:

- Defining and executing a validation procedure
- Indicating that the map is valid or invalid
- Defining a message field
- Writing a message to the screen

OPL supports both non-procedural and procedural panel data validation. Both validation types require at least one validation procedure coded in the program. The following sections discuss these topics.

5.1 Defining a Validation Procedure

Validation procedures are defined to an OPL program using the VALIDATION PROCEDURE operand of the MAP DEFINITION statement. Validation procedures can be defined for a field or for the entire map. In the following figure, we have defined validation procedures for the name and password fields in our sample program. The following example names two validation procedures: NAME-VALIDATION and PASSWORD-VALIDATION.

```
   MAP   DEFINITION
         NAME IS MAPB.
         FIELD NAME
           VALIDATION PROCEDURE IS NAME-VALIDATION
         FIELD PSW
           VALIDATION PROCEDURE IS PASSWORD-VALIDATION
```

Figure 8. The VALIDATION Operand of the MAP DEFINITION Statement

The NAME-VALIDATION procedure is partially shown in the following figure. The validation procedure must indicate to OPL that the map is valid or invalid with the SET MAP statement.

```
   NAME-VALIDATION.
  *
         IF condition...
           SET MAP VALID
           do process...
         ELSE
           SET MAP INVALID
           do other process...
```

Figure 9. Sample Validation Procedure with SET MAP VALID

5.2 Defining Message Fields and Displaying Messages

Now that we have created validation procedures and defined them to our sample program, we need to add the ability to send a message to the user when panel data is found invalid. To do so, use the MESSAGE operand of the MAP DEFINITION statement and the DISPLAY statement.

A message field can be defined for a field or an entire map. In the following example, we have defined two message fields (MSGNAME and MSGPASS).

```
      MAP   DEFINITION
            NAME IS MAPB.
            FIELD NAME
              VALIDATION PROCEDURE IS NAME-VALIDATION
              MESSAGE-FIELD IS MSGNAME
            FIELD PSW
              VALIDATION PROCEDURE IS PASSWORD-VALIDATION
              MESSAGE-FIELD IS MSGPASS.
```

Figure 10. The MESSAGE Operand of the MAP DEFINITION Statement

Note that this example assumes that we have defined the fields MSGNAME and MSGPASS in our BMS map and that the resulting copy book contains all the names necessary for OPL to process these message fields.

Now that we have defined the message fields, we can move data into them with the DISPLAY statement. The DISPLAY statement is part of the validation procedure. When OPL encounters a DISPLAY statement, the message field associated with the field or map that is being validated is initialized before the next map transmission.
In the following example, if the data in the name field is alphabetic, the DISPLAY statement moves spaces into the message field (defined as MSGNAME is the previous figure). If the data is not alphabetic, an error message is moved into the message field.

```
      NAME-VALIDATION.
     *
            IF NAMEI IS ALPHABETIC
              SET MAP VALID
              DISPLAY SPACES
            ELSE
              SET MAP INVALID
              DISPLAY "NAME MUST BE ALPHABETIC".
```

Figure 11. The DISPLAY Statement in a Validation Procedure

5.3 Performing Non-procedural Validation

Non-procedural validation is performed by coding a VALIDATE operand on the RESPONSE DEFINITION statement. The VALIDATE operand is associated with a particular AID key. When that AID key is pressed by the user, processing continues with a RECEIVE MAP function. OPL generates code that performs the following:

- The MAP VALID condition is set
- Field validations are performed in the order specified on the MAP DEFINITION statement.
- Map validation is performed if specified on the MAP DEFINITION statement.
- If an INVALID MAP condition is set by any of the procedures, the map is retransmitted.

It is important to note that an incorrect MAP INVALID condition can cause an infinite loop because the map will continue to be sent until valid. Therefore, it is strongly recommended that a GOBACK or a CONTINUE be explicitly coded when using non-procedural validation.

If we define validation procedures, add a message field, and add a VALIDATE operand to our sample program, we have an OPL program that validates the name and password on a signon screen, as shown in Figure 8. When the user presses the ENTER key, the two validation procedures will be performed. If an invalid map condition exists, the map is sent again with the specified messages appearing in the defined message fields. This process will continue until a valid map condition occurs or until the user presses PF3. (PF3 is defined as GOBACK, which immediately returns control to CICS). The TRANSFER CONTROL statement will be executed only when a valid map condition exists.

```
IDENTIFICATION DIVISION.
PROGRAM-ID.  SIGNON.
TRANSACTION-ID.  SON1.
ENVIRONMENT DIVISION.
DATA  DIVISION.
MAP   SECTION.
MAP   DEFINITION
      NAME IS MAPB
      FIELD NAME
        VALIDATION PROCEDURE IS NAME-VALIDATION
        MESSAGE-FIELD IS MSGNAME
      FIELD PSW
        VALIDATION PROCEDURE IS PASSWORD-VALIDATION
        MESSAGE-FIELD IS MSGPASS.
RESPONSE DEFINITION
      WHEN ENTER
        VALIDATE
      WHEN PF3
        GOBACK
      WHEN PA1 OR PA2
        REFRESH.
PROCEDURE DIVISION.
      SET ATTRIBUTE HIGH ON NAME.
      SET ATTRIBUTE INVISIBLE ON PSW.
      SET CURSOR ON NAME.
      TRANSMIT.
      TRANSFER CONTROL TO TRANSACTION-ID MENU
      GOBACK.
NAME-VALIDATION.
      IF NAMEI IS ALPHABETIC
        SET MAP VALID
        DISPLAY SPACES
      ELSE
        SET MAP INVALID
        DISPLAY "NAME MUST BE ALPHABETIC".
PASSWORD-VALIDATION
      SEARCH ALL SIGNON-ENTRY
      AT END
        SET MAP INVALID
        DISPLAY "INVALID SIGNON ATTEMPT"
      WHEN SIGNON-NAME(INDX) EQUAL NAMEI
        IF SIGNON-PSWD(INDX) EQUAL PSWI
          SET MAP VALID
          DISPLAY SPACES
        ELSE
          SET MAP INVALID
          DISPLAY "PASSWORD INVALID".
```

Figure 12. Program with Non-Procedural Validation

Note that this signon program is not very secure, as it does not limit the number of times the user can enter the password. Procedural validation, described in the next section, can offer more control during validation.

5.4 Performing Procedural Validation

Procedural validation allows the program complete control over which data is validated and the order in which the validation is performed. An OPL VALIDATE statement is used to invoke the procedures named in the MAP DEFINITION. Like non-procedural validation, at least one VALIDATION PROCEDURE must be defined.

The validation procedures should SET MAP VALID or SET MAP INVALID. However, when using the procedural VALIDATE statement, invalid maps will NOT be retransmitted until valid. The program must test the MAP VALID/MAP INVALID condition and explicitly execute a transmit.

There are two formats of the procedural VALIDATE statement: VALIDATE MAP and VALIDATE FIELD. If the VALIDATE MAP statement is coded, any validation procedures defined for fields in that map will be invoked, followed by the validation procedure defined for the map. If the VALIDATE FIELD statement is coded, only validation procedures specified for fields will be invoked.

Let's take our signon program and change it from non-procedural validation to procedural validation. We have changed the VALIDATE statement in the RESPONSE DEFINITION to CONTINUE, added a VALIDATE statement to the procedure division, added code to count the number of signon attempts, and added a WORKING STORAGE section to store the signon count.

In this example, the VALIDATE statement is executed after the user presses the ENTER key, because the RESPONSE DEFINITION specifies CONTINUE as the process for the ENTER key. Any statement following the TRANSMIT gets control when the key defined as continue is pressed. The validation procedures defined in the MAP DEFINITION for fields NAME and PSW are invoked next. Procedural code then checks for a map valid condition. If the map is valid, control is transferred to the next transaction. If the map is invalid, and if there have been three signon attempts, control returns to CICS. Otherwise, we retransmit the map.

For simplicity, the details of searching for the valid password have not been included in this sample program.

```
IDENTIFICATION DIVISION.
PROGRAM-ID.  SIGNON.
TRANSACTION-ID.  SON1.
ENVIRONMENT DIVISION.
DATA  DIVISION.
MAP   SECTION.
MAP   DEFINITION
      NAME IS MAPB
      FIELD NAME
        VALIDATION PROCEDURE IS NAME-VALIDATION
        MESSAGE-FIELD IS MSGNAME
      FIELD PSW
        VALIDATION PROCEDURE IS PASSWORD-VALIDATION
        MESSAGE-FIELD IS MSGPASS.
RESPONSE DEFINITION
      WHEN ENTER
        CONTINUE
      WHEN PF3
        GOBACK
      WHEN PA1 OR PA2
        REFRESH.
WORKING-STORAGE SECTION.
01    SIGNON-COUNT               PIC S9     VALUE +1.
PROCEDURE DIVISION.
      SET ATTRIBUTE HIGH ON NAME.
      SET ATTRIBUTE INVISIBLE ON PSW.
      SET CURSOR ON NAME.
      TRANSMIT.
      VALIDATE FIELDS NAME PSW
      IF MAP-VALID
        TRANSFER CONTROL TO TRANSACTION-ID MENU
      ELSE
        IF SIGNON-COUNT LESS 3
          ADD 1 TO SIGNON-COUNT
          TRANSMIT
        ELSE
          TRANSFER CONTROL TO CICS.
      GOBACK.
NAME-VALIDATION.
      IF NAMEI IS ALPHABETIC
        SET MAP VALID
        DISPLAY SPACES
      ELSE
        SET MAP INVALID
        DISPLAY "NAME MUST BE ALPHABETIC".
```

Figure 13. Program with Procedural Validation (Part 1)

```
     PASSWORD-VALIDATION
          SEARCH ALL SIGNON-ENTRY
          AT END
           SET MAP INVALID
           DISPLAY "INVALID SIGNON ATTEMPT"
          WHEN SIGNON-NAME(INDX) EQUAL NAMEI
           IF SIGNON-PSWD(INDX) EQUAL PSWI
            SET MAP VALID
            DISPLAY SPACES
           ELSE
            SET MAP INVALID
            DISPLAY "PASSWORD INVALID".
```

Figure 14. Program with Procedural Validation (Part 2)

Experienced CICS programmers will probably notice that the counting mechanism for limiting the number of signon attempts will never work. Once a transaction boundary is crossed, all data in WORKING-STORAGE is set to initial values. Therefore, the signon count will always be one! However, OPL automatically saves and restores storage areas, which is the subject of the next chapter.

6. Managing Program Storage

In CICS, program storage is deleted across transaction boundaries. When control is returned to the program, WORKING-STORAGE is initialized. OPL provides three storage statements that automatically save and restore data across transaction boundaries. The following sections describe each storage statement.

6.1 WORKING-STORAGE Processing

WORKING-STORAGE in an OPL program functions as it does in a batch COBOL program. That is, any values placed in WORKING-STORAGE are automatically saved in main storage and restored across transaction boundaries. OPL accomplishes this by defining WORKING-STORAGE as a temporary storage queue defined only to the current program. All WORKING-STORAGE data is combined into a single temporary main storage record. It is written before a TRANSMIT and automatically retrieved after the TRANSMIT. It is automatically deleted by a TRANSFER CONTROL or a GOBACK.

6.2 CONSTANT-STORAGE Processing

For data that remains constant, OPL provides the CONSTANT-STORAGE SECTION, which does not incur the processing overhead normally associated with saving storage. The CONSTANT-STORAGE SECTION contains definitions of constant data, which will not be altered by the program. Data items in this section are placed into the OPL generated WORKING-STORAGE SECTION. This storage is NOT saved during a TRANSMIT. After a TRANSMIT, all values in CONSTANT-STORAGE are set to their initial values. The use of CONSTANT-STORAGE is optional.

6.3 GLOBAL-STORAGE Processing

GLOBAL-STORAGE is based on CICS temporary storage. Data coded in the GLOBAL-STORAGE SECTION is accessible to every program and transaction executed at the same terminal. Data in this area can be passed between programs or accessed by the same program on subsequent calls. This storage is retained until a DELETE GLOBAL-STORAGE statement is executed. The use of GLOBAL-STORAGE is optional.

Each 01 level in the GLOBAL-STORAGE SECTION represents a separate, CICS temporary storage record. Each record is identified by a storage name and terminal ID. In the following example, a date and code need to be passed to another transaction. When an OPL TRANSFER CONTROL statement is executed, a storage record is written. The name of the storage record will be "XY01" concatenated with the terminal ID. This data is available to other OPL programs simply by coding a GLOBAL-STORAGE record to match the STORAGE NAME.

```
     GLOBAL-STORAGE SECTION.
     01   XY01-TRANS-PARM STORAGE NAME IS "XY01".
          05   XY01-PARM-DATE        PIC 9(6).
          05   XY01-PARM-CODE        PIC X(2).
```

Figure 15. The GLOBAL-STORAGE SECTION Statement

In the following figure, we have added GLOBAL-STORAGE and CONSTANT-STORAGE SECTIONs to our sample program. The GLOBAL-STORAGE SECTION is used to pass the signon name to the MENU transaction. The CONSTANT-STORAGE SECTION is used to provide the message text to be displayed.

Note: The message text is no longer supplied directly in the DISPLAY statement.

```
IDENTIFICATION DIVISION.
PROGRAM-ID.  SIGNON.
TRANSACTION-ID.  SON1.
ENVIRONMENT DIVISION.
DATA  DIVISION.
MAP   SECTION.
MAP   DEFINITION
      NAME IS MAPB
      FIELD NAME
        VALIDATION PROCEDURE IS NAME-VALIDATION
        MESSAGE-FIELD IS MSGNAME
      FIELD PSW
        VALIDATION PROCEDURE IS PASSWORD-VALIDATION
        MESSAGE-FIELD IS MSGPASS.
RESPONSE DEFINITION
      WHEN ENTER
        CONTINUE
      WHEN PF3
        GOBACK
      WHEN PA1 OR PA2
        REFRESH.
WORKING-STORAGE SECTION.
01    SIGNON-COUNT              PIC S9     VALUE +1.
GLOBAL-STORAGE SECTION.
01    MENU-TRANSACTION-PARM STORAGE NAME IS "MENU".
      05    MENU-NAME           PIC X(8).
CONSTANT-STORAGE SECTION.
01    NAME-VALIDATION-MSG       PIC X(23) VALUE
                "NAME MUST BE ALPHABETIC".
PROCEDURE DIVISION.
      SET ATTRIBUTE HIGH ON NAME.
      SET ATTRIBUTE INVISIBLE ON PSW.
      SET CURSOR ON NAME.
      TRANSMIT.
      VALIDATE FIELDS NAME PSW
      IF MAP-VALID
        MOVE NAMEI TO MENU-NAME
        TRANSFER CONTROL TO TRANSACTION-ID MENU
      ELSE
        IF SIGNON-COUNT LESS 3
          ADD 1 TO SIGNON-COUNT
          TRANSMIT
        ELSE
          TRANSFER CONTROL TO CICS.
      GOBACK.
```

Figure 16. Program with GLOBAL-STORAGE and CONSTANT-STORAGE SECTIONs (Part 1)

```
     NAME-VALIDATION.
          IF NAMEI IS ALPHABETIC
            SET MAP VALID
            DISPLAY SPACES
          ELSE
            SET MAP INVALID
            DISPLAY NAME-VALIDATION-MSG.
     PASSWORD-VALIDATION
          SEARCH ALL SIGNON-ENTRY
          AT END
            SET MAP INVALID
            DISPLAY "INVALID SIGNON ATTEMPT"
          WHEN SIGNON-NAME(INDX) EQUAL NAMEI
            IF SIGNON-PSWD(INDX) EQUAL PSWI
              SET MAP VALID
              DISPLAY SPACES
            ELSE
              SET MAP INVALID
              DISPLAY "PASSWORD INVALID".
```

Figure 17. Program with GLOBAL-STORAGE and CONSTANT-STORAGE SECTIONs (Part 2)

7. The Help Facility

Help screens provide users with information to enable them to successfully use an application. Help screens are optional. In conjunction with the TRANSMIT statement, OPL can generate code to automatically display help screens. Help maps are defined on the MAP DEFINITION statement for the map that they are associated with. They can be associated with either a single field or an entire map.

An AID key must be defined as the help key. This is specified on the RESPONSE DEFINITION statement. In our sample program, we have added definitions for screen help, added field help for the name and password fields, and defined PF1 as the help key. The commands for these features are shown in Figure 18.

```
IDENTIFICATION DIVISION.
PROGRAM-ID.  SIGNON.
TRANSACTION-ID.  SON1.
ENVIRONMENT DIVISION.
DATA  DIVISION.
MAP   SECTION.
MAP   DEFINITION
      NAME IS MAPB
        HELP IS SIGNH01
      FIELD NAME
        HELP IS SIGNH02
        VALIDATION PROCEDURE IS NAME-VALIDATION
        MESSAGE-FIELD IS MSGNAME
      FIELD PSW
        HELP IS SIGNH03
        VALIDATION PROCEDURE IS PASSWORD-VALIDATION
        MESSAGE-FIELD IS MSGPASS.
RESPONSE DEFINITION
      WHEN ENTER
        CONTINUE
      WHEN PF1
        HELP
      WHEN PF3
        GOBACK
      WHEN PA1 OR PA2
        REFRESH.
WORKING-STORAGE SECTION.
01    SIGNON-COUNT               PIC S9     VALUE +1.
GLOBAL-STORAGE SECTION.
01    MENU-TRANSACTION-PARM STORAGE NAME IS "MENU"
      05   MENU-NAME             PIC X(8).
CONSTANT-STORAGE SECTION.
01    NAME-VALIDATION-MSG        PIC X(23) VALUE
                 "NAME MUST BE ALPHABETIC".
PROCEDURE DIVISION.
      SET ATTRIBUTE HIGH ON NAME.
      SET ATTRIBUTE INVISIBLE ON PSW.
      SET CURSOR ON NAME.
      TRANSMIT.
      VALIDATE FIELDS NAME PSW
      IF MAP-VALID
        MOVE NAMEI TO MENU-NAME
        TRANSFER CONTROL TO TRANSACTION-ID MENU
      ELSE
        IF SIGNON-COUNT LESS 3
          ADD 1 TO SIGNON-COUNT
          TRANSMIT
        ELSE
          TRANSFER CONTROL TO CICS.
      GOBACK.
```

Figure 18. Program Defining Help Screens (Part 1)

```
     NAME-VALIDATION.
          IF NAMEI IS ALPHABETIC
           SET MAP VALID
           DISPLAY SPACES
          ELSE
           SET MAP INVALID
           DISPLAY NAME-VALIDATION-MSG.    PASSWORD-VALIDATION
          SEARCH ALL SIGNON-ENTRY
          AT END
           SET MAP INVALID
           DISPLAY "INVALID SIGNON ATTEMPT"
          WHEN SIGNON-NAME(INDX) EQUAL NAMEI
           IF SIGNON-PSWD(INDX) EQUAL PSWI
            SET MAP VALID
            DISPLAY SPACES
           ELSE
            SET MAP INVALID
            DISPLAY "PASSWORD INVALID".
```

Figure 19. Program Defining Help Screens (Part 2)

After MAPB is transmitted, if the user presses the PF1 key and the cursor is not on either of the data fields (NAME or PSW), help map SIGNH01 is sent to the terminal. If the cursor is on the NAME field and PF1 is pressed, help map SIGNH02 is sent. If the cursor is on the PSW field and PF1 is pressed, help map SIGNHO3 is sent.

After a help map is sent, pressing the key defined as GOBACK or HELP RETURN-KEY will resend the map as it was when help was requested. (See the RESPONSE DEFINITION statement in the *OPL Reference* for details on the HELP RETURN-KEY.) Pressing any other key will redisplay the help map.

By default, a help map will overlay the currently displayed map. Help maps can be made smaller than the current map so they can be used as pop-up help windows. The ERASE operand of the MAP DEFINITION statement can be used to clear the current map and display only the help map.

Index