SC33-0086-1

Customer Information Control System/Virtual Storage (CICS/VS) Version 1 Release 5

Entry Level System User's Guide (DOS/VS)

Program Product

Program Number 5746-XX3 (CICS/DOS/VS)

| Second Edition (May 1980)

| This edition applies to Version 1 Release 5 (Version 1.5) of the IBM
| program product Customer Information Control System/Virtual Storage
| (CICS/VS), program numbers 5746-XX3 (for DOS/VS) and 5740-XX1 (for
| OS/VS). Until the OS/VS version is released, the information applicable
| to that version is for planning purposes only.

This edition is based on the CICS/VS Version 1.4.1 edition, and changes from that edition are indicated by vertical lines to the left of the changes. Note, however, that the 1.4.1 edition remains current and applicable for users of Version 1.4.1 of CICS/VS.

Information in this publication is subject to change. Changes will be published in new editions or technical newsletters. Before using this
| publication, consult the latest IBM System/370 and 4300 Processors
| Bibliography, GC20-0001, to learn which editions and technical
| newsletters are current and applicable.

It is possible that this material may contain references to, or information about, IBM products (machines and programs), programming, or services that are not announced in your country. Such references or information must not be construed to mean that IBM intends to announce such IBM products, programming, or services in your country.

Publications are not stocked at the addresses given below; requests for copies of IBM publications should be made to your IBM representative or to the IBM branch office serving your locality.

A form for reader's comments is provided at the back of this publication; if the form has been removed, comments may be addressed either to:

| International Business Machines Corporation,
| Department 812HP,
| 1133 Westchester Avenue,
| White Plains, New York 10604.

| or to:

| IBM United Kingdom Laboratories Limited,
| Programming Publications, Mail Point 095,
| Hursley Park,
| Winchester, Hampshire SO21 2JN, England.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation whatever. You may, of course, continue to use the information you supply.

Preface

This publication is intended for users of the CICS/DOS/VS Entry Level System (CICS/DOS/VS-ELS). Its purpose is to provide all of the information normally needed to design, install, and service the system, and to write and run CICS/DOS/VS-ELS application programs.

CICS/DOS/VS-ELS is a subset of CICS/DOS/VS. It requires an IBM 4300 or IBM 370 processor operating under VSE/Advanced Functions Release 2, supporting the IBM 3270 information display system.

Data files for use with the entry level system may be organized using the Indexed Sequential Access Method (ISAM) or the Virtual Storage Access Method (VSAM).

In this publication, the term VTAM refers to ACF/VTAM or to ACF/VTAME. The term BTAM refers to BTAM-ES. For further details of system requirements, refer to the publication CICS/VS General Information.

The reader is assumed to have a basic knowledge of VSE batch processing and some programming experience in one of the four languages supported, which are:

- Assembler Language
- COBOL
- PL/I
- RPG II

ORGANIZATION OF THIS MANUAL

The information in this manual is organized into four parts and five appendixes, as follows:

- Part 1 is an introduction to data communication systems in general, and to CICS/DOS/VS-ELS in particular. It also defines the syntax notation used in the rest of the manual.
- Part 2 covers application programming in five chapters. The first chapter describes the CICS/VS functions available to the entry level system user; it is intended to be used as an application design guide at first, and later as a source of reference to CICS/VS commands during application programming. The remaining chapters give information specific to each of the four programming languages supported, such as translator options, command syntax, and a sample program written in the particular language.
- Part 3 covers system programming in eight chapters. It describes how to install the system as supplied by IBM, how to run the sample programs, and how to tailor the system to the requirements of the user's installation. Also included in this part are chapters on system execution (for example, operator procedures), servicing, and performance considerations.

- Part 4 contains two chapters describing the facilities available for program checkout. The first chapter describes a comprehensive online diagnostic aid, and the second provides an introduction to trace and dump facilities.

- The appendixes are:

  - Appendix A, which is a summary of the program libraries supplied for the entry level system, and their contents.

  - Appendix B describes the fields of the EXEC interface block, which is a control block containing current information relating to the interface between the application program and the system. The fields can be read by any application program.

  - Appendix C provides system programming information for use when CICS/DOS/VS-ELS is running under the Virtual Telecommunications Access Method (VTAM).

  - Appendix D gives information relating to the printer authorization matrix that can be defined for the IBM 3274 control unit.

  - Appendix E describes how to load programmed symbol sets into program symbol store.

  - Appendix F briefly describes the Facility Error Recognition System (FERS).

  - Appendix G is a guide to transferring from the entry level system to full CICS/DOS/VS.

A bibliography follows Appendix G.

Contents

Figures

Summary of Amendments for Version 1 Release 5

This publication supersedes, for CICS/VS Version 1 Release 5, the CICS/VS Entry Level System User's Guide, SC33-0086-0. The Guide has been modified to accommodate the following changes to CICS/VS introduced in Version 1 Release 5:

- Command Syntax Checker

  A new transaction (CECS) enables an application programmer to check the syntax of CICS/VS commands.

- New Master Terminal Transaction

  Enhanced Master Terminal, Supervisory Terminal, and Ordinary Terminal transactions replace the existing transactions. The new transaction identifiers are CEMT, CEST and CEOT respectively.

- Extended Data Stream

  CICS/VS 1.5 uses an extended data stream to communicate with devices which can use COLOR, EXTENDED HIGHLIGHTING and PROGRAMMED SYMBOLS when displaying application data.

- DL/I High Level Programming Interface

  A new set of commands allows COBOL and PL/I programmers to process DL/I data bases.

- The Facility Error Recognition System (FERS)

  FERS, an integral part of CICS/VS 1.5, collects communication error data to facilitate communication system fault diagnosis.

Part 1. Introduction

Chapter 1.1. Introduction

Introduction to Data Communication

In the conventional batch processing environment, application programs are usually scheduled individually to process a batch of data records in a physical or logical sequence. Data files in such environments tend to be organized to suit the requirements of a particular application.

The real-time data communication environment differs from the batch processing environment primarily in the number and types of activities that are likely to occur concurrently within the system. A data communication system controls many requests arriving randomly, and data files tend to be organized so that many application programs can access the same data. If this type of file organization is used extensively in a data communication system, it is called a data base/data communication (DB/DC) system.

Some of the characteristics of data communication systems are:

- A data communication system is terminal-oriented. Application programs executed under the control of such a system receive input entered from terminals connected to the computer, and direct output to the same or different terminals, as well as transmitting data to and from files.
- A data communication system is capable of handling two or more application programs concurrently.
- A data communication system allows different application programs to access the same data base.

Introduction to the Entry Level System

CICS/DOS/VS-ELS (or the "entry level system," a synonym used in this manual) is a subset of CICS/VS designed for ease of installation and use. Although it has a limited set of functions, these functions have been carefully chosen for their wide range of applications. This subset is compatible with the larger CICS/DOS/VS and CICS/OS/VS systems. Little or no change need be made to an application program to enable it to run on the larger systems.

CICS/VS is an interface between the customer's application programs and the operating system, designed to enable the customer to build and operate a DB/DC system that is tailored to the requirements of a particular installation.

CICS/VS provides commands for the application programmer to use in the program to request services such as reading and writing files and controlling terminals. When these commands are executed, CICS/VS passes the requests to the operating system. In this way, the application programmer is relieved of the difficult task of providing such services on the random basis that a terminal-oriented system demands. The commands can be written to any of four syntaxes, which conform to those of the four programming languages supported.

CICS/DOS/VS Version 1.5 operates under VSE/Advanced Functions Release 2, and can be regarded as an extension of this operating system.

According to the configuration of his system, a CICS/VS user might require additional licensed programs. For details of these programs see the CICS/VS General Information Manual.

CICS/DOS/VS is executed as one job, either in a dedicated mode with no other partition operating, or in a multiprogramming mode with one or more batch partitions.

Within its partition, CICS/DOS/VS can control the concurrent processing of input from and output to many terminals by many application programs. The terminal operator invokes an application program by entering a transaction identifier at the terminal; this causes CICS/DOS/VS to attach a task to execute the program. The system can handle more than one transaction at a time by multitasking, that is, by overlapping input/output operations and processing.

Since it resides in a partition with a high priority, CICS/DOS/VS retains system control in a multiprogramming environment as long as there are transactions to be serviced. It relinquishes control to VSE only when it has no further processing to do.

CICS/DOS/VS takes advantage of operating system services; for example, it uses BTAM or VTAM for terminal input/output, and standard file access methods, such as DAM, ISAM, and VSAM. (The application programmer may not specify any files as DAM, but the system may use it for certain operations.)

The functions of the entry level system are described in Chapter 2.1, which readers are advised to survey briefly before familiarizing themselves with the chapter in Part 2 that is specific to their programming language, returning to Chapter 2.1 later for a closer examination of the system functions.

Further introductory information about the system is given in Chapter 3.1.

Syntax Notation Used in this Manual

The syntax notation used in this manual is as follows:

- Braces indicate that from a set of alternatives, one must be coded. A vertical bar separates the alternatives. For example:

  ```
  {TIME(data-value)|INTERVAL(data-value)}
  ```

- Brackets indicate that the enclosed is optional. For example:

  ```
  [FREEKB][FRSET][ALARM]
  ```

- Uppercase entries must be used exactly as shown.

- Lowercase entries must be replaced by a suitable parameter.

- Defaults are indicated by an underscore.

- "data-value" can be replaced by any element (that is, a data name or variable) of the correct data type, or by a constant that can be converted to the correct type. The data type required can be:

  - halfword binary
  - fullword binary
  - packed decimal

- "data-area" can be replaced by any element, but not a constant, of the correct data type.
- "pointer-ref" can be replaced by the name of any data pointer (or, for Assembler language, by a reference to a register).
- "name" or "identifier" (for example, file-name and queue-identifer) can be replaced by:
  - A literal constant.
  - An element containing a character string of length equal to the maximum length allowed for the name. The name must be padded with blanks to provide a character string of the length required.
- "label" can be replaced by any program label (or paragraph or section name). For assembler language, it can also be replaced by a reference to storage containing the address value of the label.
- Parentheses ( ) act as delimiters. They must be coded unless the accompanying text states otherwise.
- An ellipsis (...) indicates that the preceding element in the statement can be repeated. For example:

```
SERVREQ = (request [,...])
```

indicates that SERVREQ can be used to define one or more service requests. Multiple service requests are made by coding a list of keywords, separated by commas.

Part 2. Application Programming

Chapter 2.1. Application Design

This chapter is designed to serve both as a guide to the facilities offered by ELS for application programming and as a reference source to the associated commands.

Each section describes a set of related functions, and the CICS/VS commands that invoke them, classified as follows:

Command level interface
— the means by which CICS/VS accepts instructions written to a syntax similar to that of the application programming language

Terminal control
— screen layout and input/output to and from terminals

Access to system information
— use of CICS/VS storage areas to pass information between application programs

Access to data bases
— CICS/VS relieves the application programmer of many data management tasks.

File control
— direct-access processing of ISAM and VSAM files

Transient data control
— general queuing of data inside and outside the CICS/VS partition; sequential input/output

Temporary storage control
— storing of data in virtual storage for later retrieval by any program (a "scratchpad" facility)

Program control
— transfer of control, with or without accompanying data, between programs

Interval control
— initiating transactions (that is, starting tasks that execute application programs) according to time of day or elapsed time

Exceptional condition handling
— overriding the system default actions for exceptional conditions

Each section gives a general description of the related set of functions, followed by a description of the associated commands (with the general syntax of each), then a list of the command options for the set, and finally a list, if applicable, of the exceptional conditions that can occur during the execution of the commands. The general syntax of the commands is presented in a way that does not relate to a particular programming language; the language-specific sections of the manual show how to interpret this syntax in terms of the particular language.

The CICS/VS commands described are generally used in place of the corresponding functions in the application programming language. For

example, programmers must not use input/output functions from their own language; they must use the CICS/VS commands.

USE OF EXAMPLES IN THIS CHAPTER

Because of the differences in syntax between the programming languages supported, the examples in this chapter are shown in a generalized form that does not relate to a specific language. (For details of the syntax of CICS/VS commands in the different languages, see the language-specific chapters later in this part.) The generalized form is as follows:

1. Only CICS/VS commands are shown formally in upper case.
2. All other essential instructions are set out informally in upper and lower case English. (The sample programs supplied in each of the languages contain detailed examples of the features described in this section, in the appropriate language context.)

CICS/VS Commands

The coded form for CICS/VS commands in the examples in this section consists of:

```
EXEC CICS command-keyword [options]
```

Thus, if the command

```
EXEC CICS RETURN TRANSID ('AMNU')
```

appeared in an example, it would represent the following in the different languages:

Assembler

```
EXEC CICS RETURN TRANSID('AMNU')
```

COBOL

```
EXEC CICS RETURN TRANSID('AMNU') END-EXEC
```

PL/I

```
EXEC CICS RETURN TRANSID('AMNU');
```

RPG II (in a calculation specification)

```
Column          1         2    3
                8         8    3
-------------------------------------------------------------
                RETURN    EXEC
                TRANSID   ELEM 'AMNU'
```

Other Instructions

Essential instructions other than CICS/VS commands are set out in English. For example:

Copy the value in ACCTNO to KEYF.

would represent the following in the different languages:

Assembler

```
MVC KEYF,ACCTNO
```

COBOL

```
MOVE ACCTNO TO KEYF.
```

PL/I

```
KEYF = ACCTNO;
```

RPG II (in a calculation specification, assuming field length 5)

```
Column          1         2    3         4        5
                8         8    3         3        1
-------------------------------------------------------
                          MOVE ACCTNO    KEYF     5
```

Command Level Interface

The command level interface consists of the command language translator and the EXEC interface program.

The command language translator accepts as input a source program, written in one of four languages, in which CICS/VS commands have been coded. The translator produces as output an equivalent source program in which the CICS/VS commands have been translated into CALL statements. At execution time, the CALL statements invoke the EXEC interface program, which accepts the arguments passed in the CALL statements, sets up the parameters in the CICS/VS control blocks, and passes control to the appropriate CICS/VS facility. The four languages supported are assembler language, COBOL, PL/I, and RPG II.

The translator is executed in a separate job step. The job step sequence for preparing an application program is translate - compile or assemble - link-edit. Cataloged procedures are supplied to assist the user; for details, see Chapter 3.5, "Preparation of Application Programs." The translator requires a partition of 64K bytes except for RPG II, which requires 128K bytes.

The translator reads its input from SYSIPT, produces its output (the translated source program) on SYSPCH (or, for RPG II only, optionally on SYS003), and writes the source listing, error messages and so on, on SYSLST. The translator will reject functions that are not supported by the entry level system.

There are four separate translators, one for each language.

TRANSLATOR DATA SETS

Three translator data sets (files) are required: one for source input, one for translated output, and one for listings.

Input and Output Data Sets

The input and output data sets must be sequential. They may be on punched cards, a direct-access device (or devices), or magnetic tape. They must contain 80-byte fixed-length unblocked records.

Listing Data Set

The listing data set must be sequential. Although the listing is usually printed, it can be stored on any magnetic tape or direct access device. It must contain 121-byte fixed-length, unblocked records.

Optional Facilities

The translator provides a number of optional facilities, for example, to allow for different record formats and to specify what information is required on the listing. There are different sets of options for the different languages; for details, see the section appropriate to the language you are using.

Command Syntax Checker

CICS/VS-ELS supports a command syntax checker. To invoke the checker, an operator types the transaction code CECS. The transaction responds by displaying a menu panel which contains a list of CICS/VS commands. When the operator types one of the listed keywords onto the command line and presses ENTER, the interpreter displays the full syntax of the command.

For example, selecting the keyword ABEND produces the following display:

```
?ABEND
  STATUS: COMMAND SYNTAX CHECK

  EXEC CICS ABEND
      <ABCODE()>
      <CANCEL>
```

Note: The brackets < > indicate optional parts of the command, and brackets ( ) indicate data to be supplied (if they are empty), or existing data (if they contain text).

Program Function Keys

When an operator uses the command syntax checker, he can use program function (PF) keys to perform a variety of operations. The bottom line of the display contains a list of the keys. If the terminal has no PF keys, key pressing can be simulated by positioning the cursor under the key name within this list and pressing ENTER. The keys are as follows:

PF1: END SESSION
ends the current session of the command interpreter.

PF2: RE-DISPLAY
re-displays the current screen to enable confirmation that a change entered on the screen has in fact taken place as expected.

PF3: SWITCH HEX/CHAR
switches the display between hexadecimal and character representation. This is a mode switch; all subsequent displays will stay in the chosen mode until the next time this key is pressed.

PF4: EIB DISPLAY
shows the contents of the EXEC interface block (EIB). (See Appendix A for a description of the fields in the EIB).

PF5: VARIABLES
shows all the variables associated with the current command interpreter session, giving for each its name, length, and value.

PF6: USER DISPLAY
shows what the user would see if the terminal had been executing a transaction which contained the commands which have been executed using the interpreter.

PF7: SCROLL UP HALF
scrolls half a screenful upwards.

PF8: SCROLL DOWN HALF
scrolls half a screenful downwards.

PF9: EXPAND MESSAGES
shows all the messages generated during the syntax check of a command.

PF10: SCROLL UP
scrolls one screenful upwards.

PF11: SCROLL DOWN
scrolls one screenful downwards.

PF12: UNDEFINED
means that this key is not available with this type of display.

Terminal Control

CICS/VS-ELS supports the IBM 3270 range of displays and printers. This discussion covers the elements of display layout design, then the mechanism by which the programmer specifies the selected screen layouts to CICS/VS (that is, Basic Mapping Support (BMS)), and finally the terminal control commands and options available under ELS.

THE IBM 3270 INFORMATION DISPLAY SYSTEM

This section describes 3270 input and output operations, screen design considerations, and 3270 field concepts.

The 3270 screen layouts may be designed using the form IBM 3270 Information Display System Layout Sheet (GX27-2951).

3270 Input Operations

The terminal operator keys the transaction code and any required input data on the 3270 keyboard. Where applicable, any keyed-in data is immediately displayed on the screen for sight verification. An underscore character (cursor) indicates the screen position of the next character to be entered. The operator may use any of the following 3270 features to enter data or correct errors:

- Typamatic keys
- Forward and backward tabbing keys
- New line tabbing key
- Horizontal cursor positioning keys
- Vertical cursor positioning keys
- Backspace key
- Erase input key
- Erase end-of-field key
- Character insertion and deletion keys

These and other 3270 features are described in the Operator's Guide for the IBM 3270 Information Display System.

When the data has been correctly entered on the screen, the terminal operator may transmit it to the processing unit by using one of the 3270 input keys, the selector light pen, the cursor select key, or the operator identification card reader attached to the 3270. The following 3270 input keys are available, depending on the keyboard ordered:

- Enter key
- Clear key
- Test request key
- Cursor select key
- Program function keys (PF1-PF24) or (PF1-PF12)
- Program access keys (PA1-PA3) or (PA1-PA2).

An attention identification (AID) character is always transmitted to the processing unit for 3270 input operations. However, input data is transmitted to the processing unit only when the enter key, program function (PF) keys, or the identification card reader is used. If the selector pen or cursor select key is used, the screen locations of selected fields without data are transmitted, provided that the fields

are selection fields. (Further information on selector pen fields is given in Chapter 3.5 under the ATTRB=DET operand of the DFHMDF macro.) When a program access (PA) key or the clear key is used, no data is transmitted. The test request key is not used by CICS/VS.

The CICS/VS terminal control program reads the data entered by the operator and makes it available to the requested program. It also indicates to the application program which input key was used to transmit the data. This AID character is made available in a field called EIBAID (or, for RPG II, EAID). Alternatively if, prior to the terminal read, the application program has set up an exit routine to trap occurrences of the input key being depressed, control is transferred to that routine.

To help the programmer identify the AID character, CICS/VS has provided a set of constants, under the name DFHAID, that may be copied into a user program. For further details see under "HANDLE AID Command," later in this chapter.

To reduce transmission time and improve throughput, the 3270 transmits only modified data fields and suppresses nulls (X'00'). As a result, the input data stream is not in a form that can be conveniently handled by the application program. CICS/VS basic mapping support (discussed later in this chapter) converts the data stream to a convenient form.

3270 Output Operations

The application program processes the input data, prepares a response to send to the 3270 terminal, and specifies whether or not the 3270 screen is to be erased before the data is sent.

If the program does not specify ERASE, the output data overlays a predetermined portion of the screen without changing the remainder of the data on the screen. The operator can then view data sent to the terminal at different times by different write operations.

This overlapping of output data on the screen can be used to guide the operator with special messages without destroying the application data being viewed. However, care must be taken in designing the screen layouts to avoid conflicting use of the same area of the screen. The last few lines at the bottom of the screen are often reserved for operator messages.

After reformatting the output data, BMS uses the terminal control program to send the data to the terminal. The application program may also send special control characters to the 3270 to activate the following features:

- Sounding the audible alarm at the terminal (special feature)
- Unlocking the keyboard for data input
- Resetting the modified status of each field
- Printing the contents of a screen
- Erasing all unprotected fields

SCREEN LAYOUT DESIGN

The unique features of the 3270 system allow screen layouts to be designed for operator convenience and efficiency. The success of an online system depends on its ease-of-use, screen clarity, and terminal operator acceptance.

The following features of some IBM 3270 displays make it easier for the layout designer to fulfil the requirements:

- color
- field highlighting
- programmed symbols
- easy correction
- numeric shift for numeric data
- validation
- field delimiters or stoppers (to control the length of data entered)

The first step in designing 3270 screen layouts is to divide the screen into functional areas such as a title area, an application data area, and an operator message area.

Title Area

The title area of a screen should identify the program that displayed the data. Data fields from the same file can appear in the same screen locations for different applications, permitting the operator to become familiar with fields by their screen location. A title can be used to help the operator recognize the application. The title area is normally the top one or two lines of the screen and may contain a page number, if multiple pages are needed, field headings, and other information besides the title.

Application Data Area

The application data area comprises the main portion of the screen. Data from one or more records in the same file or multiple files is entered or displayed, depending on the application requirements.

Three types of fields are usually found in this area: keyword, data, and stopper fields.

The keyword fields contain constant information sent by the program to identify the contents of a data field. For example, a keyword field containing "ACCOUNT BALANCE:" might precede and identify a data field containing "$129.54". A keyword field might also be used in a data entry application to identify the data being entered. For example,

QUANTITY:
        means enter the quantity.

The data fields contain file data that the application program retrieves from files and displays. The data may appear exactly as stored in the file, or it may be edited by the program. Data fields may also be left blank for the operator to enter data. The application program can use the entered data to make changes to a record or to alter the processing of the program. In some cases, it may be appropriate for

the program to display characters in an entry data field to guide the operator in entering the data. For example,

DATE: MMDDYY
means enter month, day, year, each having two characters.

Stopper fields (see "Attribute Character" in the section "3270 Field Concepts") on data entry screens restrict the length of the data fields. Stopper fields containing no data are used to define the space between data fields and to stop the operator from entering too many characters in a field. For example, a field containing a street address may be 20 characters long, but for screen layout reasons an entire line of 40 characters is provided for this field. To prevent the operator from keying more than 20 characters on this line, the program should define a stopper field starting in the twenty-first position of the line. The stopper field should be protected from data entry to restrict the operator to the 20-character field.

Message Area

The message area of a screen is used to send instruction messages to assist the operator in processing a transaction. This area should be separate from the application data area to allow communication with the operator, without disturbing the application data. The message area is normally the bottom one or two lines of the screen.

3270 FIELD CONCEPTS

The 3270 is available in several screen sizes (see "Screen Sizes" later in this chapter). A program can divide these screens into multiple fields. The fields combine to produce a complete screenful of data, an 'image'.

A field starts with an attribute character, continues with data characters, and ends at the next attribute character. A field may contain only 1 character or it may span several lines, since the last character on a line is logically followed by the first character on the next line. Basic mapping support limits a field to 256 bytes and does not allow a field to extend beyond the bottom of the screen, "wrapping" the screen.

Normally, an image is divided into several fields by the program but it is possible to have an image with no fields (no attribute characters). This occurs when the operator presses the clear key. Such unformatted images are not supported by basic mapping support and are not described in this manual. An application program can use the HANDLE AID command, described later in this section, to detect the use of the clear key; also, an attempt to read from a cleared screen raises the MAPFAIL condition.

Attribute Character

The attribute character is always the first character of a field. It occupies a character position on the screen but appears as a blank. An extended data stream is used to communicate with a device which supports extended color, highlighting, programmed symbols or validation. The single blank attribute character produced by such a data stream can represent several attribute bytes.

Attribute bytes can convey the following field attributes.

- Unprotected

  In an unprotected field, the operator may enter any keyboard character.

- Numeric only

  A numeric only field is unprotected and only the numeric characters 0 through 9 and the special characters period, dash, and "DUP" may be entered. If the keyboard numeric lock feature is installed on the 3270 and the operator attempts to enter any other characters, the keyboard is locked. If the keyboard numeric lock feature is not installed, any data can be entered in the field. On a data entry keyboard, a numeric-only field causes a numeric shift to occur.

- Protected

  Data cannot be entered in a protected field. If the operator attempts to enter data, the keyboard is locked. Stopper fields following variable-length data fields are normally defined with protected attribute characters. If the operator attempts to enter more characters than the variable-length data field can contain, the stopper field following it will cause the keyboard to be locked.

- Autoskip

  An autoskip field is a protected field which automatically skips the cursor to the next unprotected field. Keyword fields and stopper fields following fixed-length data fields are normally defined with autoskip attribute characters.

  Note: The unprotected, numeric only, protected, and autoskip characteristics of the attribute character are mutually exclusive. Only one may be selected for each field.

- Normal intensity

  A normal intensity field displays the data at the normal operating intensity.

- Bright intensity

  A bright intensity field displays the data at a brighter than normal intensity. This is often used to highlight keywords, errors, or operator messages.

- Base color

  The IBM 3279 model 2A or 3A display produces a 'base color' image by using the PROTECT and INTENSIFY attributes of the IBM 3270 standard data stream to select four colors: white, red, blue, and green. A switch on the display control panel permits the operator to select default color, causing the display to behave as a monochrome 3270 display, with WHITE representing INTENSIFY. The 'protect' bit retains its protect function when conveying color information.

- Extended color

  The IBM 3279 model 2B or model 3B uses extended color attributes in an extended data stream to determine the colors of display elements. The data stream can specify the colors of multi-character fields. Seven colors can be selected. They are blue, red, pink, green, turquoise, yellow, and neutral.

  An IBM 3279 Modesl 2B or 3B will act as a Model 2A or 3A until it detects an extended color attribute byte in the data stream. It will display the image in default color or base color, according to the setting of the switch on the control panel.

  As soon as an extended color attribute is received, the display treats the whole image as an extended color image. Fields which have no color attribute adopt the default colors (green for normal intensity, white for bright). If the color control switch has been set to 'base color', the part of the image which has already been displayed will change from base color to default color. Such a change, which could disturb an operator, can be avoided by applying an extended color attribute to the first field in any image which uses extended color.

  The device interprets extended color attributes to determine the colors of fields in an image.

- Extended highlighting

  Extended highlighting can be applied to characters, or character fields, in a display which uses the extended data stream. It can take one of three forms: REVERSE VIDEO, BLINK, or UNDERLINE.

- Nondisplay

  A nondisplay field does not display the data on the screen for operator viewing and does not print the field data. This might be used to enter security information when the screen is visible to others. This attribute characteristic should be used with care, as the operator loses the ability to verify the data entered in a nondisplay field. This field might also be used to store messages on the screen. The messages can be displayed later by changing the attribute character to bright or normal intensity. The 'protect' bit retains its protect function while conveying color information.

  Note: The normal, bright, and nondisplay characteristics of the attribute character are mutually exclusive. Only one may be selected for each field.

- Programmed symbols

  As well as the standard display symbol sets, the IBM 3278 and the IBM 3279 Model 2B or 3B can have optional additional symbol store, enabling them to display up to six 191-character symbol sets, whose fonts and codes are defined by the user. Characters in different display fields can be selected from different symbol sets. This feature uses the extended data stream.

  Appendix E describes how to load Symbol Sets.

- Selector pen detectable

  A selector pen detectable field is sensitive to the selector pen (special feature) and the cursor select key. Two types of detectable fields are possible: a delayed detectable field and an immediately detectable field.

  If a delayed detectable field is selected by the operator using the selector pen, the modified-data tag (MDT) is turned on. If an immediately detectable field is selected, the modified-data tag is turned on and transmission occurs. Further information on selector pen fields is given in Chapter 3.5 under the ATTRB=DET operand of the DFHMDF macro. The publication IBM 3270 Information Display System Component Description describes the use of detectable fields.

- Validation

  The extended data stream can be used to define an input field in an IBM 8775 display as a MANDATORY FILL, or MANDATORY ENTER field.

  Mandatory fill
  : Input field must be filled before pressing ENTER.

  Mandatory enter
  : The operator must key data into the input field before pressing ENTER

- Modified-data tag (MDT)

  The modified-data tag is turned on when fields are modified by the operator. When the operator presses the enter key or a PF key, only fields which have been modified by the operator or selected by the selector pen are transmitted to the computer. The program may send fields to the 3270 with the modified-data tag already on to guarantee that the field will be returned with the next transmission.

- Insert-cursor indicator

  The insert-cursor indicator is not a field attribute. Instead, it positions the cursor under the first data character of the field. If multiple fields specify insert cursor, the cursor is positioned with the last field specified. If no insert cursor is specified, the cursor is placed at position zero (row 1, column 1) on the screen.

Screen Sizes

As mentioned above, the 3270 is available in several screen sizes. Some 3270 devices are available with two screen sizes: the DEFAULT (small) size and the ALTERNATE (large) size. The system programmer specifies the screen sizes in the terminal control table (TCT), and specifies, in the program control table (PCT) for each transaction, which of the two possible sizes that transaction will use. See Chapter 3.4 for details.

If ERASE is not specified on a terminal output command, the screen will be unchanged from its previous setting, that is, the previous transaction selection, or the default if the operator has just switched on or has cleared the screen.

In normal practice, this means that an application program should specify ERASE with its first output request. On receipt of a CLEAR key indication, CICS/VS will preserve the selected screen size, so that an ERASE is not needed for output requests following the first.

BASIC MAPPING SUPPORT (BMS)

The facility used by the ELS programmer to interface with 3270 devices is known as Basic Mapping Support (BMS).

Basic mapping support provides commands and options that can be used to specify formatting in a standard way. BMS adapts data streams provided by the application program, to satisfy the requirements of a particular device. Conversely, data received from a device is converted by BMS to a standard form.

Data Mapping And Formatting

Data mapping is the technique used by BMS to convert the standard non-device-dependent data format, which the application program uses, to and from the device-dependent data stream required for the particular terminal in use. Device-dependent control characters are embedded or removed by BMS during this processing.

The standard format ("field data format") in which the application program can provide or accept BMS data is as follows:

Data is provided to BMS as separate fields. Each field is given a symbolic field name, which is used when passing data to, or retrieving data from, BMS. The data stream is presented to BMS as a series of fields, each of which has the following form:

LENGTH
: Two bytes containing the length (in bytes) of the following field.

ATTR
: A single byte defining attributes of the field, such as protected or unprotected.

EXT. ATTRS.
: Four bytes which define special field attributes to a device with special features. For example, they can convey field color information to an IBM 3279.

FIELD DATA
: The data to be displayed, and to which the attributes will be applied.

Defining Maps

As a preparatory step to using BMS mapping in a particular application, two forms of map are assembled offline by means of CICS/VS macro instructions:

1. Physical map - used by BMS to convert data to or from the format required by the application program.

2. Symbolic description map - used by the application program to refer to the data in storage. This map is a set of source statements held in a source library and copied into the application program during compilation.

All maps form part of a map set; a map set may consist of one or more related maps that are generated and stored together in the CICS/VS libraries.

A map set is prepared using the following macros: DFHMSD to define the map set, DFHMDI to define the maps in the map set, and DFHMDF to define and name the fields in each map. The set of macro instructions for each map set is assembled twice, once to produce the physical maps, and once to produce the symbolic description maps.

Note: Application programmers may need to create and catalog their own maps; however, for the purposes of this manual, operations using macros are treated as system programming. Accordingly, details of the BMS macros and associated information are in Chapter 3.5 of this manual.

Map sets are accessed by BMS through program control. Each map set name must be entered in the processing program table (PPT) by the system programmer (see Chapter 3.4).

Using Maps

The symbolic description map provides names for fields and groups of fields that may be sent to and received from the devices supported by BMS. The symbolic description map must be copied into each application program that uses the associated physical map. (The method of copying varies according to programming language. See "Copying Symbolic Description Maps," in Chapter 3.5.) Data can then be passed to and from the application program under the field names in the symbolic description map. (The actual names used in the application program are those defined by the DFHMDF macro instructions with the addition of the suffix "I" for input or "O" for output.) Since the application program is written to manipulate the data under the field names, altering the map format by adding new fields or rearranging old fields does not necessarily alter the program logic.

If the map format is altered, it is necessary in most cases to make the appropriate changes to the macro instructions that describe the map and reassemble both the physical map and the symbolic description map. The application program must then be recompiled so that the new symbolic description map is copied into the program. If changes to the map are restricted to addition of extended attributes, and the extended attributes will not be changed by the application program, only the physical map need be reassembled.

An application program has access to the input and output data fields using the names supplied to the fields when the maps were generated. The application-program logic should be dependent on the named fields and their contents but should be independent of the relative positions of the data fields within the terminal format. If it becomes necessary to reorganize or add to a map format, the existing application program must be recompiled to gain access to the new positions of these data fields. Reprogramming is not necessary to account for new fields or for the changed terminal format of those fields.

CICS/VS provides a list of the standard attribute bytes that may be used by the programmer in modifying the attribute assigned to a field. The programmer must copy these constants into the program using the name DFHBMSCA (DBMSCA for RPG II). The following constants will then be available for the programmer's use.

CONSTANT	RPG II CONSTANT	ATTRIBUTES
DFHBMUNP	DBMUNP	Unprotected
DFHBMUNN	DBMUNN	Unprotected numeric only
DFHBMPRO	DBMPRO	Protected
DFHBMASK	DBMASK	Autoskip
DFHBMBRY	DBMBRY	Bright intensity
DFHBMDAR	DBMDAR	Nondisplay
DFHBMFSE	DBMFSE	MDT on (field set)
DFHBMPRF	DBMPRF	Protected and MDT on
DFHBMASF	DBMASF	Autoskip and MDT on
DFHBMASB	DBMASB	Autoskip and bright intensity
DFHBMPEM	DBMPEM	Printer end of message
DFHBMPNL	DBMPNL	Printer new-line character
DFHDFT	DDFT	Default override for attribute values in maps
DFHBLUE	DBLUE	Blue
DFHRED	DRED	Red
DFHPINK	DPINK	Pink
DFHGREEN	DGREEN	Green
DFHTURQ	DTURQ	Turqouise
DFHYELLO	DYELLO	Yellow
DFHNEUTR	DNEUTR	Neutral
DFHBLINK	DBLINK	Blink
DFHREVRS	DREVRS	Reverse video
DFHUNDLN	DUNDLN	Underline
DFHMFIL	DMFIL	Mandatory fill
DFHMENT	DMENT	Mandatory enter
DFHMFE	DMFE	Mandatory fill and mandatory enter

Additional attribute constants may be created and cataloged to the source statement library by defining single-character areas, with the appropriate value, in the source language.

The value of the attribute constant may be determined by referring to the manual IBM 3270 Information Display System Component Description.

The application program can temporarily modify the attributes or the initial data of any named field in an output map. BMS supplies a collection of named attribute combinations so that the application program remains essentially independent of the data stream format. (See "Attribute Character," earlier in this section).

The ability to add to map definitions without changing application programs permits modular design and implementation of systems, with a progressive expansion of the screen formats. Design and programming of the first stages of applications can begin before later stages have been designed. These early implementations are protected from updates in the terminal formats.

Display and Print Options

In addition to the techniques discussed here, the following options are provided and are grouped together in the syntax displays that accompany the descriptions of the commands. The explanations of the options are given later in this section, under "Terminal Control Options."

```
ALARM
CURSOR
ERASE
ERASEAUP
FREEKB
FRSET
HONEOM
L40
L64
L80
PRINT
```

Symbolic Cursor Positioning

The CURSOR option of the SEND MAP command can be used to position the cursor on completion of an output operation. Alternatively, a method called symbolic cursor positioning can be used. Symbolic cursor positioning allows a field in the data to be marked, symbolically, such that the cursor is placed under the first data byte of the field on the output screen.

Requirements for the use of SCP are as follows:

- MODE=INOUT must be specified in the DFHMSD macro.
- CURSOR, without an argument, must be specified in the command.
- The length field, suffix "L", associated with the field under which the cursor is to be placed must be initialized to -1.

TERMINAL CONTROL COMMANDS

Terminal control commands are provided to:

- Receive data from a terminal into a data area in the program (RECEIVE MAP)
- Display a formatted screen with initial data only (SEND MAP MAPONLY)
- Modify an already formatted screen, by sending only changed data (SEND MAP DATAONLY)
- Display a formatted screen together with variable data (SEND MAP)
- Erase unprotected (that is, keyed-in) fields on a screen, leaving the screen format and initial data intact (ISSUE ERASEAUP)
- Print the contents of a screen (ISSUE PRINT)
- Pass control on receipt of an attention identifier from a terminal (HANDLE AID)
- Load a programmed symbol set (see Appendix E)

Note: The SEND MAP, SEND MAPONLY, and SEND DATAONLY commands are treated together in this section under "SEND MAP Command."

RECEIVE MAP Command

```
RECEIVE MAP(name)
        [MAPSET(name)]
        [INTO(data-area) | SET(pointer-ref)]

Exceptional conditions:  ERROR, MAPFAIL
```

The RECEIVE MAP command is used to map data from a terminal into a data area in the application program.

If the MAPSET option is omitted, the MAPSET name is assumed to be the same as that specified in the MAP option.

The data area into which the data is to be mapped can be specified in the INTO option. Alternatively, BMS will supply a data area and place its address in the pointer specified in the SET option.

If neither the INTO option nor the SET option is specified, it is assumed that the data is to be mapped into the map area, that is, the data area defined by the symbolic description map copied into the program. If the data is to be written into another data area, it must be named in the INTO option. The data area named must be large enough to accommodate the mapped data.

Once the data has been mapped, fields within the mapped data can be referred to by the field names specified in the DFHMDF macro instructions used to define the map with the additional suffix "I". (For example, a field named PERSNO must be referred to in the application program as PERSNOI.)

In the symbolic description map definition, the DFHMSD macro must have the TIOAPFX=YES operand specified either explicitly or implicitly (see Chapter 3.5).

SEND MAP Command

```
SEND MAP(name)
     [MAPSET(name)]
     [FROM(data-area)]
     [LENGTH(data-value)]
     [DATAONLY | MAPONLY]

     [ALARM]
     [CURSOR[(data-value)]]
     [ERASE | ERASEAUP]
     [FREEKB]
     [FRSET]
     [HONEOM | L40 | L64 | L80]
     [PRINT]

Exceptional condition:  ERROR
```

The SEND MAP command is used to map output data to a terminal.

If the MAPSET option is omitted, the MAPSET name is assumed to be the same as that specified in the MAP option.

If the FROM option is omitted, it is assumed that the data to be mapped is in the map area, that is, the data area defined by the symbolic description map copied into the program. If the data is to be obtained from another data area, it must be named in the FROM option; the LENGTH option is not required unless the data to be mapped is less than the total length of the data defined by the map.

In the symbolic description map definition, the DFHMSD macro must have the TIOAPFX=YES operand specified either explicitly or implicitly (see Chapter 3.5).

The MAPONLY option specifies that only the map, with initial data, is to be written. This option could be used, for example, to send a menu to a screen for the operator to enter selections into. (After any entered data has been read, the screen can be cleared of keyed-in data by the ISSUE ERASEAUP command, described below.) The MAPONLY option cannot be used with the FROM option.

The DATAONLY option specifies that only data supplied by the application program is to be written; it can be used to modify some of the data on the screen while leaving the remainder of the screen intact. When the DATAONLY option is used, the screen must be formatted, and the receiving field must have a format compatible with the data being entered.

Unless the MAPONLY option is specified, data placed in either the map area or the data area named in the FROM option is merged with initial data from the map. The user-supplied data and/or attribute character supplied for a given field replaces the corresponding default data and/or attribute character from the map. If the first byte of the user-supplied data for a field is X'00', the data from the map for that field is used. (Unused data fields must be cleared to binary zeros.) If the user-supplied attribute for a field is X'00', the attribute from the map for that field is used.

Attribute bytes can be modified by moving one of the symbolic names described above, under "Attribute Character," to an attribute field.

ISSUE ERASEAUP Command

```
ISSUE ERASEAUP
```

The ISSUE ERASEAUP command erases all unprotected fields on a screen, positions the cursor at the first unprotected field, resets modified data tags in unprotected fields to zero, and unlocks the keyboard.

The command is useful in applications such as continuous data entry, where the operator enters records on a formatted screen. A screen format with protected keyword fields is first sent to the terminal; then, after the entered data has been read, ISSUE ERASEAUP allows further entry against the protected keywords. This saves line time, since the screen format does not have to be retransmitted.

ISSUE PRINT Command

```
ISSUE PRINT
```

The ISSUE PRINT command prints the contents of a screen on the first eligible and available 3270 printer.

A printer is eligible if:

- The printer is attached to the same control unit as the screen.
- The printer's buffer size is at least equal to that of the screen.
- The printer and screen are specified in the same DFHTCT TYPE=GPENTRY macro (see Chapter 3.4).
- The printer is flagged as eligible by specifying

  DFHTCT TYPE=GPENTRY,TRMFEAT=(P)

A printer is available if it is operational and not already printing.

For a BTAM system, if the screen image cannot be printed immediately, the data is automatically queued in temporary storage until it can be printed.

For a VTAM system, the data might be printed on an alternative printer, depending on whether the system programmer has arranged this (see the CICS/VS System Programmer's Reference Manual).

The ISSUE PRINT command should not be confused with the local copy operation available with 3274 and 3276 controllers. (For information on the local copy operation, see the IBM 3270 Component Description.) The ISSUE PRINT command is a CICS/VS function that prints the contents of a screen; the local copy operation is a hardware feature, available with some 3270 control units, that prints the contents of a screen.

HANDLE AID Command

```
HANDLE AID option[(label)] [option[(label) ]]...
```

The HANDLE AID command can be used to pass control to a specified label when a particular attention identifier (AID) is received from a display device; control is passed after the input operation is completed. In the absence of any HANDLE AID request for an AID, control returns to the application program at the point immediately following the input request. A HANDLE AID request will take precedence over a HANDLE CONDITION request; if an AID is received during an input operation, for which a HANDLE AID request is active, control will pass to the label specified in that request, regardless of any exceptional conditions that may have occurred (but that did not stop receipt of the AID).

The HANDLE AID options that can be specified are:

- any of the program attention key names (PA1, PA2, or PA3)
- any of the program function key names (PF1 through PF24)
- CLEAR or ENTER (for the keys of the same names)
- LIGHTPEN (for a selector pen attention or cursor select key attention)
- OPERID (for the Operator Identification Card Reader, or the extended (standard) magnetic stripe reader)
- ANYKEY (which is a general option that is equivalent to any PA key, any PF key, or the CLEAR key)

A HANDLE AID request made to handle a particular AID remains active until the task is terminated or until another HANDLE AID request for that AID is made, when the new request overrides the old one. (If no label is specified in the new request, the request has the effect of deactivating the request.) The effect of a HANDLE AID request extends only to the program in which it is issued. Each new program in a task starts without any active HANDLE AID requests. When control returns to a program from a program at a lower logical level, the HANDLE AID requests that were active in the higher-level program before control was transferred from it are reactivated, and any HANDLE AID requests activated in the lower-level program are deactivated.

If an OPERID attention identifier is received, the EXEC interface block can be inspected, to determine which of the magnetic stripe readers (MSR OR MSRE) has been used. MSRE generates an AID of X'E7', and MSR generates AID X'E6'.

If any AID covered by the general option ANYKEY is received and there is no active HANDLE AID request for the particular AID but there is an active HANDLE AID ANYKEY request, control will pass to the label specified in this request. Any HANDLE AID request for a specific AID overrides the HANDLE AID ANYKEY request as far as that AID is concerned.

The following example shows a HANDLE AID request that sets up one label for the PA1 key AID, and a second label for the PA2 and PA3 key AIDs, all of the PF key AIDs except PF10, and the CLEAR key AID:

```
EXEC CICS HANDLE AID          Handle AID characters
PA1 (LAB1)                    Specify label for PA1
ANYKEY(LAB2)                  Specify label for ANYKEY group
PF10                          Exclude PF10 from ANYKEY group
```

The maximum number of key names that can be coded on a single HANDLE AID command is 12.

Alternatively the program may examine the value of the EIBAID field of the EXEC interface block (EIB) to determine which attention key had been pressed by the terminal operator. The 3270 terminal transmits an attention identification character AID, which is stored in the EIBAID (EAID for RPG II) field of the EXEC interface block (EIB). The program may compare the contents of this field with the following constants copied into the working storage area.

CONSTANT	ATTENTION KEY	CONSTANT	ATTENTION KEY
DFHENTER	ENTER KEY	DFHPF11	PF11 KEY
DFHCLEAR	CLEAR KEY	DFHPF12	PF12 KEY
DFHPA1	PA1 KEY	DFHPF13	PF13 KEY
DFHPA2	PA2 KEY	DFHPF14	PF14 KEY
DFHPA3	PA3 KEY	DFHPF15	PF15 KEY
DFHPF1	PF1 KEY	DFHPF16	PF16 KEY
DFHPF2	PF2 KEY	DFHPF17	PF17 KEY
DFHPF3	PF3 KEY	DFHPF18	PF18 KEY
DFHPF4	PF4 KEY	DFHPF19	PF19 KEY
DFHPF5	PF5 KEY	DFHPF20	PF20 KEY
DFHPF6	PF6 KEY	DFHPF21	PF21 KEY
DFHPF7	PF7 KEY	DFHPF22	PF22 KEY
DFHPF8	PF8 KEY	DFHPF23	PF23 KEY
DFHPF9	PF9 KEY	DFHPF24	PF24 KEY
DFHPF10	PF10 KEY	DFHPEN	SELECTOR PEN or CURSOR SELECT KEY
DFHOPID	OPERID or MSR		
DFHMSRE	Extended (standard) MSR		

Note: For RPG II, the constant names are as above, except that the "FH" is removed; for example, "DFHENTER" would be "DENTER" for RPG II.

The clear, PA1, PA2, and PA3 keys do not transmit any data to CICS/VS. They normally signal a special operator request that does not require data, such as print, page forward, page backward, or exit from a repeating transaction.

Terminal Control Options

ALARM

specifies that the 3270 audible alarm feature is to be activated.

CURSOR[(data-value)]

specifies the position to which the cursor is to be returned upon completion of a write operation.

The data value must be a halfword binary value that specifies the cursor position relative to zero; the range of values that can be specified depends on the size of the screen being used. If no data value is specified, symbolic cursor positioning is assumed; see "Symbolic Cursor Positioning" earlier in this section.

ERASE

specifies that the screen is to be erased and the cursor returned to the upper left corner of the screen before this page of output is displayed. (ERASE will also select the screen size to be used — either default or alternate — depending on the parameters in the TCT and PCT entries for the terminal and transaction.)

ERASEAUP

specifies that all unprotected fields are to be erased before the screen is displayed.

FREEKB
specifies that the keyboard should be unlocked after the data is written. If FREEKB is omitted, the keyboard remains locked.

FROM(data-area)
specifies the data area containing the data to be mapped by a SEND MAP or RECEIVE MAP command.

If FROM is specified, the MAPONLY option must not be specified. If this option is omitted from a SEND MAP command, and the map name is a literal constant, the name of the data area is assumed to be the map name with the addition of the suffix "O".

FRSET
specifies that the modified data tags (MDTs) of all fields currently in the 3270 buffer are to be reset to the not-modified condition (that is, field reset) before any map data is written to the buffer.

If an attribute for a field is generated by a BMS request, either from the map (by the ATTRB operand of the DFHMDF macro) or from the data, the request will control the MDT setting for the field.

HONEOM
See L40, L60, L80, HONEOM

INTO(data-area)
specifies the data area into which the mapped data is to be written. If neither INTO nor SET is specified and the map name is a literal constant, the name of the data area is assumed to be the map name with the addition of the suffix "I". If the data area has not been generated by the BMS map definition process, it must start with a 12-byte TIOA prefix.

L40, L64, L80, HONEOM
specify the line length for a 3270 printer; a carrier return and line feed are forced after the specified number of characters have been printed on the line. If HONEOM is specified, the line length is the default line length of the device (usually 132).

MAP(name)
specifies the name, up to seven characters long, of the map to be used.

MAPSET(name)
specifies the name, up to seven characters long, of the map set to be used. The map set must reside in the CICS/VS program library, and an entry for it must exist in the processing program table (PPT). If the MAPSET option is not specified, the name given in the MAP option is assumed to be that of the map set.

PRINT
specifies that the data is to be sent to a printer.

SET(pointer-ref)
specifies the pointer that is to be set to the address of the input or output data.

Terminal Control Exceptional Conditions

MAPFAIL
occurs, on input only, if the data to be mapped has a length of zero or does not contain a start-buffer-address (SBA) sequence.

Default action: terminate the task abnormally.

ERROR
See under "Exceptional Condition Handling" for an explanation of the ERROR condition.

Access to System Information

It is possible to write many application programs using the CICS/VS command-level interface without any knowledge of or reference to CICS/VS control blocks and storage areas. However, it is sometimes necessary to obtain information that is valid outside the local environment of the application program; the ADDRESS command is provided to make access to such information possible. The control blocks are introduced below.

Common Work Area (CWA)

The common work area may be used by any task during CICS/VS operation for storage or retrieval of information such as statistics. If requested, it is supplied by CICS/VS as a pool of storage available to all users. As there is only one CWA, standards must be established to coordinate its use by all programs.

Transaction Work Area (TWA)

If the transaction work area is requested, it is supplied by CICS/VS for storage throughout a transaction. The common work area is shared by all terminals using CICS/VS, but each terminal has a separate TWA. Therefore, the program may store variable information, such as counters and switch settings in the TWA, without concern for other terminals sharing the program. Constants may also be stored in the TWA, but initial values must be provided by the program.

Fields within the TWA are described by the program. The TWA size must be specified in the program control table before the program is run. Although the TWA size may range up to approximately 32K, typical sizes should not exceed a few hundred bytes. The TWA is acquired by CICS/VS when the transaction is initiated, and freed on transaction termination.

Terminal Control Table User's Area (TCTUA)

Occasionally, a program may need to save variable data between transactions. For example, an application with multiple programs may need to store intermediate results. The terminal control table user's area serves this purpose. It is supplied by CICS/VS as storage to be used to retain data between transactions from a single terminal.

Fields within the TCTUA are defined by the program. The largest size TCTUA (maximum 255 bytes) used by any program must be specified in the terminal control table before CICS/VS execution. As the data stored in the TCTUA exists from program to program, or as long as CICS/VS is running, TCTUA usage must be coordinated between programs, since there is only one TCTUA per terminal.

EXEC Interface Block (EIB)

In addition to the usual CICS/VS control blocks, each task in a command-level environment has a control block called the EXEC interface block (EIB) associated with it. An application program can access all of the fields in the EIB of a task by name. The EIB contains information that is useful during the execution of an application program, such as the transaction identifier, the time and date (initially when the transaction was initiated, and subsequently, if updated by the application program), and the cursor position on a display device. The EIB also contains information that will be helpful when a dump is being used to debug a program. Refer to Appendix B for details of the EIB.

ADDRESS Command

```
ADDRESS option(pointer-ref)
        [option(pointer-ref)]...
```

The ADDRESS command is used to obtain access (addressability) to specified CICS/VS storage areas. The pointer reference specified in the option is set to the address of the storage area named in the option. If the storage area does not exist, the pointer reference is set to X'FF000000'.

The ADDRESS command options are:

CWA — common work area

TCTUA — terminal control table user area

TWA — transaction work area

The CWA, TCTUA, and TWA, described above, are used to pass information between application programs. The CWA can be used at any time, but the TCTUA can be used only if the same terminal is associated with the application programs involved (which can be in different tasks), and the TWA can be used only if the application programs are in the same task. The following example shows how the ADDRESS command can be used to obtain access to the TWA:

```
EXEC CICS ADDRESS  TWA(WAPTR)
```

(Information can also be passed between programs using the COMMAREA option of program control commands, described under "Passing Data between Programs," later in this section.)

Access to Data Bases

CICS/VS transactions can access two kinds of data bases:

- Standard VSE files holding a data base.
- Data Language/I (DL/I) data bases.

Standard VSE files are processed by CICS/VS file control, which permits the retrieval, addition, and updating of records in ISAM and VSAM files, and the deletion of records in VSAM files. File control relieves the programmer of buffer management, blocking and deblocking, and access-method dependencies.

A DL/I data base gives the application programmer a much greater degree of data independence than is given by file control. The programmer is presented with a logical view of the data base in terms of a hierarchy of segments. ELS application programs can access these segments for online enquiry without the programmer having to know how they are organized. (Online updating of DL/I data bases is not available in ELS; a batch program must be used.)

The Entry Level System supports the DL/I high level language interface as well as the DL/I call interface.

The actual processing of a DL/I data base is performed by the program product Data Language/I DOS/VS (program number 5746-XX1), with which CICS/VS interfaces.

For information on the use of DL/I, please refer to the CICS/VS Application Programmer's Reference Manual (Command Level, RPG II, or Macro Level).

File Control

The CICS/VS file control program processes fixed-length or variable-length, blocked or unblocked, or undefined records of an indexed file. (Sequential files are processed by the transient data control program.)

CICS/VS uses the standard access methods Indexed Sequential Access Method (ISAM) and Virtual Storage Access Method (VSAM). If an ISAM file is converted to a VSAM file organization, using VSAM file conversion utilities, no alteration to application programs that access the file is necessary, but the system programmer must change the file control table (see Chapter 3.4).

File control commands can be used to:

- Read a record from a file (READ).
- Write a new record to a file (WRITE).
- Update an existing record in a file (REWRITE).
- Delete a single record or a group of records from a VSAM key-sequenced data set (DELETE).
- Browse through a VSAM file (STARTBR, READNEXT, READPREV, RESETBR, and ENDBR).
- Release the task's exclusive control over a file (UNLOCK).

File Identification

Files are identified in file control commands by the DATASET option, and must have been defined previously in the file control table. These definitions may be set up with the help of the system programmer. Logical record handling only is required in the application program; buffers and work areas are acquired automatically by CICS/VS.

Direct Access to Records

When records are read directly (that is, searched for by a search argument such as a key) using the READ command, the record is retrieved and placed in main storage according to which of the options INTO, SET, and LENGTH have been specified.

The INTO option is used to specify the area into which the record is to be placed. For variable-length records, the LENGTH option must specify the maximum length of record that the application program will accept. If the record exceeds this value, it is truncated and the LENGERR condition occurs. If the LENGTH option is specified for fixed-length records, the length specified must be equal to the record length, or the LENGERR condition occurs. After the record has been retrieved, the data area specified in the LENGTH option is set to the actual record length (before any truncation occurred).

Alternatively, a pointer reference can be specified in the SET option. CICS/VS then acquires an area large enough to hold the record and sets the pointer reference to the address of that area. After the

record has been retrieved, if a data area is specified in the LENGTH option, it is set to the record length.

The READ command can be used for both read-only and read-for-update operations. If the record is to be updated, the UPDATE option must be specified. When a record has been read for update, CICS/VS maintains exclusive control (which varies according to the access method in use) to prevent another task accessing the record until it has been rewritten, or until exclusive control is released by an UNLOCK command, or (for VSAM only) until the record is deleted.

When adding records, using the WRITE command, or when updating records, using the REWRITE command, the record to be written is specified in the FROM option, and its length in the LENGTH option. (LENGTH can be omitted for fixed-length records.)

When a record has been read for update, the REWRITE command should be issued as soon as possible, to avoid obstructing file storage and possibly preventing other transactions from accessing the record.

VSAM DATA SETS

Record Identification

Records in VSAM files are identified in one of three ways: by key, by relative byte address or by relative record number. One of these must be specified (in the RIDFLD option) as the search argument. If a relative byte address is supplied, the RBA option must be specified; if a relative record number is supplied, the RRN option must be specified.

VSAM Keys

When writing records, a complete key must be provided. When reading records (except for the purpose of updating a protected, key-sequenced file, when a complete, equal-key search must be specified), the search key provided can be a complete key or a generic key, and, for either type, the search can be for an equal key or a greater-or-equal key. Greater-or-equal searches are made when the GTEQ option is specified.

If a complete key is specified, CICS/VS searches for the record having that key; if it cannot be found and the GTEQ option is specified, the first record having a key greater than the specified key is retrieved, but if the GTEQ option is not specified, the NOTFND exceptional condition occurs.

If a generic key is specified, its length must be specified in the KEYLENGTH option, and the GENERIC option must be specified. The search for the required record uses only the number of characters contained in the generic key. CICS/VS searches for the first record having a matching generic key; if no matching record is found, and the GTEQ option is specified, the first record having a greater generic key than the specified generic key is retrieved, but if the GTEQ option is not specified, the NOTFND exceptional condition occurs.

VSAM Exclusive Control

When a VSAM record is read for update, VSAM maintains exclusive control of the control interval containing that record. An attempt by the same task to read a second record from the same control interval for update, or add a new record to the same control interval before exclusive control is released (by an UNLOCK request), will cause a lockout. To prevent lockout, the ERROR exceptional condition occurs if, following a READ UPDATE command, a second READ UPDATE command or a WRITE command is executed before exclusive control is released by a REWRITE, UNLOCK, or DELETE command.

Deletion of VSAM Records

Records in a VSAM key-sequenced file can be deleted, either singly or in groups, using the DELETE command. Single records can be searched for using a key, relative byte address, or relative record number. Groups of records can be deleted only if the records all have a common starting group of characters in their keys (that is, a common generic key).

A record that has been read for update (that is, with the UPDATE specified in the READ command), may be deleted also by a DELETE command, but only if a complete key has been specified. If deletion is attempted for a record with a generic key, or if the DELETE command includes the RIDFLD option, the ERROR condition will occur.

Note that a DELETE request that follows a READ UPDATE request must not have the RIDFLD option specified, or the ERROR condition occurs.

VSAM Mass Insert Operations

The MASSINSERT option specifies that a VSAM mass insert operation is in progress; it must be specified in every WRITE command that is part of the operation.

A mass insert operation must be ended to ensure that all records are written to the file; a READ command will not necessarily find a record that has been added by an incomplete mass insert operation. A mass insert operation is ended by an UNLOCK command; any incomplete operations will be ended when the task ends.

VSAM Browse Operations

When reading VSAM records sequentially, the STARTBR command specifies the starting point only for the browse; no records are retrieved. The READNEXT command reads records sequentially from the data set, starting with the specified record. The READPREV command does the same as the READNEXT command, except that records are read in reverse order.

Records are retrieved and placed in main storage using the INTO, SET, and LENGTH options in the same way as for direct access.

The starting point can be reset at any time by a RESETBR command. The ENDBR command ends a browse.

When more than one browse is required on a data set at the same time, the REQID option must be included in every browse command to distinguish between the browse operations.

A browse can be started at any record in a VSAM data set. A record identification field of zero, or the options KEYLENGTH(0) and GENERIC, will start the browse at the first record. Any other starting point must be specified in the same way as a single record is retrieved, using a key (complete or generic), relative byte address, or relative record number.

The RESETBR command can be used not only to reset the starting position for the browse, but also to change the type of search argument (key, relative byte address, or relative record number).

The record identification field is updated by CICS/VS with the complete key, relative byte address, or relative record number of the record retrieved each time a READNEXT or READPREV command is executed. For a given browse all associated commands must use the same record identification field.

VSAM data sets can be browsed backwards as well as forwards. Specifying a record identification field of hexadecimal 'FF's will start the browse at the end of the data set.

VSAM Skip-Sequential Processing

Skip-sequential processing can be performed on a VSAM data set. The identifier (key, relative byte address, or relative record number) of the next record required must be placed in the record identification field specified in the RIDFLD option of the READNEXT command. This record need not be the next sequential record in the data set, but must have a key, relative byte address, or relative record number greater than the last record accessed. (A READPREV command should not be used.) This procedure allows quick random access to a VSAM file by reducing index search time.

The identifier must be of the same form (key, relative byte address, or relative record number) as that specified in the STARTBR (or the last RESETBR) command for this browse. If the STARTBR or last RESETBR command specified a generic key, the new identifier must also be a generic key, but it need not be of the same length.

If the STARTBR or last RESETBR command specifies an equal-key search (complete or generic), a READNEXT command using skip-sequential processing may result in a NOTFND condition.

Sharing VSAM Resources

CICS/VS permits the sharing of VSAM resources. Resources to be shared can be specified by the DFHFCT TYPE=SHRCTL macro instruction, as explained in the Chapter 3.4. When a task requires resources in several VSAM files at the same time and these files are sharing resources, the possibility of a lockout increases.

VSAM Alternate Indexes

The VSAM Alternate Index feature allows access to a file using several indexes, which contain alternate keys to the records in the file. A record can be accessed by many different keys; also, many records can have the same alternate key in an alternate index.

Accessing a record via an alternate index is similar to accessing a normal key-sequenced file, unless records having non-unique alternate keys are involved. If the (alternate) key provided in a READ command is not unique, the first record in the file having that key is read, and the DUPKEY exceptional condition occurs.

FILE CONTROL COMMANDS

READ Command

```
READ {INTO(data-area) | SET(pointer-ref)}
     [LENGTH(data-area)]
     DATASET(name)
     RIDFLD(data-area)
     [KEYLENGTH(data-value)]      (mandatory with GENERIC)
     [GENERIC]                    (VSAM only)
     [GTEQ | EQUAL]               (VSAM only)
     [RBA | RRN]                  (VSAM only)
     [UPDATE]

Exceptional conditions:  LENGERR, NOTFND, DUPKEY, ERROR
```

The READ command is used to retrieve a single record from a file.

The following example shows how to read a record from a file into a specified data area:

```
EXEC CICS READ                    Read a record
     INTO(RECORD)                 Data area
     DATASET('MASTER')            File
     RIDFLD(ACCTNO)               Record identification field
```

The following example shows how to read a record from a VSAM file, using a generic key, specifying a greater-or-equal key search, and specifying that the record is later to be rewritten into a specified data area:

```
EXEC CICS READ                    Read a record
     SET(RECBAR)                  Request pointer reference set
     LENGTH(RECLEN)               Record length
     DATASET('MASTVSAM')          File
     RIDFLD(ACCTNO)               Record identification field
     KEYLENGTH(4)                 Generic key length
     GENERIC                      Key is generic
     GTEQ                         Greater-or-equal search
     UPDATE                       Record is to be rewritten
```

UNLOCK Command

```
UNLOCK DATASET(name)

Exceptional conditions:  ERROR
```

The UNLOCK command releases exclusive control that has been applied as a result of a READ UPDATE command. It is used when a record has been retrieved for update but can then be seen not to need updating. The effect is to allow other programs to access the data that was to be updated.

The UNLOCK command also terminates a VSAM mass insert operation.

WRITE Command

```
WRITE FROM(data-area)
      [LENGTH(data-value)]
      DATASET(name)
      RIDFLD(data-area)
      [KEYLENGTH(data-value)]
      [RBA | RRN]                (VSAM only)
      [MASSINSERT]               (VSAM only)

Exceptional conditions:  LENGERR, DUPREC, ERROR
```

The WRITE command is used to write a record to a file. For example:

```
EXEC CICS WRITE                  Write a record
     FROM(RECORD)                Data area
     LENGTH(DATLEN)              Record length
     DATASET('MASTER')           File
     RIDFLD(KEYFLD)              Record identification field
```

REWRITE Command

```
REWRITE FROM(data-area)
        [LENGTH(data-value)]
        DATASET(name)

Exceptional conditions:  LENGERR, ERROR
```

The REWRITE command is used to update a record in a file. The record to be updated must first be read by a READ command with the UPDATE option. For example:

```
EXEC CICS REWRITE                Update a record
     FROM(RECORD)                Data area
     DATASET('MASTER')           File
```

DELETE Command (VSAM only)

```
DELETE DATASET(name)
       [RIDFLD(data-area)]              (mandatory with GENERIC)
       [KEYLENGTH(data-value)]          (mandatory with GENERIC)
       [GENERIC [NUMREC(data-area)]]
       [RBA | RRN]

Exceptional conditions:  NOTFND, ERROR
```

The DELETE command is used to delete a single record or, if a generic key is provided, a group of records, from a VSAM key-sequenced file; when used for a group of records, the RIDFLD option is required.

Unless a generic key is used, this command can be used also to delete a VSAM record that has been read for update, instead of using a REWRITE or UNLOCK command. When used in this way, RIDFLD must not be specified.

The following example shows how to delete a group of records in a VSAM data set:

```
EXEC CICS DELETE                  Delete record
     DATASET('MASTVSAM')          File
     RIDFLD(ACCTNO)               Record identification field
     KEYLENGTH(4)                 Generic key length
     GENERIC                      Key is generic
     NUMREC(NUMDEL)               Return number deleted
```

STARTBR Command (VSAM only)

```
STARTBR DATASET(name)
        RIDFLD(data-area)
        [KEYLENGTH(data-value)]     (mandatory with GENERIC)
        [GTEQ | EQUAL]
        [RBA | RRN]
        [REQID(data-value)]

Exceptional conditions:  ERROR
```

This command is used to specify the record in a file at which the browse is to start. No records will be retrieved until a READNEXT or READPREV command is executed.

READNEXT Command (VSAM only)

```
READNEXT {INTO(data-area) | SET(pointer-ref)}
         [LENGTH(data-area)]
         DATASET(name)
         RIDFLD(data-area)
         [KEYLENGTH(data-value)]
         [RBA | RRN]
         [REQID (data-value)]

Exceptional conditions:  DUPKEY, ENDFILE, ERROR,
                         LENGERR, NOTFND
```

This command is used to retrieve records in sequential order from a file. It can also be used during VSAM skip-sequential processing.

The RIDFLD option must specify the same data area as was specified in the RIDFLD option in the corresponding STARTBR command or in the last RESETBR command, but the contents of the data area can be different.

READPREV Command (VSAM only)

```
READPREV {INTO(data-area) | SET(pointer-ref)}
         [LENGTH(data-area)]
         DATASET(name)
         RIDFLD(data-area)
         [KEYLENGTH(data-value)]
         [RBA | RRN]
         [REQID(data-value)]

Exceptional conditions:  DUPKEY, ENDFILE, ERROR,
                         LENGERR, NOTFND
```

This command is used to retrieve records in reverse sequential order from a VSAM file.

The RIDFLD option must specify the same data area as was specified in the RIDFLD option in the corresponding STARTBR command or in the last RESETBR command, but the contents of the data area can be different.

RESETBR Command (VSAM only)

```
RESETBR DATASET(name)
        RIDFLD(data-area)
        [KEYLENGTH(data-value)]          (mandatory with GENERIC)
        [GENERIC]
        [GTEQ|EQUAL]
        [RBA | RRN]
        [REQID(data-value)]

Exceptional conditions:  ERROR
```

This command is used to specify the record in a file at which the browse is to be restarted.

The RESETBR command can be issued at any time. It is equivalent to an ENDBR STARTBR sequence.

END BROWSE Command (VSAM only)

```
ENDBR DATASET(name)
      [REQID(data-value)]

Exceptional condition:  ERROR
```

This command is used to end a browse.

File Control Options

DATASET(name)
specifies the symbolic name of the file to be accessed. The name must be alphameric, up to seven characters long, and must have been defined in the file control table (FCT). For details of the FCT, see Chapter 3.4.

FROM(data-area)
specifies the record that is to be written to the file.

GENERIC (VSAM only)
specifies that the search key is a generic (partial) key, the binary length of which is specified in the KEYLENGTH option. The search for a record is satisfied when a record is found that has the same starting characters as that specified.

GTEQ|EQUAL (VSAM only)
EQUAL specifies that the search will be satisfied only by a record having the same key (complete or partial) as that specified in the RIDFLD option.

GTEQ specifies that if the search for a record having the same key (complete or partial) as that specified in the RIDFLD option is unsuccessful, the first record having a greater key will satisfy the search.

INTO(data-area)
specifies the data area into which the record retrieved from the file is to be written.

KEYLENGTH(data-value)
specifies the length of the key (halfword binary) that has been specified in the RIDFLD option. When a generic key is specified, KEYLENGTH is mandatory, and must specify that key. This option can also be specified whenever a key is specified; if the length specified is different from the length specified in the FCT, the ERROR condition occurs.

LENGTH (parameter)
specifies a halfword binary value indicating the length of data to be read or written. This option is not needed for fixed-length records.

For WRITE or REWRITE, the parameter may be a data value.

For READ INTO, READNEXT INTO, or READPREV INTO, the parameter must be a data area specifying the maximum length of data the program will handle. If the value specified is less than zero, zero is assumed. If the actual data length exceeds the value specified, the data is truncated and the LENGERR condition occurs. On completion of the retrieval operation, the data area is set to the original length of the data, except for undefined record format, when the area is set to the maximum record length.

For READ SET, READNEXT SET, or READPREV SET, the parameter must be a data area. On completion of the retrieval operation, the data area is set to the length of the data.

MASSINSERT (VSAM only)
specifies that the WRITE command is part of a mass-insert operation.

RBA (VSAM only)
specifies that the record identification field in the RIDFLD option contains a relative byte address.

REQID (data-value) (VSAM only)
specifies, as a halfword binary value, a unique request identifier for a particular browse operation. This option is used when multiple browse operations are performed on one data set. If this option is not specified, a default value of zero is assumed.

RIDFLD (data-area)
specifies the record identification field. The contents can be a key (for ISAM and VSAM files), or a relative byte address or relative record number (for VSAM files).

RRN (VSAM only)
specifies that the record identification field in the RIDFLD option contains a relative record number.

SET (pointer-ref)
specifies the pointer-reference which is to be set to the address of the retrieved record.

UPDATE
specifies that the record is to be obtained for updating or (VSAM only) deletion. If this option is omitted, a read-only operation is assumed.

File Control Exceptional Conditions

DUPKEY (VSAM only)
occurs if a record is retrieved via an alternate index in which the key that is used is not unique. It will not occur as a result of a READNEXT request that reads the last of the records having the non-unique key.

Default action: terminate the task abnormally.

DUPREC
occurs if an attempt is made to add a record to a file in which the same key already exists.

Default action: terminate the task abnormally.

ENDFILE
occurs if an end-of-file condition is detected during a browse operation.

Default action: terminate the task abnormally.

LENGERR
occurs if any of the following situations exist:

- The LENGTH option is not specified for an input (without the SET option specified) or output operation involving variable-length records.
- The length specified for an output operation exceeds the maximum record size; the record is truncated.
- The length of a record read during an input operation (with the INTO option specified) exceeds the value specified in the LENGTH option; the record is truncated, and the data area supplied in the LENGTH option is set to the actual length of the record.
- An incorrect length is specified for an input or output operation involving fixed-length records.

Default action: terminate the task abnormally.

NOTFND
occurs if an attempt to retrieve or delete a record based on the search argument provided is unsuccessful.

Default action: terminate the task abnormally.

ERROR
See under "Exceptional Condition Handling" for an explanation of the ERROR condition.

Transient Data Control

Transient data control provides a generalized queuing facility and an efficient means of sequential input/output. Data can be queued for subsequent internal or external processing. Selected data, as specified by the application programmer, can be routed to or from predefined symbolic destinations, either intrapartition or extrapartition. Destinations are intrapartition if associated with a facility allocated to the CICS/VS partition and extrapartition if the data is directed to some destination outside the CICS/VS partition. The definitions for the destinations must be contained in the destination control table (DCT) established by the system programmer when the CICS/VS system is generated (see Chapter 3.4 of this manual).

Transient data control commands are provided to:

- Write data to a transient data queue (WRITEQ TD).
- Read data from a transient data queue (READQ TD).
- Delete an intrapartition transient data queue (DELETEQ TD).

Intrapartition Destinations

Intrapartition destinations are queues of data on direct-access storage devices for use with one or more programs running as separate tasks. Data directed to or from these internal destinations is called intrapartition data; it consists of variable-length records. Intrapartition destinations can be associated with either a terminal or an output file. Intrapartition data may ultimately be transmitted on request to the destination terminal or retrieved sequentially from the output file.

The storage associated with an intrapartition queue can be reused. The system programmer can specify, for each symbolic destination, whether or not storage is to be reused as the data on it is read. If the storage is specified to be nonreusable, an intrapartition queue continues to grow, irrespective of whether the data has been read, until a DELETEQ TD command is issued; the whole queue is then deleted.

Extrapartition Destinations

Extrapartition destinations are queues, residing on any sequential device, that are accessible by programs outside (or within) the CICS/VS partition. In general, sequential extrapartition destinations are used for storing and retrieving data outside the CICS/VS partition. For example, one task might read data from a remote terminal, edit the data, and write the results to a file for later processing in another partition. Logged data, statistics, and transaction error messages are examples of data that can be written to extrapartition destinations. In general, extrapartition data created by CICS/VS is intended for subsequent batched input to non-CICS/VS programs. Data can also be routed to an output device such as a line printer.

Data directed to or from an external destination is called extrapartition data and consists of sequential records that are fixed-length or variable-length, blocked or unblocked. The record format for a particular extrapartition destination must be described by the system

programmer when setting up the destination control table. (Refer to Chapter 3.4 for details.)

Indirect Destinations

Intrapartition and extrapartition destinations can be used as indirect destinations, which are symbolic references to other destinations. This can help in program maintenance, in that data can be routed to a destination known by a different symbolic name, without changing programs that use the original name; only the destination control table (DCT) need be changed. Since indirect destinations are established by destination control table entries, the application programmer need not usually be concerned with how this is done. Further information is available in Chapter 3.4.

Automatic Transaction Initiation (ATI)

For intrapartition destinations, CICS/VS provides the option of automatic transaction initiation. The system programmer establishes a basis for automatic transaction initiation by specifying a nonzero trigger level and a transaction identifier for a particular intrapartition destination in the destination control table. (See the discussion of the DFHDCT TYPE=INTRA macro in Chapter 3.4.) When the number of entries (created by WRITEQ TD commands issued by one or more programs) in the queue reaches the specified trigger level, the specified transaction is automatically initiated. Control is passed to a program that processes the data in the queue; the program must issue repetitive READQ TD commands to deplete the queue.

Once the queue has been depleted, a new automatic transaction initiation cycle begins. That is, a new transaction is scheduled for initiation when the specified trigger level is again reached, whether or not the prior transaction is complete.

To ensure that termination of an automatically initiated transaction occurs when the queue is empty, the application program should test for a QZERO condition rather than for some application-dependent factor such as an anticipated number of records; only the QZERO condition indicates a depleted queue.

TRANSIENT DATA CONTROL COMMANDS

WRITEQ TD Command

```
WRITEQ TD QUEUE(name)
          FROM(data-area)
          [LENGTH(data-value)]

Exceptional condition:  ERROR
```

The WRITE TD command is used to write transient data to a predefined symbolic destination. The destination is identified in the QUEUE option.

The FROM option specifies the data to be written to the queue, and the LENGTH option specifies the record length. The LENGTH option need not be specified for extrapartition queues of fixed-length records if the data area is the same length as the records.

The following example shows how to write data to a predefined symbolic destination; in this case, the control system message log (CSML):

```
EXEC CICS WRITEQ TD              Write to transient data queue
     QUEUE('CSML')               Queue name (destination)
     FROM(MESSAGE)               Data to be written
     LENGTH(LENG)                Data length
```

READQ TD Command

```
READQ TD QUEUE(name)
         {INTO(data-area) | SET(pointer-ref)}
         [LENGTH(data-area)]

Exceptional conditions:  ERROR, LENGERR, QZERO
```

The READQ TD command is used to read transient data from a predefined symbolic source. The source is identified in the QUEUE option.

The INTO option is used to specify the area into which the data is to be placed. The LENGTH option must specify a data area that contains the maximum length of record that the program will accept. If the record exceeds this value, it is truncated and the LENGERR condition occurs. After the retrieval operation, the data area specified in the LENGTH option is set to the record length (before any truncation occurred). (The LENGTH option need not be specified for extrapartition queues of fixed-length records if the length is known and a data area of the correct size is available.)

Alternatively, a pointer reference can be specified in the SET option. CICS/VS then acquires an area large enough to hold the record and sets the pointer reference to the address of that area. After the retrieval operation, the data area specified in the LENGTH option is set to the record length.

The following example shows how to read a record from an intrapartition queue, which in this case is the control system message log (CSML), into a data area specified in the request:

```
EXEC CICS READQ TD               Read from transient data queue
     QUEUE('CSML')               Queue name (source)
     INTO(DATA)                  Data area
     LENGTH(LENG)                Length program will accept
```

The next example shows how to read a record from an extrapartition queue having fixed-length records into a data area provided by CICS/VS; the pointer reference specified by the SET option is set to the address

of the storage area reserved for the data record. It is assumed that the record length is known.

```
EXEC CICS READQ TD                  Read from transient data queue
     QUEUE(EX1)                     Queue name (source)
     SET(PREF)                      Request pointer reference set
```

DELETEQ TD Command

```
DELETEQ TD QUEUE(name)

Exceptional condition:  ERROR
```

The DELETEQ TD command is used to delete all of the transient data associated with a particular intrapartition destination (queue). All storage associated with the destination is freed.

This command must be used to free the storage associated with a destination specified as nonreusable in the destination control table. Otherwise, the storage remains allocated to the destination and continues to grow whenever data is written to the destination.

Transient Data Control Options

FROM(data-area)
specifies the data that is to be written to the transient data queue.

INTO(data-area)
specifies the user data area into which the data read from the transient data queue is to be written. If this option is specified, move-mode access is implied.

LENGTH(parameter)
specifies a halfword binary value to be used with WRITEQ TD and READQ TD commands.

For a WRITEQ TD command, the parameter must be a data value that is the length of the data that is to be written.

For a READQ TD command with the INTO option, the parameter must be a data area that specifies the maximum length of data that the program can accept. If the value specified is less than zero, zero is assumed. If the length of the data exceeds the value specified, the data is truncated to that value and the LENGERR condition occurs. On completion of the retrieval operation, the data area is set to the original length of the data.

For a READQ TD command with the SET option, the parameter must be a data area. On completion of the retrieval operation, the data area is set to the length of the data.

QUEUE(name)
specifies the symbolic name of the queue (destination) to be written to, read from, or deleted. The name must be alphameric, up to four characters in length, enclosed in single quotes (' '), and must have been defined in the destination control table (DCT) by the system programmer.

SET(pointer-ref)
specifies a pointer reference that is to be set to the address of the data read from the queue. If this option is specified, locate-mode access is implied.

Transient Data Control Exceptional Conditions

LENGERR
occurs in any of the following situations:

- The LENGTH option is not coded for an input (without the SET option) or output operation involving variable-length records.
- The length specified on output is greater than the maximum record size specified for the queue in the DCT.
- The record read from a queue is longer than the length specified for the input area; the record is truncated and the data area supplied in the LENGTH option is set to the actual record size.
- An incorrect length is specified for a fixed-length-record input or output operation.

Default action: terminate the task abnormally.

QZERO
occurs when the destination (queue) accessed by a READQ TD command is empty.

Default action: terminate the task abnormally.

ERROR
See under "Exceptional Condition Handling" for an explanation of the ERROR condition.

Temporary Storage Control

Temporary storage control allows an application program to store data in virtual storage for later retrieval by any program. This is sometimes known as a scratchpad facility. The same data can be retrieved as often as required.

Temporary storage control commands are provided to:

- Write data to a temporary storage queue (WRITEQ TS).
- Read data from a temporary storage queue (READQ TS).
- Delete a temporary storage queue (DELETEQ TS).

Temporary Storage Queues

Temporary storage queues are identified by symbolic names of up to eight characters assigned by the originating task. Temporary data can be retrieved by the originating task or by any other task using the symbolic name assigned to it. Specific items within a queue are referred to by relative position numbers. To avoid conflicts caused by duplicate names, a naming convention should be established and followed by all programmers; for example, the operator identifier, terminal identifier, or transaction identifier could be appended as a prefix or suffix to each programmer-supplied symbolic name.

Temporary storage queues remain intact, and can be accessed as often as required, until they are deleted. Thus, even after the originating task is terminated, temporary data can be accessed by other tasks through references to the symbolic name under which it was stored. However, temporary storage is released at CICS/VS shutdown, so data cannot be passed from day to day unless the system remains in operation.

In general, temporary storage queues should be used only when direct access to the records is necessary, or for passing data between transactions; transient data control provides facilities for efficient handling of sequential files.

WRITEQ TS Command

```
WRITEQ TS QUEUE(name)
          FROM(data-area)
          LENGTH(data-value)
          [ITEM(data-area) [REWRITE]]

Exceptional conditions:  ERROR, ITEMERR
```

The WRITEQ TS command is used to store temporary data in a temporary storage queue.

The queue is identified in the QUEUE option. The FROM and LENGTH options are used to specify the data that is to be written to the queue, and its length.

The REWRITE option determines whether new records are to be written to a queue, or if existing records are to be updated. If the REWRITE option is not specified, the data is written to the queue specified in the QUEUE option. CICS/VS assigns an item number for this record in the queue and, if the ITEM option is specified, sets the data area supplied in that option to the item number. If the record starts a new queue, the item number is 1; subsequent item numbers follow on sequentially.

If the REWRITE option is specified, the ITEM option must also be specified to identify the item to be replaced by the data identified by the FROM option. If the specified queue cannot be found, the REWRITE option is ignored, a new queue is started, and the ITEMERR exceptional condition occurs. Failure to find the correct item in a queue also causes the ITEMERR condition to occur, but the data is not stored.

The following example shows how to write a new record to a temporary storage queue:

```
EXEC CICS WRITEQ TS              Write to temporary storage queue
     QUEUE(UNIQNAME)             Queue name
     FROM(MESSAGE)               Data to be written
     LENGTH(LENGTH)              Data length
     ITEM(DREF)                  Accept item number
```

The next example shows how to update a record in a temporary storage queue:

```
EXEC CICS WRITEQ TS              Write to temporary storage queue
     QUEUE('TEMPQ1')             Queue name
     FROM(DATAFLD)               Data to be written
     LENGTH(40)                  Data length
     ITEM(ITEMFLD)               Provide item number
     REWRITE                     Data is to update record
     MAIN                        Queue is in main storage
```

READQ TS Command

```
READQ TS QUEUE(name)
         {INTO(data-area) | SET(pointer-ref)}
         LENGTH(data-area)
         [ITEM(data-value)]

Exceptional conditions:  ERROR, ITEMERR, LENGERR
```

The READQ TS command is used to retrieve data from a temporary storage queue. The queue is identified in the QUEUE option.

The INTO option can be used to specify the area into which the data is to be placed. The LENGTH option must specify a data area that contains the maximum length of record that the program will accept. If the record length exceeds the specified maximum length, the record is truncated and the LENGERR condition occurs. After the retrieval operation, the data area specified in the LENGTH operand is set to the record length (before any truncation occurred).

Alternatively, a pointer reference can be specified in the SET option. CICS/VS then acquires an area of sufficient size to hold the

record and sets the pointer reference to the address of the record. After the retrieval operation, the data area specified in the LENGTH option is set to the record length.

The ITEM and NEXT options are used to specify which record within a queue is to be read. If the ITEM option is specified, the record with the specified item number is retrieved. If the NEXT option is specified, the next record after the last record to be retrieved (by any task) is retrieved.

The following example shows how to read the first (or only) record from a temporary storage queue into a data area specified in the request:

```
EXEC CICS READQ TS                  Read from temporary storage queue
     QUEUE(UNIQNAME)                Queue name
     INTO(DATA)                     Data area
     LENGTH(LDATA)                  Length program will accept
```

The following example shows how to read the next record from a temporary storage queue into a data area provided by CICS/VS; the pointer reference specified by the SET option is set to the address of the storage area reserved for the data record.

```
EXEC CICS READQ TS                  Read from temporary storage queue
     QUEUE(DESCRQ)                  Queue name
     SET(PREF)                      Request pointer reference set
     LENGTH(LENG)                   Length of data retrieved
     NEXT                           Specify next record in queue
```

DELETEQ TS Command

```
DELETEQ TS QUEUE(name)

Exceptional condition:  ERROR
```

This command is used to delete all of the temporary data associated with a particular temporary storage queue. All storage associated with the queue is freed.

Temporary data should be deleted at the earliest possible time to avoid using too much storage.

Temporary Storage Control Options

FROM(data-area)
: specifies the data that is to be written to temporary storage.

INTO(data-area)
specifies the user data area into which the data is to be written. The data area may be any variable, array, or structure. If this option is specified, move-mode access is implied.

ITEM(parameter)
specifies a halfword binary value to be used with WRITEQ TS and READQ TS commands.

When used with a WRITEQ TS command in which the REWRITE option is not specified, the parameter must be a data area that is to be set to the item number assigned to this record in the queue. If the REWRITE option is specified, the data area specifies the item in the queue that is to be replaced.

When used with a READQ TS command, the ITEM option specifies the item number of the logical record to be retrieved from the queue. The parameter must be a data value that is to be taken as the relative number of the logical record to be retrieved. This number may be the number of any item that has been written to the temporary storage queue.

LENGTH(parameter)
specifies a halfword binary value to be used with WRITEQ TS and READQ TS commands.

For a WRITEQ TS command, the parameter may be a data value that is the length of the data that is to be written.

For a READQ TS command with the INTO option, the parameter must be a data area that specifies the maximum length of data that the program can accept. If the value specified is less than zero, zero is assumed. If the length of the data exceeds the value specified, the data is truncated to that value and the LENGERR condition occurs. On completion of the retrieval operation, the data area is set to the original length of the data.

For a READQ TS command with the SET option, the parameter must be a data area. On completion of the retrieval operation, the data area is set to the length of the data.

QUEUE(name)
specifies the symbolic name of the queue to be written to, read from, or deleted. The name must be alphameric, up to eight characters in length, and must be unique within the CICS/VS system. Do not use hexadecimal 'FA' through 'FF' as the first character of the name; these characters are reserved for CICS/VS use.

Temporary Storage Control Exceptional Conditions

ITEMERR
occurs when the item number specified or implied by a READQ TS command, or a WRITEQ TS command with the REWRITE option, is invalid (that is, outside the range of entry numbers assigned for the queue); or if the queue referred to in a WRITEQ TS cannot be found.

Default action: terminate the task abnormally.

LENGERR

occurs if the length of the stored data is greater than the value specified by the LENGTH option for move-mode input operations.

Default action: terminate the task abnormally.

ERROR

See under "Exceptional Condition Handling" for an explanation of the ERROR condition.

Program Control

Program control governs the flow of control between application programs in a CICS/VS system. The name of an application program referred to in a program control command must have been placed in the processing program table (PPT) by the system programmer before CICS/VS is started (see Chapter 3.4).

Program control commands are provided to:

- Link one user-written application program to another, anticipating return to the requesting program (LINK).
- Transfer control from one user-written application program to another, with no return to the requesting program (XCTL).
- Return control from one user-written application program to another or to CICS/VS (RETURN).

Application Program Logical Levels

Application programs running under CICS/VS are executed at various logical levels. The first program to receive control within a task is at the highest logical level. When one application program is linked to another, expecting an eventual return of control, the linked-to program is considered to reside at the next lower logical level. When control is simply transferred from one application program to another, without expecting return of control, the two programs are considered to reside at the same logical level. Figure 2.1-1 illustrates the concept of logical levels.

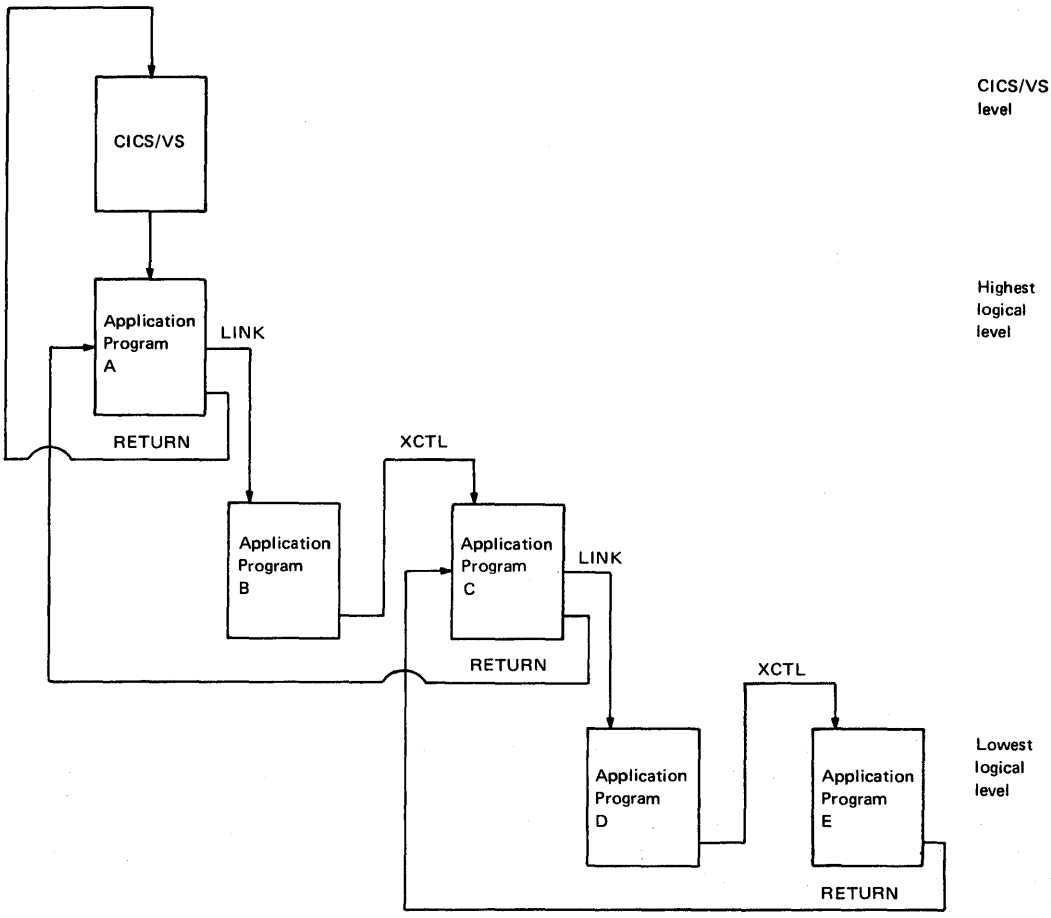


Figure 2.1-1. Application Program Logical Levels

LINK Command (Anticipating Return)

```
LINK PROGRAM(name)
     [COMMAREA(data-area) LENGTH(data-value)]

Exceptional condition:  ERROR
```

The LINK command is used to pass control from an application program at one logical level to an application program at the next lower logical level. If the linked-to program is not already in main storage, it will be loaded. When the RETURN command is executed in the linked-to program, control is returned to the program initiating the linkage, at the next sequential executable instruction.

The following example shows how to request a link to an application program called PROG1:

```
EXEC CICS LINK                    Link to a program
     PROGRAM('PROG1')             Program name
```

The COMMAREA option can be used to pass data to the linked-to program. For further details, see "Passing Data Between Programs" later in this section. The LENGTH option specifies the length of the data being passed.

The linked-to program operates independently of the program that issues the LINK command. For example, the effects of any HANDLE CONDITION or HANDLE AID commands in the linking program are not inherited by the linked-to program, but the original HANDLE CONDITION and HANDLE AID commands are restored on return to the linking program.

XCTL Command (without Return)

```
XCTL PROGRAM(name)
     [COMMAREA(data-area) LENGTH(data-value)]

Exceptional condition:  ERROR
```

The XCTL command is used to transfer control from one application program to another at the same logical level. The program from which control is transferred is released. If the program to which control is transferred is already in main storage, it will be reloaded only if it is an RPG II program.

The following example shows how to request a transfer of control to an application program called PROG2:

```
EXEC CICS XCTL                    Transfer control
     PROGRAM('PROG2')             Program name
```

The COMMAREA option can be used to pass data to the invoked program. For further details, see "Passing Data Between Programs" later in this section. The LENGTH option specifies the length of the data to be passed.

RETURN Command

```
RETURN [TRANSID(name) [COMMAREA(data-area) LENGTH(data-value)]]

Exceptional condition:  ERROR
```

The RETURN command is used to return control from an application program either to an application program at the next higher logical level or to CICS/VS.

When the command is issued in a lower-level program, the program to which control is returned will have relinquished control by issuing a LINK command and will reside one logical level higher than the program returning control.

When the command is issued in a program at the highest logical level, control returns to CICS/VS. If the task is associated with a terminal,

the TRANSID option can be used to specify the transaction identifier for the next program to be associated with that terminal; this causes subsequent input entered from the terminal to be interpreted wholly as data. (See "Pseudo-conversational Programming," below.) In addition, the COMMAREA option can be used to pass data to the new transaction. Further details follow, under "Passing Data Between Programs." The LENGTH option specifies the length of the data to be passed. The COMMAREA and LENGTH options can be used only when the RETURN command is returning control to CICS/VS.

Pseudo-conversational Programming

Pseudo-conversational programming is the term used to describe a technique that allows a conversational effect to be achieved without the overhead of true conversational programming. It is advocated for entry level system users because it allows transactions to free resources while awaiting input from terminals.

Instead of a task remaining in existence to communicate with the terminal, it is allowed to terminate, having first identified the next transaction to be associated with the terminal. (See the TRANSID option of the RETURN command.) When data is next sent from the terminal, the identified transaction is invoked to continue the session.

Passing Data Between Programs

This section describes how data can be passed when control is passed to another program by a program control request. (Data can be passed between application programs and transactions in other ways. For example, the data can be stored in a CICS/VS storage area outside the local environment of the application program, such as the transaction work area (TWA); see "Access to System Information" in this section for details. Another way is to store the data in temporary storage; see "Temporary Storage Control" in this section.)

The COMMAREA option of the LINK, XCTL, and RETURN commands can specify the name of a data area (known as a communication area) to be used to pass data to the program being invoked or to the transaction identified in the TRANSID option. (The TRANSID option specifies a new transaction that will be initiated when input is next received from the terminal associated with the current transaction.) The length of the communication area is specified in the LENGTH option.

The invoked program receives the data as a parameter. The program must contain a definition of a data area to allow access to the passed data. The data area is subject to conventions that differ according to programming language. See the appropriate language-specific section of this manual for details.

The data area need not be of the same length as the original communication area; if access is required only to the first part of the data, the new data area can be shorter, but must not be longer.

The invoked program can determine the length of any communication area that has been passed to it by accessing the EIBCALEN field in the EIB of the task. If no communication area has been passed, the value of EIBCALEN will be zero; otherwise, EIBCALEN will always contain the value specified in the LENGTH option of the LINK, XCTL, or RETURN command, regardless of the size of the data area in the invoked program.

When a communication area is passed by means of a LINK command, the invoked program is passed a pointer to the communication area itself. Any changes made to the contents of the data area in the invoked program are available to the invoking program, when control returns to it; to access any such changes, the program names the data area specified in the original COMMAREA option. When a communication area is passed by means of a RETURN command, a copy of the communication area is made, and addressability to the copy is passed to the invoked program.

The invoked program can access the EIBFN field in the EIB to determine which type of request invoked the program. Any testing of the field must be carried out before any CICS/VS commands are issued. If a LINK or XCTL request invoked the program, the approprate code will be found in the field; if a RETURN request was used, no CICS/VS commands will have been issued in the task, and the field will contain zeros.

The following example shows how a LINK command causes data to be passed to the program being linked to; the XCTL command is coded in a similar way.

Two programs are named PROG1 and PROG2.

In PROG1:

A structure named COMREGION includes, as its first element, a three-character data area named FIELD.

Place the value 'ABC' in FIELD.

```
EXEC CICS LINK PROGRAM('PROG2') COMMAREA(COMREGION) LENGTH(3)
.
.
.
```

In PROG2:

Overlay the storage for COMREGION in PROG1 with a similar structure whose first element is a three-character data area named FIELD. (Note: The technique varies according to language. For example, PL/I uses a based structure in PROG2, picking up a pointer to COMREGION as a parameter in the PROG2 PROCEDURE statement; COBOL programs must name the structure as DFHCOMMAREA in the linkage section of PROG2. See the appropriate language-specific section for further information.)

Test for EIBCALEN greater than zero and FIELD equal to 'ABC'.

```
.
.
.
```

The next example shows how a RETURN command causes data to be passed to a new transaction.

Two programs are named PROG1 and PROG2.

In PROG1:

A structure named TERMSTOR includes, as its first two elements, a 3-character data area named FIELD and a 17-character data area named DATAFLD.

Place the value 'XYZ' in FIELD.

```
EXEC CICS RETURN TRANSID('TRN2')
     COMMAREA(TERMSTOR) LENGTH(20)

.
.
.
```

In PROG2:

Overlay the storage for TERMSTOR in PROG1 with a similar structure.

Test for EIBCALEN greater than zero and FIELD equal to 'XYZ'.

```
.
.
.

EXEC CICS RETURN
```

Program Control Options

COMMAREA(data-area)
: specifies a communication area that is to be made available to the invoked program. For LINK commands, a pointer to the data area is passed; for XCTL and RETURN commands, because the data area is freed before the next program is invoked, a copy of the data area is created and a pointer to the copy is passed.

LENGTH(parameter)
: specifies a halfword binary value to be used with LINK, XCTL, and RETURN commands. The parameter must be a data value that is the length of the communication area. If a negative value is supplied, zero is assumed.

PROGRAM
: specifies the name of the program to which control is to be passed. The name may be up to eight characters long, must be enclosed in single quotes(' '), and must have been defined in the processing program table (PPT).

TRANSID(name)
specifies the transaction identifier to be used with the next input message entered from the terminal with which the task that issued the RETURN command has been associated. The specified name must consist of up to four characters and must have been defined in the program control table (PCT).

Program Control Exceptional Conditions

ERROR
See under "Exceptional Condition Handling" for an explanation of the ERROR condition.

Interval Control

CICS/VS maintains a time-of-day clock so that various interval control functions can be performed at the correct time; such functions are called time-controlled functions. The time of day is obtained from the operating system at intervals whose frequency, and thus the accuracy of the time-of-day clock, depends on the transaction mix and the frequency of transaction switching operations.

Interval control commands are provided to:

- Request the current date and time of day (ASKTIME).
- Initiate a transaction and store data for the transaction (START).
- Retrieve data stored for a transaction (RETRIEVE).
- Cancel the effect of previous interval control commands (CANCEL).

Specifying Expiration Times

The time at which a time-controlled function is to be performed is called the expiration time. Expiration times can be specified absolutely, as a time of day, or as an interval that is to elapse before the function is to be performed.

If the specified expiration time is later (by up to 18 hours) than the current clock time, the requested function is performed when the specified time occurs. If the specified expiration time is the same as the current clock time, or up to and including six hours preceding the current clock time, the specified time is considered to have expired and the requested function is performed immediately. If the specified expiration time precedes the current clock time by more than six hours, the requested function is performed the next day at the specified time.

Specifying Request Identifiers

As a means of symbolically identifying the request and any data associated with it, a unique request identifier is assigned to each START command. The application programmer can specify his own request identifier by means of the REQID option; if none is assigned by the programmer, then for START commands only, CICS/VS assigns a unique request identifier and places it in the field EIBREQID in the EXEC interface block (EIB). A request identifier should be specified by the application programmer if the request may be canceled at some later time (see "CANCEL Command," later in this section).

ASKTIME Command

The ASKTIME command is used to update the CICS/VS time-of-day clock, and the fields EIBDATE and EIBTIME in the EXEC interface block. The two fields initially contain the date and time when the transaction was initiated.

START Command

```
START TRANSID(name)
      [INTERVAL(hhmmss) | INTERVAL(0) | TIME(hhmmss)]
      [TERMID(name)]
      [REQID(name)]
      [FROM(data-area) LENGTH(data-value)]

Exceptional condition:  ERROR
```

The START command is used to initiate a transaction at a specified time and, optionally, to pass data to that transaction. The application programmer provides the identifier of the transaction, the location of any data to be stored, and, if the transaction must communicate with a terminal, a terminal identifier. CICS/VS stores the data and queues the request until the specified time occurs. Then, as soon as all necessary resources are available (for example, a terminal), the transaction is initiated. Only one transaction is initiated if several START requests for the same transaction and terminal expire at the same time or prior to terminal availability.

A transaction can be initiated immediately by specifying an expiration time equal to the current time of day (or more than 18 hours later than it) explicitly or by using a zero time interval value. (This provides a way of initiating a transaction to send data to a local printer.)

The following example shows how to request initiation of a specified transaction not associated with a terminal:

```
EXEC CICS START                 Request transaction initiation
     TRANSID('TRNL')            Transaction identifier
     INTERVAL(10000)            Start transaction in one hour
     REQID(NONGL)               Request identifier
```

The next example shows how to request initiation of a transaction associated with a terminal. Since no request identifier is specified in this example, CICS/VS assigns one and makes it available to the application program by placing it in the EIBREQID field of the EXEC interface block.

```
EXEC CICS START                 Request transaction initiation
     TRANSID('TRN1')            Transaction identifier
     TIME(PACKTIME)             Expiration time
     TERMID('STA5')             Terminal identifier
```

Passing Data between Transactions

Data is passed to a new transaction by specifying the FROM and LENGTH options. The new transaction retrieves data by issuing the RETRIEVE command, which is described later. If the transaction to be started is not associated with a terminal, each START request results in a separate transaction being started. Therefore, only one data record can be passed to a transaction that is not associated with a terminal.

If the transaction to be started is associated with a terminal, it is possible to pass many data records to the new transaction by issuing further START requests having the same transaction identifier, terminal identifier, and expiration time as the original request.

If, owing to some delay in obtaining resources (for example, the terminal), a transaction is not initiated at the correct time, other START requests may have expired by the time the transaction is initiated. If these requests have the same transaction identifier and terminal identifier as the original request, only the one transaction is initiated; the data stored by the other requests becomes available to this transaction. So does data stored by requests that expire while this transaction is still active. If any data made available thus is still unretrieved at the end of the transaction, a new transaction is initiated.

The following example shows how to request initiation of a transaction associated with a terminal and request that data be made available to it:

```
EXEC CICS START                    Request transaction initiation
     TRANSID('TRN2')               Transaction identifier
     TIME(173000)                  Expiration time 1730
     TERMID('STA3')                Terminal identifier
     REQID(DATAREC)                Request identifier field
     FROM(DATAFLD)                 Data address
     LENGTH(100)                   Data length
```

RETRIEVE Command

```
RETRIEVE {INTO(data-area) | SET(pointer-ref)}
         LENGTH(data-area)

Exceptional conditions:  ERROR, LENGERR, NOTFND, ENDDATA
```

The RETRIEVE command is used to retrieve data stored by expired START commands. This command is the only way to retrieve such data, and it should be issued only by a task started in response to a START command.

The INTO option is used to specify the area into which the data is to be placed. The LENGTH option must specify a data area that contains the maximum length of record that the application program will accept. If the record length exceeds the specified maximum, it is truncated and the LENGERR condition occurs. After the retrieval operation, the data area specified in the LENGTH option is set to the record length (before any truncation occurred).

Alternatively, a pointer reference can be specified in the SET option. CICS/VS then acquires an area large enough to hold the record and sets the pointer reference to the address of that area. After the retrieval operation, the data area specified in the LENGTH option is set to the record length.

A transaction that is not associated with a terminal can access only the single data record associated with the original START command; it does so by issuing a RETRIEVE command. The storage occupied by the data associated with the transaction is freed on execution of the RETRIEVE command, or upon termination of the transaction if no RETRIEVE command is executed prior to termination.

A transaction that is associated with a terminal can access all data records associated with all expired START commands having the same transaction identifier and terminal identifier as the START command that initiated the transaction; it does so by issuing consecutive RETRIEVE commands. Expired data records are presented to the transaction upon request in expiration time sequence, starting with any data stored by the command that started the transaction, and including data from any commands that have expired since the transaction started. When all expired data records have been retrieved, the ENDDATA exceptional condition occurs. The storage occupied by the single data record associated with a START command is released after the data has been retrieved by a RETRIEVE command; any storage occupied by data that has not been retrieved is released when the CICS/VS system is terminated.

The following example shows how to request retrieval of a data record stored for a transaction into a specified data area:

```
EXEC CICS RETRIEVE          Retrieve time-ordered data
     INTO(DATAFLD)          User-provided data area
     LENGTH(LENG)           Length program will accept
```

The next example shows how to request retrieval of a data record stored for a transaction into a data area provided by CICS/VS, which sets the pointer reference PREF to the address of the area.

```
EXEC CICS RETRIEVE          Retrieve time-ordered data
     SET(PREF)              Request pointer reference set
     LENGTH(LENG)           Set to length of data
```

CANCEL Command

```
CANCEL REQID(name)

Exceptional conditions:  ERROR, NOTFND
```

The CANCEL command is used to cancel a previously issued START command. The effect of the cancellation is as if the original request had never been made. The cancellation request is effective only prior to expiration of the original request.

Interval Control Options

FROM(data-area)
specifies the data that is to be stored for a transaction that is to be initiated later.

INTERVAL(hhmmss)
specifies the time that is to elapse from the issuing of the command to the initiation of the transaction. When the command is executed, CICS/VS calculates the expiration time by adding the specified time to the current clock time. If the calculated time of day is the same as the current clock time, or more than 18 hours later than it, the specified time is considered to have expired.

This option is used in START commands, to specify when a new transaction should be initiated.

The time interval is specified in the form "hhmmss" where "hh" represents hours from 00 to 99, "mm" represents minutes from 00 to 59, and "ss" represents seconds from 00 to 59.

INTO(data-area)
specifies the user data area into which retrieved data is to be written. If this option is specified, move-mode access is implied.

LENGTH(parameter)
specifies a halfword binary value to be used with START and RETRIEVE commands.

For a START command, the parameter may be a data value that is the length of the data that is to be stored for the new transaction.

For a RETRIEVE command with the INTO option, the parameter must be a data area that specifies the maximum length of data that the program is prepared to handle. If the value specified is less than zero, zero is assumed. If the length of the data exceeds the value specified, the data is truncated to that value and the LENGERR condition occurs. On completion of the retrieval operation, the data area is set to the original length of the data.

For a RETRIEVE command with the SET option, the parameter must be a data area. On completion of the retrieval operation, the data area is set to the length of the data.

REQID(name)
specifies a unique name to identify a request.

This option can be used in a START command when another transaction is to be provided with the capability of canceling an unexpired request, and in CANCEL commands.

The length of the name must be less than or equal to eight characters. If this option is omitted from a START command, CICS/VS generates a unique request identifier in the EIBREQID field of the EXEC interface block.

SET(pointer-ref)
specifies the pointer reference to be set to the address location of retrieved data. This option implies locate-mode access.

TERMID(name)
specifies the symbolic identifier of the terminal associated with a transaction to be initiated as a result of a START command. This option is required when the transaction to be initiated must communicate with a terminal; it should be omitted otherwise. The name must be alphameric, up to four characters in length, and must have been defined in the terminal control table (TCT) by the system programmer (see Chapter 3.4).

TIME(hhmmss)
specifies the expiration time. If the specified time is the same as the current clock time, or up to and including six hours preceding the current clock time, the specified time is considered to have expired.

This option is used in START commands to specify when a new transaction should be started.

The time is specified in the form "hhmmss" where "hh" represents hours from 00 to 99, "mm" represents minutes from 00 to 59, and "ss" represents seconds from 00 to 59.

TRANSID(name)
specifies the symbolic identifier of the transaction to be initiated as the result of a START command. The name must be up to four characters in length and must have been defined in the program control table (PCT) by the system programmer (see Chapter 3.4).

Interval Control Exceptional Conditions

ENDDATA
occurs if no more data is stored for a task issuing a RETRIEVE command. It can be considered a normal end-of-file response when retrieving data records sequentially.

Default action: terminate the task abnormally.

LENGERR
occurs in a move-mode retrieval operation if the length specified is less than the actual length of the stored data.

Default action: terminate the task abnormally.

NOTFND
occurs if any of the following situations exists:

- The request identifier specified in a CANCEL command fails to match an unexpired time-controlled-function command.

- The RETRIEVE command is issued by a transaction that was not initiated in response to a START command.
- The request identifier associated with a START command fails to remain unique; when a RETRIEVE command is issued, CICS/VS cannot determine where the data is stored.

Default action: terminate the task abnormally.

ERROR

See under "Exceptional Condition Handling" for an explanation of the ERROR condition.

Exceptional Condition Handling

Exceptional conditions may occur during the execution of a CICS/VS command and, unless specified otherwise in the application program, a default action for each condition will be taken automatically by CICS/VS. Usually, this default action is to terminate the task abnormally. However, to prevent abnormal termination, the application program can issue a HANDLE CONDITION command. The command must include the name of the condition and, optionally, a label to which control is to be passed if the condition occurs.

The HANDLE CONDITION command for a given condition applies only to the program in which it is specified, remaining active until the associated task is terminated, or until another HANDLE CONDITION command for the same condition is encountered, in which case the new command overrides the previous one.

When control returns to a program from a program at a lower logical level, the HANDLE CONDITION commands that were active in the higher-level program before control was transferred are reactivated, and those in the lower-level program are deactivated. (Information about logical levels is given earlier in this section.)

Some exceptional conditions can occur during the execution of any one of a number of unrelated CICS/VS commands. If the same action is required for all occurrences, a single HANDLE CONDITION command at the beginning of the program will suffice. If different actions are required, HANDLE CONDITION commands, specifying different labels, at appropriate points in the program will suffice. The same label can be specified for all commands and EIBFN and EIBRCODE (fields in the EXEC interface block) can be tested to find out which exceptional condition has occurred and in which command. The EIB is described in Appendix B.

The ERROR Exceptional Condition

Apart from the exceptional conditions associated with individual commands, there is a general exceptional condition named ERROR whose default action also is to terminate the task abnormally. If no HANDLE CONDITION command is active for a condition, but one is active for ERROR, control will be passed to the label specified for ERROR. A HANDLE CONDITION command (with or without a label) for a condition overrides the HANDLE CONDITION ERROR command for that condition.

Do not issue any commands in an error routine that may give rise to the same condition that caused the branch to the routine; be especially careful not to cause a loop on the ERROR condition. A way to avoid this would be to issue a HANDLE CONDITION ERROR command — without a label — at the start of any routine that handles the ERROR condition. This would result in system default action for any recurrence of ERROR while in the ERROR routine.

HANDLE CONDITION COMMAND

```
HANDLE CONDITION condition[ (label) ]
                 [condition[ (label) ]]...
```

The HANDLE CONDITION command is used to specify the label to which control is to be passed if an exceptional condition occurs. No more than twelve conditions are allowed in the same command. The ERROR condition can also be used to specify that other conditions are to cause control to be passed to the same label. If label is omitted, the default action for the condition will be taken.

The following example shows the handling of exceptional conditions, such as DUPREC, LENGERR, and so on, that can occur when a WRITE command is used to add a record to a data set. In this example, DUPREC is to be handled as a special case; system default action is to be taken for LENGERR; and all other conditions are to be handled by the generalized error routine ERRHANDL.

```
EXEC CICS HANDLE CONDITION      Handle exceptional conditions
          ERROR(ERRHANDL)       General label
          DUPREC(DUPRTN)        Label of duplicate-record routine
          LENGERR               Default action requested
```

HANDLE CONDITION Command Options

condition[(label)]

"condition" specifies the name of the exceptional condition, and "label" specifies the location within the program to be branched to if the condition occurs. If this option is not specified, the default action for the condition is taken, unless the default action is to terminate the task abnormally, in which case the ERROR condition occurs. If the option is specified without a label, any HANDLE CONDITION command for the condition is deactivated, and the default action is taken if the condition occurs.

LIST OF EXCEPTIONAL CONDITIONS

The following list shows exceptional conditions that can occur during the execution of CICS/VS commands. Each condition is followed by one or more keywords, showing CICS/VS commands during the execution of which the condition may occur. For the meaning of a condition, and the default action associated with that condition, refer to the list of exceptional conditions given at the end of each subsection. Note that some error conditions that can occur are not documented in this manual; however they can be covered by the ERROR condition. A full list of error condition is given in the Application Programmer's Reference Manual (Command Level). The EIBRCODE field of the EXEC Interface Block (see Appendix B) contains a code related to the conditions.

DUPKEY

READ (VSAM), READNEXT

DUPREC

WRITE

ENDDATA

RETRIEVE

ENDFILE
READNEXT, READPREV

ERROR
— can occur during the execution of any command

ITEMERR
READQ TS, WRITEQ TS

LENGERR
READ, READNEXT, READPREV, READQ TD, READQ TS, RETRIEVE, REWRITE, WRITE

MAPFAIL
RECEIVE MAP

NOTFND
CANCEL, DELETE, READ, READNEXT, READPREV, RETRIEVE

QZERO
READQ TD

Chapter 2.2. Assembler Language Programming

CICS/VS acts as an interface between assembler language application programs and the operating system. When an assembler language application program is designed to run under CICS/VS, certain commands can be replaced by CICS/VS commands. This substitution is usually mandatory. In particular, applications must always use CICS/VS commands to perform input and output operations.

The first part of this chapter describes the use of CICS/VS commands in assembler language programming, and lists some programming rules. The second part demonstrates the principles by describing, listing, and analysing a sample program.

Translator Invocation

For details of translator invocation, see Chapter 3.5, "Preparation of Application Programs."

Assembler Language Translator Options

The translator provides optional facilities, which can be requested by job control statements. Some of the options have default values.

Translator options are specified in the *ASM job control statement, and are written as a list within the CICS keyword option.

The *ASM statement must obey the same syntax and continuation rules as the assembler comment statement.

Translator Options	Default
NOSPIE	-

NOSPIE
: is used to prevent the translator from trapping unrecoverable errors; instead, a dump is produced.

Command Syntax

The format of a CICS/VS command is as follows:

1. The keyword EXECUTE or its abbreviation, EXEC. This keyword must appear in an operator position.

2. The identifier CICS

3. The function

4. A sequence of options. The options may be written in any order. They may be separated by either a comma or a blank.

5. Remarks may be added to a line by following an option with either a comma or a period, then a blank, and then the remark.

6. Continuation follows normal Assembler language conventions: there must be a non-blank character in column 72, and continuation lines must start in column 16.

The general format is:

```
{EXECUTE|EXEC} CICS function [option]...
```

General Rules for Assembler Language Programming

For an assembler language application program, each EXEC CICS command is replaced by an invocation of the DFHEICAL macro. The DFHEICAL macro builds an argument list in dynamic storage, so that the application program is reentrant, and then invokes the EXEC interface program. A definition of this dynamic storage is provided automatically by the translator inserting an invocation of the macros DFHEISTG and DFHEIEND. The translator will also insert an invocation of the DFHEIENT macro, which performs prolog initialization code and an invocation of the DFHEIRET macro which performs epilog code.

The following example shows a simple assembler language application program that sends a BMS map to a terminal.

```
INSTRUCT CSECT
         EXEC CICS SEND MAP('XDFHAMA') MAPONLY ERASE
         END
```

which is translated to:

```
INSTRUCT CSECT
         DFHEIENT                    INSERTED BY TRANSLATOR
*        EXEC CICS SEND MAP('XDFHAMA') MAPONLY ERASE
         DFHEICAL (23,5),('1804C000080000000046204000020','XDFHAMA',DF *
               HEIV00)
         DFHEIRET                    INSERTED BY TRANSLATOR
         DFHEISTG                    INSERTED BY TRANSLATOR
         DFHEIEND                    INSERTED BY TRANSLATOR
         END
```

The dynamic storage that is obtained for building the parameter list may be extended by the user to provide reentrant storage for assembler variables. The following example shows a simple assembler language application program that uses variables in dynamic storage.

```
DFHEISTG DSECT
         COPY  XDFHAMA             INPUT MAP DSECT
         COPY  XDFHAMB             OUTPUT MAP DSECT
MESSAGE  DS    CL39
```

```
INQUIRY  CSECT
         EXEC  CICS RECEIVE MAP('XDFHAMA')
         MVC   NUMBO,KEYI
         MVC   MESSAGE,=CL(L'MESSAGE)'THIS IS A MESSAGE'
         EXEC  CICS SEND MAP('XDFHAMB')
         END
```

which is translated to:

```
DFHEISTG DSECT
         DFHEISTG                  INSERTED BY TRANSLATOR
         COPY  XDFHAMA             INPUT MAP DSECT
         COPY  XDFHAMB             OUTPUT MAP DSECT
MESSAGE  DS    CL39
INQUIRY  CSECT
         DFHEIENT                  INSERTED BY TRANSLATOR
*        EXEC CICS RECEIVE MAP('XDFHAMA')
         DFHEICAL (23,5),('1802C000080000000040900000020','XDFHAMA',XD *
               FHAMAI)
         MVC   NUMBO,KEYI
         MVC   MESSAGE,=CL(L'MESSAGE)'THIS IS A MESSAGE'
*        EXEC CICS SEND MAP('XDFHAMB')
         DFHEICAL (23,5),('1804C000080000000004E004000020','XDFHAMA',XD *
               FHAMBO)
         DFHEIRET                   INSERTED BY TRANSLATOR
         DFHEISTG                   INSERTED BY TRANSLATOR
         DFHEIEND                   INSERTED BY TRANSLATOR
         END
```

The use of the reserved name DFHEISTG as the DSECT name indicates that dynamic storage is to be provided for the extra user variables within that named DSECT.

The invocation of an assembler language application program using the command level interface obeys system standards and the invocation of the EXEC interface program by an EXEC CICS command also obeys system standards. Details are given below.

On entry to an assembler language program using the command level interface;

```
Register 1   contains the address of the parameter list.
Register 15  contains the address of the entry point.
Register 14  contains the address of the return point.
Register 13  contains the address of a save area.
```

The parameter list consists of two entries, as follows:

- Address of the EXEC interface block.
- Address of the COMMAREA or zeros if it does not exist.

A copy book, DFHEIBLK, containing a DSECT which describes the EXEC interface block is automatically included.

Each EXECUTE CICS command is replaced by an invocation of the DFHEICAL macro which expands to a system-standard call sequence using the following registers:

```
Register 15  contains the entry point of the EXEC interface program.
Register 14  contains the return address in the application program.
Register 0   is undefined.
Register 1   contains the address of the parameter list.
```

The entry point is resolved in a stub (DFHEAI) which must be link-edited with the application program.

Storage for the parameter list is provided automatically by the translator, which inserts invocations of the macros DFHEISTG and DFHEIEND. These macros define the storage required for the parameter list and a save area. The translator also inserts an invocation of the DFHEIENT macro after the first CSECT or START statement. This macro saves registers, obtains an initial allocation of the storage defined by DFHEISTG, sets up a base register (default register 3), a dynamic storage register (default register 13), and a register to address the EXEC interface block (default register 11).

Exit from the assembler language program can be achieved by the EXEC CICS RETURN command or by the DFHEIRET macro, which is inserted by the translator before the END statement to restore registers and return to the address in register 14.

The dynamic storage defined by DFHEISTG can be extended by the user to provide reentrant storage for user variables. This is done by defining the user variables in a DSECT with the reserved name DFHEISTG. The translator inserts the DFHEISTG macro after the DFHEISTG DSECT statement. In this way the DSECT finally describes dynamic storage consisting of the parameter list area, other areas needed by the EXEC interface, and space for user variables.

The user may also modify or extend the defaults used by the DFHEIENT macro by coding the macro himself. The macro can have up to three keyword arguments, as follows:

CODEREG – base register
DATAREG – dynamic storage register
EIBREG – register to address the EXEC interface block.

and must be coded instead of the first CSECT or START statement, as shown in the following example:

```
INSTRUCT  DFHEIENT  CODEREG=(2,3,4),DATAREG=5,EIBREG=6
```

The symbolic register DFHEIPLR is equated to the first DATAREG either explicitly specified or obtained by default. DFHEIPLR will be assumed by the expansion of an EXEC command to contain the value set up by DFHEIENT. It is the user's responsibility to either dedicate this register or ensure that it is restored before each EXEC command.

Restrictions

The following assembler language features cannot be used in an application program designed for use with CICS/VS.

1. The assembler language instructions COM (Common control section), ICTL, and OPSYN.
2. Private code must not contain EXEC commands.

EXEC Commands Inside Macros and COPY Code

Macros can generate EXEC commands and COPY code can contain EXEC commands, but such macros and COPY code must be translated and stored in the source library in translated form for later inclusion by the assembler.

Invoking Assembler Application Programs by a CALL Statement

Assembler application programs containing EXEC commands can be treated as separate CICS/VS programs that have their own PPT entries and that can be invoked by assembler language, COBOL, RPGII, or PL/I programs using EXEC CICS LINK or XCTL commands (see Chapter 2.1).

However, since assembler application programs containing EXEC commands are invoked by a system standard call, they can also be invoked by a COBOL, RPGII, or PL/I CALL statement or by an assembler CALL macro. A single CICS/VS program with one PPT entry may consist of a module containing separate CSECTs linked together, although they may have been compiled or assembled separately.

Also, assembler language application programs containing EXEC commands can be linked with other assembler language programs, or with programs in one of the high level languages (HLL), COBOL, RPGII, or PL/I, but with only one. When an HLL program is linked with an assembler language program the main program, and the language specified in the PPT, must be the HLL program.

Since assembler language application programs containing EXEC commands are always passed the two parameters EIB and COMMAREA when they are invoked, the CALL statement or macro must pass these two parameters followed, optionally, by other parameters.

Description of UPDATE Sample Program

The update sample program combines the facilities of file update, file add and file inquiry.

The update program maps in the account number and reads the file record. The required fields from the file area, and a title depending on the invoking transaction-identifier, are moved to the map area. In the case of the file add function being required, the number entered onto map XDFHAMA and a title are moved to the map area of XDFHAMB. Then XDFHAMB, containing the record fields, is displayed at the terminal. If the function of this transaction is file inquiry, the program ends here.

The update program then reads and maps in the record to be added or updated, and edits the fields. The sample program only suggests the type of editing that might be done. The user should insert editing steps needed to ensure valid changes to the file. Those fields which have been changed are moved to the file area. Log information is moved to the transient data area. The file record is then either added or updated, depending on the function required of the program. Either the message "FILE UPDATED" or "RECORD ADDED" is inserted in XDFHAMA and the map is transmitted to the terminal.

Note: This program demonstrates a pseudo-conversational programming technique, where control is returned to CICS/VS along with a transaction

identifier whenever a response is requested from the operator. Associated with each return of control to CICS/VS is a storage area containing details associated with the previous invocation of this transaction.

Listing of UPDATE Sample Program and Maps

```
DFHEISTG DSECT
         COPY   XDFHAMA                MAP A
         COPY   XDFHAMB                MAP B
RETREG   EQU    2                     SET UP REGISTER USAGE
R06      EQU    6
R07      EQU    7
R08      EQU    8
R09      EQU    9
FILEDS   DS     0C
         COPY   FILEA                  RECORD DESCRIPTION FOR FILEA
COMPTR   EQU    4                     POINTER TO COMMAREA
         COPY   LOGA                   LOG FILE RECORD DESCRIPTION
         COPY   DFHBMSCA               BMS ATTRIBUTE BYTES
MESSAGES DS     CL39                  TEMP STORE FOR MESSAGES
KEYNUM   DS     CL9                   TEMP STORE FOR FILE RECORD KEY
COMLEN   DS     1H                    LENGTH OF COMMAREA
XDFHAALL CSECT
         CLC    EIBTRNID,=CL(L'EIBTRNID)'AINQ' IS INVOKING T-ID 'AINQ'?
         BE     OKTRANID               OK HERE, SO CONTINUE
         CLC    EIBTRNID,=CL(L'EIBTRNID)'AUPD' IS IT 'AUPD'?
1        BE     OKTRANID               OK HERE, SO CONTINUE
         CLC    EIBTRNID,=CL(L'EIBTRNID)'AADD' FINALLY, IS IT 'AADD'?
         BNE    ERRORS                 IF NOT, GO TO ERROR ROUTINE
OKTRANID DS     0H                     CORRECT INVOKING TRANSACTION ID HERE
2        LH     COMPTR,EIBCALEN        HAS A COMMAREA BEEN RETURNED?
         LTR    COMPTR,COMPTR
         BNZ    COMRETND               ...YES, SO GO GET MAP
3        EXEC CICS HANDLE CONDITION MAPFAIL(AMNU) ERROR(ERRORS)
4        EXEC CICS RECEIVE MAP('XDFHAMA')
         OC     KEYI,KEYI              IS KEYI HEX ZEROS?
         BZ     NOTFOUND               ..YES, SO TREAT AS NOT FOUND
5        MVC    KEYNUM,KEYI            ..NO, SO SAVE KEY TO FILE
         XC     XDFHAMBO(XDFHAMBE-XDFHAMBO),XDFHAMBO CLEAR MAP
         CLC    EIBTRNID,=CL(L'EIBTRNID)'AADD' IS INVOKING T-ID 'AADD'?
         BNE    INQUPD                 ..NO, SO GO TEST FOR OTHER ID'S
6        MVC    TITLEO,=CL(L'TITLEO)'FILE ADD' SET UP TITLE
         MVC    MSG3O,=CL(L'MSG3O)'ENTER DATA AND PRESS ENTER KEY'
         MVI    NUMBA,DFHBMFSE         SET MDT ON NUMBER
7        MVC    NUMB,KEYI              PUT KEY IN COMMAREA
         MVC    NUMBO,KEYI             ...AND ON MAP ENTRY
8        MVI    AMOUNTA,C'J'           NUMERIC AND MDT ATTRIBUTE BYTE
         MVC    AMOUNTO,=C'-0000.00'  PROMPTING FIELD FOR MAP
         MVC    COMLEN,=H'7'          SET UP LENGTH OF COMMAREA TO BE RTND
         BAL    RETREG,MAPSEND         GO SEND MAP
         B      CICSCONT               GO RETURN CONTROL TO CICS
INQUPD   DS     0H                     HERE INVOKING T-ID IS AINQ, OR AUPD
9        EXEC CICS HANDLE CONDITION NOTFND(NOTFOUND)
10       EXEC CICS READ DATASET('FILEA') INTO(FILEA) RIDFLD(KEYNUM)
11       CLI    STAT,X'FF'             IS RECORD CODED AS NOT FOUND?
         BE     NOTFOUND               ..YES, SO BRANCH TO NOTFOUND ROUTINE
         CLC    EIBTRNID,=CL(L'EIBTRNID)'AINQ' IS INVOKING T-ID AINQ?
         BNE    UPDTSECT               ..NO, SO BRANCH TO AUPD ROUTINE
12       MVC    TITLEO,=CL(L'TITLEO)'FILE INQUIRY' SET UP TITLE ON MAP
         MVC    MSG3O,=CL(L'MSG3O)'PRESS ENTER TO CONTINUE' SET UP TITLE
         BAL    RETREG,MAPBUILD        GO BUILD MAP
         BAL    RETREG,MAPSEND         GO SEND MAP
13       EXEC CICS RETURN TRANSID('AMNU')
UPDTSECT DS     0H                     UPDATE ROUTINE
14       MVC    TITLEO,=CL(L'TITLEO)'FILE UPDATE' SET UP MAP TITLE
         MVC    MSG3O,=CL(L'MSG3O)'CHANGE FIELDS AND PRESS ENTER'
15       MVC    COMLEN,=H'80'          STORE LENGTH OF COMMAREA
         BAL    RETREG,MAPBUILD        GO BUILD MAP
```

```
         BAL   RETREG,MAPSEND      GO SEND MAP
         B     CICSCONT            GO RETURN CONTROL TO CICS
*****************************************************************
*                                                               *
*    HERE A COMMAREA HAS BEEN RETURNED, AND IS THEREFORE SECOND *
*    INVOCATION OF THIS PROGRAM                                 *
*                                                               *
*****************************************************************
COMRETND DS    0H                  HERE COMMAREA HAS BEEN RETURNED
         L     COMPTR,DFHEICAP     GET ADRESSABILITY TO COMMAREA
16       EXEC CICS HANDLE CONDITION MAPFAIL(NOTMODF) ERROR(ERRORS)    *
               DUPREC(DUPREC) NOTFND(NOTFOUND)
17       EXEC CICS RECEIVE MAP('XDFHAMB')
         CLC   EIBTRNID,=CL(L'EIBTRNID)'AUPD' IS INVOKING T-ID AUPD?
         BNE   SECADD              ..NO, SO BRANCH TO SECOND AADD ROUT
18       EXEC CICS READ UPDATE DATASET('FILEA') INTO(FILEA)           *
               RIDFLD(NUMB-FILEDS(COMPTR))
19       CLC   FILEREC,FILEREC-FILEDS(COMPTR) RECORD CHANGED ON FILE?
         BE    OKREC               ..NO, SO BRANCH AND CONTINUE
20       MVC   MSG1O,=CL(L'MSG1O)'FILE ALREADY UPDATED - REENTER'
         MVI   MSG1A,DFHBMBRY      BRIGHTEN MESSAGE ON SCREEN
         MVI   MSG3A,DFHBMDAR      DARKEN OPERATOR INSTRUCTION
         BAL   RETREG,MAPBUILD     GO BUILD MAP
21       EXEC CICS SEND MAP('XDFHAMB') DATAONLY
         MVC   COMLEN,=H'80'       SET UP LENGTH OF COMMEREA
         B     CICSCONT            GO RETURN CONTROL TO CICS
OKREC    DS    0H                  HERE RECORD IS OK FOR UPDATE
         BAL   RETREG,CHECK        GO TEST RECORD TO BE UPDATED
         MVI   STAT,C'U'           MOVE 'UPDATE' BYTE TO FILE RECORD
         BAL   RETREG,FILESTUP     GO SET UP FILE RECORD
22       MVC   MESSAGES,=CL(L'MESSAGES)'FILE UPDATED' SET UP MESSAGE
         B     AMNU                COMPLETE, GO FINISH.
SECADD   DS    0H                  SECOND ADD ROUTINE
         MVC   NUMB,NUMB-FILEDS(COMPTR)   MOVE SAVED RECORD KEY TO FILE
         BAL   RETREG,CHECK        GO CHECK RECORD TO BE ADDED
         XC    FILEDS,FILEDS       RECORD IS OK HERE,SO CLEAR FILE AREA
         MVI   STAT,C'A'           MOVE 'ADDED' BYTE TO FILE RECORD
         BAL   RETREG,FILESTUP     GO WRITE RECORD ON FILE
23       MVC   MESSAGES,=CL(L'MESSAGES)'RECORD ADDED'  SET UP MESSAGE
         B     AMNU                COMPLETE, GO FINISH.
CICSCONT DS    0H                  THIS ROUTINE RETURNS CONTROL TO CICS
24       EXEC CICS RETURN TRANSID(EIBTRNID) COMMAREA(FILEDS)          *
               LENGTH(COMLEN)
AMNU     DS    0H                  ENDING ROUTINE
         XC    XDFHAMAO(XDFHAMAE-XDFHAMAO),XDFHAMAO CLEAR MAP
25       MVI   MSGA,DFHBMBRY       BRIGHTEN MESSAGE FIELD ON MAP
         MVC   MSGO,MESSAGES       MOVE ANY MESSAGE TO MAP AREA
26       EXEC CICS SEND MAP('XDFHAMA') ERASE
27       EXEC CICS RETURN
*****************************************************************
*                                                               *
*                  GENERAL ROUTINES                             *
*                                                               *
*****************************************************************
MAPBUILD DS    0H                  ROUTINE TO BUILD MAP XDFHAMB
         MVC   NUMBO,NUMB          MOVE FILE KEY TO MAP AREA
         MVC   NAMEO,NAME          MOVE NAME TO MAP AREA
         MVC   ADDRO,ADDRX         MOVE ADRESS TO MAP AREA
28       MVC   PHONEO,PHONE        MOVE PHONE TO MAP AREA
         MVC   DATEO,DATEX         MOVE DATE TO MAP AREA
         MVC   AMOUNTO,AMOUNT      MOVE AMOUNT TO MAP AREA
         MVC   COMMENTO,COMMENT    MOVE COMMENT TO MAP AREA
         BR    RETREG              RETURN
MAPSEND  DS    0H                  ROUTINE TO SEND MAP XDFHAMB
29       EXEC CICS SEND MAP('XDFHAMB') ERASE
```

```
         BR    RETREG               RETURN
CHECK    DS    0H                   ANY INPUT FROM SCREEN? ROUTINE
         LA    R06,XDFHAMBO         R6 POINTS TO START OF MAP XDFHAMB
         LA    R07,(XDFHAMBE-XDFHAMBO) R7 CONTAINS LENGTH OF MAP B
         LA    R08,HEXZERO          R8 POINTS TO HEXZERO
         LA    R09,L'HEXZERO        R9 CONTAINS LENGTH OF HEXZERO
         ICM   R09,B'100',HEXZERO   X'00' INTO TOP BYTE OF R9
30       CLCL  R06,R08              DOES MAP CONTAIN ANY INPUT?
         BE    NOTMODF              ..NO, SO RAISE NOTMODIFIED
         CLC   EIBTRNID,=CL(L'EIBTRNID)'AADD' IS INVOKING T-ID 'ADDS'?
         BE    ADNAMCHK             ..YES, SO GO TO 'AADD' NAME CHECK
UPNAMCHK DS    0H                   UPDATE TRANSACTION HERE
         OC    NAMEI,NAMEI          HAS NAME BEEN CHANGED?
         BZR   RETREG               ..NO, SO DON'T CHECK IT
ADNAMCHK TRT   NAMEO,TAB            ..YES, IS IT ALPHABETIC?
         BM    DATAERR              ..NO, SO RAISE ERROR
         BR    RETREG               ..YES, SO RETURN
FILESTUP DS    0H                   ROUTINE TO SET UP FILE RECORD
31       OC    NAMEI,NAMEI          HAS NAME BEEN ENTERED?
         BZ    ADRTST               ..NO, BRANCH
         MVC   NAME,NAMEI           ..YES, PUT IT IN FILE AREA
ADRTST   OC    ADDRI,ADDRI          HAS ADRESS BEEN ENTERED?
         BZ    PHNTST               ..NO, BRANCH
         MVC   ADDRX,ADDRI          ..YES, PUT IT IN FILE AREA
PHNTST   OC    PHONEI,PHONEI        HAS PHONE BEEN ENTERED?
         BZ    DATTST               ..NO, BRANCH
         MVC   PHONE,PHONEI         ..YES, PUT IT IN FILE AREA
DATTST   OC    DATEI,DATEI          HAS DATE BEEN ENTERED?
         BZ    AMTTST               ..NO, BRANCH
         MVC   DATEX,DATEI          ..YES, PUT IT IN FILE AREA
AMTTST   OC    AMOUNTI,AMOUNTI      HAS AMOUNT BEEN ENTERED?
         BZ    COMTST               ..NO, BRANCH
         MVC   AMOUNT,AMOUNTI       ..YES, PUT IT IN FILE AREA
COMTST   OC    COMMENTI,COMMENTI HAS COMMENT BEEN ENTERED?
         BZ    CONTINUE             ..NO, CONTINUE
         MVC   COMMENT,COMMENTI     ..YES, PUT IT IN FILE AREA
CONTINUE DS    0H                   FILE RECORD IS NOW SET UP
         MVC   LOGREC,FILEREC       MOVE FILE RECORD TO LOG AREA
32       MVC   LDAY,EIBDATE         MOVE DATE TO LOG AREA
         MVC   LTIME,EIBTIME        MOVE TIME TO LOG AREA
         MVC   LTERML,EIBTRMID      MOVE TERMINAL-ID TO LOG AREA
33       EXEC CICS WRITEQ TD QUEUE('LOGA') FROM(LOGA) LENGTH(92)
         CLC   EIBTRNID,=CL(L'EIBTRNID)'AUPD'  UPDATE REQUIRED?
         BNE   ADDWRITE             ..NO, SO BRANCH
34       EXEC CICS REWRITE DATASET('FILEA') FROM(FILEA)
         BR    RETREG               FINISHED, SO RETURN
ADDWRITE DS    0H                   ADD FUNCTION REQUIRED
35       EXEC CICS WRITE DATASET('FILEA') FROM(FILEA)                  *
               RIDFLD(NUMB-FILEDS(COMPTR))
         BR    RETREG               FINISHED, SO RETURN
DATAERR  DS    0H                   GENERAL ROUTINES
36       MVI   NAMEA,DFHBMFSE       PRESERVE CONTENTS OF SCREEN
         MVI   ADDRA,DFHBMFSE       BY SETTING THE MODIFIED DATA TAG
         MVI   PHONEA,DFHBMFSE      ON THE FIELDS ON THE SCREEN.
         MVI   DATEA,DFHBMFSE
         MVI   AMOUNTA,DFHBMFSE
         MVI   COMMENTA,DFHBMFSE
         MVI   MSG3A,DFHBMBRY       BRIGHTEN ERROR MESSAGE
         MVI   MSG1A,DFHBMDAR       DARKEN OPERATOR INSTRUCTION
37       MVC   MSG3O,=CL(L'MSG3O)'DATA ERROR - PLEASE REENTER'
38       EXEC CICS SEND MAP('XDFHAMB') DATAONLY
         CLC   EIBTRNID,=CL(L'EIBTRNID)'AUPD'  UPDATE REQUIRED?
         BE    UPDTERR              ..YES, SO BRANCH
         MVC   COMLEN,=H'7'         ..NO,SET UP COMLEN
         B     CICSCONT             GO RETURN CONTROL TO CICS
```

```
UPDTERR  DS    0H
         MVC   COMLEN,=H'80'        UPDATE, SET UP REQUIRED COMLEN
         B     CICSCONT
NOTMODF  DS    0H                   SCREEN NOT CHANGED
39       MVC   MESSAGES,=CL(L'MESSAGES)'FILE NOT MODIFIED'  MESSAGE
         B     AMNU                 COMPLETE, GO FINISH
DUPREC   DS    0H                   DUPLICATE RECORD
40       MVC   MESSAGES,=CL(L'MESSAGES)'DUPLICATE RECORD'  MESSAGE
         B     AMNU                 COMPLETE, GO FINISH
NOTFOUND DS    0H                   RECORD NOT FOUND
41       MVC   MESSAGES,=CL(L'MESSAGES)'INVALID NUMBER-PLEASE REENTER'
         B     AMNU                 COMPLETE, GO FINISH
ERRORS   DS   0H                    GENERAL ERROR ROUTINE
42       EXEC CICS DUMP DUMPCODE('ERRS')
         MVC   MESSAGES,=CL(L'MESSAGES)'TRANSACTION TERMINATED'
         B     AMNU                 COMPLETE, GO FINISH
HEXZERO  DC    X'00'                CONSTANT FOR COMPARISONS
TAB      DC    256X'FF'             TRANSLATE TABLE
         ORG   TAB+X'40'            BLANK
         DC    X'00'
         ORG   TAB+X'4B'            CHAR '.'
         DC    X'00'
         ORG   TAB+X'C1'            CHARS 'A' - 'I'
         DC    9X'00'
         ORG   TAB+X'D1'            CHARS 'J' - 'R'
         DC    9X'00'
         ORG   TAB+X'E2'            CHARS 'S' - 'Z'
         DC    8X'00'
         ORG
         END
```

Program Notes

1. The possible invoking transaction identifiers are tested.
2. The length of the COMMAREA is tested.
3. The program exits are set up.
4. Map XDFHAMA is received.
5. The account number is saved.
6. If the program was invoked by the transaction-identifier 'AADD' a title and command message are moved to the title area.
7. The record key is moved to the map area and to the COMMAREA.
8. In the case of the AADD transaction, the amount field has the modified data tag and the numeric attribute byte set on so only numeric data can be entered, and if no data is entered, the field contains the original data if it has not been modified when the contents of map XDFHAMB were mapped in.
9. The exit for the record not found condition is set up.
10. The file control READ reads the file record into the file area.
11. If the record is coded as deleted, it is treated as not found.
12. If the program was invoked by the transaction-identifier 'AINQ' a title and command message are moved to the map area.

13. This invocation of the program ends.

14. If the program was invoked by the transid 'AUPD' a title and command message are moved to the map area.

15. The length of the COMMAREA to be returned is set up.

16. The error exits are set up.

17. This command maps in the contents of the screen.

18. The file control READ UPDATE reads the file using the number from the last invocation of this program, which is stored in COMMAREA.

19. The fields from the last invocation are checked against those on the current file record.

20. A message and attribute bytes are moved.

21. The contents of the map XDFHAMB are sent to the terminal.

22. The message 'FILE UPDATED' is moved to MESSAGES.

23. The message 'RECORD ADDED' is moved to MESSAGES.

24. Control is returned to CICS together with the name of the transaction to be invoked when an attention key is pressed at the terminal, and data associated with this transaction is returned in the COMMAREA.

25. The bright attribute is turned on, and MESSAGES is moved to the map area.

26. The screen is erased, and map XDFHAMA is transmitted to the screen.

27. The program ends.

28. The fields from the file area are moved to the map area.

29. The screen is erased, and the map XDFHAMB is sent to the terminal.

30. Any required editing steps should be inserted here. A suitable form of editing should be used here to ensure that valid records are placed on the file.

31. The record to be written to the file is created.

32. The record fields, date, time, and terminal identification are moved to the transient data area.

33. This record is written to a transient data file.

34. The updated record is rewritten to the file.

35. The record is written to the file.

36. The fields from the map have the modified data tag attribute set so that data is still in those fields when the map is received.

37. An error message is moved.

38. The contents of the map XDFHAMB are sent to the screen.

39. If no fields were modified, the message 'FILE NOT MODIFIED' is moved to MESSAGES.

40. If a duplicate record condition exists, the message 'DUPLICATE RECORD' is moved to MESSAGES.

41. If the file record was not found, the message 'INVALID NUMBER - PLEASE REENTER' is moved to MESSAGES.

42. On an error (notes 4, 10, 13, 17, 18, 21, 24, 26, 29, 33, 34, and 38) a dump is taken and the message 'TRANSACTION TERMINATED' is moved to messages.

MAP XDFHAMA (MENU SCREEN)

Map Definition

```
MAPSETA  DFHMSD TYPE=&SYSPARM,MODE=INOUT,CTRL=(FREEKB,FRSET),LANG=ASM, *
               TIOAPFX=YES,EXTATT=MAPONLY,COLOR=BLUE
XDFHAMA  DFHMDI SIZE=(12,40)
         DFHMDF POS=(1,10),LENGTH=21,INITIAL='OPERATOR INSTRUCTIONS'
                  HIGHLIGHT = UNDERLINE
         DFHMDF POS=(3,1),LENGTH=27,INITIAL='OPERATOR INSTR - ENTER AMN*
               U'
         DFHMDF POS=(4,1),LENGTH=38,INITIAL='FILE INQUIRY   - ENTER AIN*
               Q AND NUMBER'
         DFHMDF POS=(5,1),LENGTH=38,INITIAL='FILE BROWSE    - ENTER ABR*
               W AND NUMBER'
         DFHMDF POS=(6,1),LENGTH=38,INITIAL='FILE ADD       - ENTER AAD*
               D AND NUMBER'
         DFHMDF POS=(7,1),LENGTH=38,INITIAL='FILE UPDATE    - ENTER AUP*
               D AND NUMBER'
MSG      DFHMDF POS=(11,1),LENGTH=39,INITIAL='PRESS PA1 TO PRINT--PRESS*
                CLEAR TO EXIT'
         DFHMDF POS=(12,1),LENGTH=18,INITIAL='ENTER TRANSACTION:'
         DFHMDF POS=(12,20),LENGTH=4,ATTRB=IC,COLOR=GREEN,
                  HILIGHT=REVERSE
         DFHMDF POS=(12,25),LENGTH=6,INITIAL='NUMBER'
KEY      DFHMDF POS=(12,32),LENGTH=6,ATTRB=NUM,COLOR=GREEN,
                HILIGHT=REVERSE
         DFHMDF POS=(12,39),LENGTH=1
         DFHMSD TYPE=FINAL
         END
```

Menu Screen Layout

```
+----------------------------------------+
|                                        |
|         +OPERATOR INSTRUCTIONS         |
|+OPERATOR INSTR - ENTER AMNU            |
|+FILE INQUIRY   - ENTER AINQ AND NUMBER |
|+FILE BROWSE    - ENTER ABRW AND NUMBER |
|+FILE ADD       - ENTER AADD AND NUMBER |
|+FILE UPDATE    - ENTER AUPD AND NUMBER |
|                                        |
|                                        |
|+PRESS PA1 TO PRINT--PRESS CLEAR TO EXIT|
|+ENTER TRANSACTION:+XXXX+NUMBER+XXXXXX+ |
+----------------------------------------+
```

Figure 2.2-1. Menu Screen Layout (as seen on 40-character Screen)

MAP XDFHAMB (FILE SCREEN)

Map Definition

```
MAPSETB  DFHMSD TYPE=&SYSPARM,MODE=INOUT,CTRL=(FREEKB,FRSET),LANG=ASM, *
               TIOAPFX=YES,EXTATT=MAPONLY
XDFHAMB  DFHMDI SIZE=(12,40)
TITLE    DFHMDF POS=(1,15),LENGTH=12
         DFHMDF POS=(3,1),LENGTH=8,INITIAL='NUMBER:',COLOR=BLUE
NUMB     DFHMDF POS=(3,10),LENGTH=6
         DFHMDF POS=(3,17),LENGTH=1
         DFHMDF POS=(4,1),LENGTH=8,INITIAL='NAME:   ',COLOR=BLUE
NAME     DFHMDF POS=(4,10),LENGTH=20,ATTRB=(UNPROT,IC)
         DFHMDF POS=(4,31),LENGTH=1
         DFHMDF POS=(5,1),LENGTH=8,INITIAL='ADDRESS:',COLOR=BLUE
ADDR     DFHMDF POS=(5,10),LENGTH=20,ATTRB=UNPROT
         DFHMDF POS=(5,31),LENGTH=1
         DFHMDF POS=(6,1),LENGTH=8,INITIAL='PHONE:   ',COLOR=BLUE
PHONE    DFHMDF POS=(6,10),LENGTH=8,ATTRB=UNPROT
         DFHMDF POS=(6,19),LENGTH=1
         DFHMDF POS=(7,1),LENGTH=8,INITIAL='DATE:    ',COLOR=BLUE
DATE     DFHMDF POS=(7,10),LENGTH=8,ATTRB=UNPROT
         DFHMDF POS=(7,19),LENGTH=1
         DFHMDF POS=(8,1),LENGTH=8,INITIAL='AMOUNT:  ',COLOR=BLUE
AMOUNT   DFHMDF POS=(8,10),LENGTH=8,ATTRB=NUM
         DFHMDF POS=(8,21),LENGTH=1
         DFHMDF POS=(9,1),LENGTH=8,INITIAL='COMMENT:',COLOR=BLUE
COMMENT  DFHMDF POS=(9,10),LENGTH=9,ATTRB=UNPROT
         DFHMDF POS=(9,20),LENGTH=1
MSG1     DFHMDF POS=(11,1),LENGTH=39
MSG3     DFHMDF POS=(12,1),LENGTH=39
         DFHMSD TYPE=FINAL
         END
```

File Screen Layout

```
┌──────────────────────────────────────┐
│                                      │
│          +XXXXXXXXXXXX               │
│                                      │
│+NUMBER: +XXXXXX+                     │
│+NAME:   +XXXXXXXXXXXXXXXXXXXX+       │
│+ADDRESS:+XXXXXXXXXXXXXXXXXXXX+       │
│+PHONE:  +XXXXXXXX+                   │
│+DATE:   +XXXXXXXX+                   │
│+AMOUNT: +XXXXXXXX+                   │
│+COMMENT:+XXXXXXXXX+                  │
│+XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX │
│+XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX │
└──────────────────────────────────────┘
```

Figure 2.2-2. File Screen Layout (as seen on 40-character Screen)

Chapter 2.3. COBOL Programming

CICS/VS acts as an interface between COBOL language application programs and the operating system. When an COBOL language application program is designed to run under CICS/VS, certain commands can be replaced by CICS/VS commands. This substitution is usually mandatory. In particular, applications must always use CICS/VS commands to perform input and output operations.

The first part of this chapter describes the use of CICS/VS commands in COBOL language programming, and lists some programming rules. The second part demonstrates the principles by describing, listing, and analysing a sample program.

Translator Invocation

For details of translator invocation, see Chapter 3.5, "Preparation of Application Programs."

COBOL Translator Options

The translator provides optional facilities, which can be requested by job control statements. Some of the options have default values.

Translator options are specified in the CBL job control statement, and within the XOPTS keyword option. For example:

```
CBL   XOPTS(QUOTE SPACE2)
```

The options can be specified in any order, and can be separated by commas or blanks. If the options are coded in the EXEC job control statement, the XOPTS keyword (and its associated parentheses) is unnecessary. Only options for the translator are permitted.

Note: For compatibility with existing coding, CICS/VS-ELS 1.5 will continue to recognize the job control keyword CICS, which has been replaced by keyword XOPTS.

The CBL statement can also contain options which apply to the following compiler. These options will be ignored by the translator (that is, they will not be checked for validity), but will be copied through into the output data set. For example, a program preceded by:

```
CBL XOPTS (QUOTE),ATTRIBUTES
```

will be passed to the compiler preceded by:

```
CBL ATTRIBUTES
```

Translator Options	Default
CICS	-
DLI	-
DEBUG|NODEBUG	NODEBUG
FE	-
FLAGW|FLAGE|FLAGI	FLAGW
LIST|NOLIST	LIST
NUM|NONUM	NONUM
NOSPIE	-
QUOTE|APOST	APOST
SEQ|NOSEQ	SEQ
SPACE1|SPACE2|SPACE3	SPACE1
XREF|NOXREF	NOXREF

CICS

Specifies that the program contains "EXEC CICS...." commands.

This option is assumed as a default if CBL CICS (.....) is coded, but not if CBL XOPTS (.....) is coded.

DLI

Specifies that the program contains "EXEC DLI..." commands.

DEBUG|NODEBUG

specifies whether or not the execution diagnostic facility (EDF) is to display the line numbers of commands as shown in the source listing. The default is NODEBUG. (EDF is described in Chapter 4.1.)

FE

produces translator informatory messages which print (in hexadecimal notation) the bit pattern corresponding to the first argument of the translated call. This bit pattern has the encoded information that the EXEC interface program uses to determine which function is required and which options are specified. If FE is specified, all diagnostic messages are listed, whatever the FLAG option specifies.

FLAGI|FLAGW|FLAGE
: specifies which diagnostics the translator is required to list: FLAGI specifies diagnostics at all severity levels; FLAGW specifies diagnostics at severity levels W, C, E, and D; and FLAGE specifies diagnostics at severity levels C, E, and D. The default is FLAGW.

LIST|NOLIST
: specifies whether or not the translator is to produce a listing of its COBOL input. The default is LIST.

NOSPIE
: is used to prevent the translator from trapping unrecoverable errors; instead, a dump is produced.

NUM|NONUM
: specifies whether or not the translator is to use the line numbers appearing in columns 1 through 6 of the card as the line number in its diagnostic messages and cross-reference listing. If NUM is not specified, the translator generates its own line numbers.

QUOTE|APOST
: QUOTE indicates to the translator that the double quotation marks (") should be accepted as the character to delineate literals; APOST indicates that the apostrophe (') should be accepted instead. The default is APOST.

SEQ|NOSEQ
: indicates whether or not the translator is required to check the sequence of source statements. If SEQ is specified and a statement is not in sequence it is flagged. The default is SEQ.

SPACE1|SPACE2|SPACE3
: indicates the required type of spacing to be used in the output listing: SPACE1 specifies single spacing; SPACE2 double spacing; and SPACE3 triple spacing. The default is SPACE1.

XREF|NOXREF
: specifies whether or not the translator is required to provide a cross-reference list of all the CICS/VS commands used in its input. The default is NOXREF.

Command Syntax

The format of a CICS/VS command is as follows:

1. The verb EXECUTE or its abbreviation, EXEC
2. The identifier CICS
3. The function
4. A sequence of options. The options may be written in any order.
5. The statement terminator END-EXEC.

The general format is:

```
{EXECUTE|EXEC} CICS function [option]...  END-EXEC.
```

General Rules for COBOL Programming

A CICS/VS COBOL application program contains the four standard divisions a COBOL programmer is familiar with, the identification division, environment division, data division, and procedure division.

IDENTIFICATION DIVISION

The identification division may include any entries specified in COBOL. For simplicity, the examples in this manual show only the program identification entry:

```
IDENTIFICATION DIVISION.
PROGRAM-ID.  program-name.
```

ENVIRONMENT DIVISION

The environment division may include the configuration section, but not the input-output section. CICS/VS provides commands to replace all COBOL input and output verbs. The examples in this manual show only the environment division entry:

```
ENVIRONMENT DIVISION.
```

DATA DIVISION

The data division describes the data areas available to the program. These areas are defined in the working-storage and linkage sections. The file section is not used, as CICS/VS includes commands to perform all file operations.

Storage Area Summary

The table below summarizes the four types of storage areas in which an application program may store data. CWA, TWA, and TCTUA are areas supplied by CICS/VS additional to those normally available to the COBOL programmer.

AREA NAME	AREA TYPE	COBOL SECTION	HOW CREATED	TYPICAL CONTENTS	AREA DURATION
WORKING STORAGE	WORK AREA	WORKING-STORAGE SECTION	COBOL PROGRAMS	CONSTANTS &VARIABLES	PROGRAM DURATION
CWA*	WORK AREA	LINKAGE SECTION	CICS/VS GENERATION	SYSTEM VARIABLES	CICS/VS DURATION
TWA	WORK AREA	LINKAGE SECTION	PCT	VARIABLES	SINGLE TASK
TCTUA**	WORK AREA	LINKAGE SECTION	TCT	VARIABLES	CICS/VS DURATION

*Maximum size: 3584 bytes
**Maximum size: 255 bytes

Working-Storage Section

The working-storage section contains the working storage area of the application program and any data areas required for move-mode input/output operations.

To simplify 3270 programming, CICS/VS provides two sets of constants that should be copied into the working-storage section. The first set, DFHAID, is used to determine which 3270 attention key was pressed to initiate a transaction. The second set, DFHBMSCA, contains some commonly-used 3270 attribute bytes which may be used to modify the attribute bytes on a screen. (Information about these constants is given in Chapter 2.1 of this manual.)

The program may describe other constants in the working-storage section. The following is an example of how this section might be coded.

```
DATA DIVISION.
WORKING-STORAGE SECTION.
01  DFHAID COPY DFHAID.
01  DFHBMSCA COPY DFHBMSCA.
01 ...
```

Linkage Section

The linkage section defines the user overlay areas. User overlay areas may be used to describe the CWA, TWA, TCTUA and any data areas required for locate-mode input/output operations.

The linkage section is divided into two parts, data area pointers and data area descriptions.

Data Area Pointers

Data area pointers are used so that the program may access data in external areas without copying the data. A separate pointer is required for each overlay area referred to by the program. The following is an example of how the pointers are coded in part one of the linkage section:

```
LINKAGE SECTION.
01  POINTERS.
        02 FILLER PICTURE S9(8) COMP.        (See Note 1.)
        02 AREA1-PTR PICTURE S9(8) COMP.     (See Note 2.)
        02 AREA2-PTR PICTURE S9(8) COMP.
```

Notes:

1. FILLER
   Specify exactly as shown.

2. AREA1-PTR
   Specify to provide a pointer to a storage area provided by CICS/VS.

Data Area Descriptions

The second part of the linkage section contains the record description entries for the areas accessed by means of pointers.

For each record description entry in part two of the linkage section, there must be a corresponding data area pointer in part one of the linkage section. The correspondence between pointers and record descriptions is determined by the sequence of the pointers and record descriptions. The first pointer in part one of the linkage section is associated with the first record description in part two of this section, the second pointer with the second record description, and so on.

When coding the linkage section, the programmer must be careful of the sequence of pointers and record descriptions. The following is an example of how the data area record descriptions are coded in part two of the linkage section and how the pointers relate to data area descriptions.

```
LINKAGE SECTION.
01  POINTERS.
        02 FILLER PICTURE S9(8) COMP.
        02 AREA1-PTR PICTURE S9(8) COMP.
        02 AREA2-PTR PICTURE S9(8) COMP.

01  AREA1.
        02 COUNT...
        02 NAME-FIELD...

01  AREA2.
        02 FILE-KEY...
        02 INTERMEDIATE-RESULTS...
```

AREA1-PTR addresses AREA1. AREA2-PTR addresses AREA2. The pointer values are set up by means of a CICS/VS command. The fields of the corresponding data area can then be referred to simply by coding their name.

The sample programs in this chapter illustrate coding for the identification, environment, and data divisions.

PROCEDURE DIVISION

This section describes the use of the procedure division and introduces the CICS/VS commands used in the procedure division to perform CICS/VS services.

The procedure division of a COBOL program must not use the following COBOL verbs or features:

```
READ, WRITE, OPEN, CLOSE
ACCEPT, DISPLAY (except with the system console)
TRACE, EXHIBIT
FLOW, STATE
STXIT, SYMDMP
STRING, UNSTRING
SORT, REPORT WRITER, SEGMENTATION
STOP RUN
```

If the default for the COBOL compiler is OPT then NOOPT must be specified in the CBL card indicating no optimizer compilation.

CICS/VS provides special commands to perform input and output operations to terminals and files.

Commands may be written anywhere in a COBOL program that a COBOL executable statement may be written.

Separate COBOL routines cannot be link-edited together.

Description of UPDATE Sample Program

The update sample program combines the facilities of file update, file add and file inquiry.

The update program maps in the account number and unless the invoking transaction-identifier is 'CADD', reads the file record. The required fields from the file area, and a title depending on the invoking transaction-identifier, are moved to the map area. In the case of the file add function being required, the number entered onto map XDFHCMA, and a title, are moved to the map area of XDFHCMB. Then XDFHCMB, containing the record fields, is displayed at the terminal. If the function of this transaction is file inquiry, the program ends here.

The update program then reads and maps in the record to be added or updated, and edits the fields. The sample program only suggests the type of editing that might be done. The user should insert editing steps needed to ensure valid changes to the file. Those fields which have been changed are moved to the file area. Log information is moved to the transient data area. The file record is then either added or

updated, depending on the function required of the program. Either the message 'FILE UPDATED' or 'RECORD ADDED' is inserted in XDFHCMA and the map is transmitted to the terminal.

Note: This program demonstrates a pseudo-conversational programming technique, where control is returned to CICS/VS along with a transaction identifier whenever a response is requested from the operator. Associated with each return of control to CICS/VS is a storage area containing details associated with the previous invocation of this transaction.

Listing of UPDATE Sample Program and Maps

```
      IDENTIFICATION DIVISION.
      PROGRAM-ID. UPDATE.
      ENVIRONMENT DIVISION.
      DATA DIVISION.
      WORKING-STORAGE SECTION.
      77  MESSAGES PICTURE X(39).
      77  NAMET PIC X(20).
      77  KEYNUM PICTURE 9(6).
      77  COMLEN PICTURE S9(4) COMP.
      01  XDFHCMAI COPY XDFHCMA.
      01  XDFHCMBI COPY XDFHCMB.
      01  FILEA COPY FILEA.
      01  LOGA COPY LOGA.
      01  DFHBMSCA COPY DFHBMSCA.
      01  COMMAREA COPY FILEA.
      LINKAGE SECTION.
      01  DFHCOMMAREA COPY FILEA.
      PROCEDURE DIVISION.
1        IF EIBTRNID NOT = 'CINQ'
           OR EIBTRNID NOT = 'CADD'
           OR EIBTRNID NOT = 'CUPD' THEN GO TO ERRORS.
2        IF EIBCALEN NOT = 0 THEN
3           MOVE DFHCOMMAREA TO COMMAREA GO TO READ-INPUT.
4         EXEC CICS HANDLE CONDITION MAPFAIL(MENU)
                                     ERROR(ERRORS) END-EXEC.
5         EXEC CICS RECEIVE MAP('XDFHCMA') END-EXEC.
          IF KEYI = LOW-VALUES THEN GO TO NOTFOUND.
6         MOVE KEYI TO KEYNUM
          MOVE LOW-VALUES TO XDFHCMBO.
7         IF EIBTRNID = 'CADD' THEN
             MOVE 'FILE ADD' TO TITLEO
             MOVE 'ENTER DATA AND PRESS ENTER KEY' TO MSG3O
8            MOVE KEYI TO NUMB IN COMMAREA, NUMBO
9            MOVE 'J' TO AMOUNTA
             MOVE '£0000.00' TO AMOUNTO
             MOVE 7 TO COMLEN GO TO MAP-SEND.
10        EXEC CICS HANDLE CONDITION NOTFND(NOTFOUND) END-EXEC.
11        EXEC CICS READ DATASET('FILEA') INTO(FILEA) RIDFLD(KEYNUM)
                                                      END-EXEC
12        IF STAT IN FILEA = HIGH-VALUE THEN GO TO NOTFOUND.
          IF EIBTRNID = 'CINQ' THEN
13           MOVE 'FILE INQUIRY' TO TITLEO
             MOVE 'PRESS ENTER TO CONTINUE' TO MSG3O
             PERFORM MAP-BUILD THRU MAP-SEND
14           EXEC CICS RETURN TRANSID('CMNU') END-EXEC.
          IF EIBTRNID = 'CUPD' THEN
15           MOVE 'FILE UPDATE' TO TITLEO
             MOVE 'CHANGE FIELDS AND PRESS ENTER' TO MSG3O
16           MOVE FILEREC IN FILEA TO FILEREC IN COMMAREA
             MOVE 80 TO COMLEN.
      MAP-BUILD.
          MOVE NUMB IN FILEA TO NUMBO
          MOVE NAME IN FILEA TO NAMEO
17        MOVE ADDRX IN FILEA TO ADDRO
          MOVE PHONE IN FILEA TO PHONEO
          MOVE DATEX IN FILEA TO DATEO
          MOVE AMOUNT IN FILEA TO AMOUNTO
          MOVE COMMENT IN FILEA TO COMMENTO.
      MAP-SEND.
18        EXEC CICS SEND MAP('XDFHCMB') ERASE END-EXEC.
      FIN.
```

```
           GO TO CICS-CONTROL.
       READ-INPUT.
    19     EXEC CICS HANDLE CONDITION MAPFAIL(NOTMODF) NOTFND(NOTFOUND)
                      ERROR(ERRORS) DUPREC(DUPREC) END-EXEC.
    20     EXEC CICS RECEIVE MAP('XDFHCMB') END-EXEC.
           IF EIBTRNID = 'CUPD' THEN
    21        EXEC CICS READ UPDATE DATASET('FILEA') INTO(FILEA)
                         RIDFLD(NUMB IN COMMAREA) END-EXEC
    22        IF FILEREC IN FILEA NOT = FILEREC IN COMMAREA THEN
                 MOVE 'FILE ALREADY UPDATED - REENTER' TO MSG1O
    23           MOVE DFHBMBRY TO MSG1A
                 MOVE DFHBMDAR TO MSG3A
                 PERFORM MAP-BUILD
    24           EXEC CICS SEND MAP('XDFHCMB') END-EXEC
                 MOVE 80 TO COMLEN
                 MOVE FILEREC IN FILEA TO FILEREC IN COMMAREA
                 GO TO CICS-CONTROL
              ELSE
                 MOVE 'U' TO STAT IN FILEA
                 PERFORM CHECK THRU FILE-WRITE
    25           MOVE 'FILE UPDATED' TO MESSAGES GO TO MENU.
           IF EIBTRNID = 'CADD' THEN
                 MOVE LOW-VALUES TO FILEREC IN FILEA
                 MOVE 'A' TO STAT IN FILEA
                 PERFORM CHECK THRU FILE-WRITE
    26           MOVE 'RECORD ADDED' TO MESSAGES GO TO MENU.
       CHECK.
           IF NAMEI = LOW-VALUES AND
              ADDRI = LOW-VALUES AND
    27        PHONEI = LOW-VALUES AND
              DATEI = LOW-VALUES AND
              AMOUNTI = LOW-VALUES AND
              COMMENTI = LOW-VALUES GO TO NOTMODF.
           MOVE NAMEI TO NAMET
           TRANSFORM NAMET CHARACTERS FROM '.' TO ' '.
           IF EIBTRNID = 'CADD' THEN
              IF NAMET NOT ALPHABETIC THEN GO TO DATA-ERROR.
           IF EIBTRNID = 'CUPD' THEN
           IF NAMEI NOT = LOW-VALUES
           AND NAMET NOT ALPHABETIC THEN GO TO DATA-ERROR.
       FILE-WRITE.
           IF EIBTRNID = 'CADD' THEN MOVE NUMB IN COMMAREA TO
                                         NUMB IN FILEA.
           IF NAMEI NOT = LOW-VALUE MOVE NAMEI TO NAME IN FILEA.
    28     IF ADDRI NOT = LOW-VALUE MOVE ADDRI TO ADDRX IN FILEA.
           IF PHONEI NOT = LOW-VALUE MOVE PHONEI TO PHONE IN FILEA.
           IF DATEI NOT = LOW-VALUE MOVE DATEI TO DATEX IN FILEA.
           IF AMOUNTI NOT = LOW-VALUE MOVE AMOUNTI TO AMOUNT IN FILEA.
           IF COMMENTI NOT = LOW-VALUE THEN
              MOVE COMMENTI TO COMMENT IN FILEA.
           MOVE FILEREC IN FILEA TO LOGREC.
           MOVE EIBDATE TO LDAY
    29     MOVE EIBTIME TO LTIME
           MOVE EIBTRMID TO LTERML
    30     EXEC CICS WRITEQ QUEUE('LOGA') FROM(LOGA) LENGTH(92)
                                                     END-EXEC.
           IF EIBTRNID = 'CUPD' THEN
    31        EXEC CICS REWRITE DATASET('FILEA') FROM(FILEA) END-EXEC
           ELSE
    32        EXEC CICS WRITE DATASET('FILEA') FROM(FILEA)
                                               RIDFLD(NUMB IN COMMAREA)
                                               END-EXEC.
       DATA-ERROR.
           MOVE DFHBMBRY TO MSG3A
    33     MOVE 'DATA ERROR - CORRECT AND PRESS ENTER' TO MSG3O
```

```
34          MOVE DFHBMFSE TO NAMEA, ADDRA, PHONEA, DATEA, AMOUNTA,
                COMMENTA.
35          EXEC CICS SEND MAP('XDFHCMB') DATAONLY END-EXEC.
            IF EIBTRNID = 'CADD' THEN MOVE 7 TO COMLEN
            ELSE MOVE 80 TO COMLEN.
        CICS-CONTROL.
36          EXEC CICS RETURN TRANSID(EIBTRNID) COMMAREA(COMMAREA)
                                               LENGTH(COMLEN) END-EXEC.
        NOTMODF.
37          MOVE 'FILE NOT MODIFIED' TO MESSAGES
            GO TO MENU.
        DUPREC.
38          MOVE 'DUPLICATE RECORD' TO MESSAGES
            GO TO MENU.
        NOTFOUND.
39          MOVE 'INVALID NUMBER - PLEASE REENTER' TO MESSAGES
            GO TO MENU.
        ERRORS.
40          EXEC CICS DUMP DUMPCODE('ERRS') END-EXEC
            MOVE 'TRANSACTION TERMINATED' TO MESSAGES.
        MENU.
            MOVE LOW-VALUE TO XDFHCMAO
41          MOVE DFHBMBRY TO MSGA
            MOVE MESSAGES TO MSGO
42          EXEC CICS SEND MAP('XDFHCMA') ERASE END-EXEC
43          EXEC CICS RETURN END-EXEC.
            GOBACK.
```

Program Notes

1. The possible invoking transaction-identifier's are tested.
2. The length of the COMMAREA is tested.
3. If it has a length, the COMMAREA returned is moved to working storage in the program.
4. The program exits are set up.
5. Map XDFHCMA is received.
6. The account number is saved.
7. If the program was invoked by the transaction identifier 'CADD', a title and command message are moved to the title area.
8. The record key is moved to the COMMAREA and to the map area.
9. In the case of the CADD transaction, the amount field has the modified data tag and the numeric attribute byte set on so that only numeric data can be entered, and if no data is entered, the field contains the original data if it has not been modified when the contents of map XDFHCMB are mapped in.
10. The error exit is set up for the record not found condition.
11. The file control READ reads the file record into the file area.
12. If the record is coded as deleted, it is treated as not found.
13. If the program was invoked by the transaction-identifier 'CINQ' a title and command message are moved to the map area.

14. This invocation of the program ends.

15. If the program was invoked by the transaction identifier 'CUPD' a title and command message are moved to the map area.

16. The file record is moved to the COMMAREA and the length of the COMMAREA to be returned is set up.

17. The fields from the file area are moved to the map area.

18. The screen is erased and the map XDFHCMB is sent to the terminal.

19. The program exits are set up.

20. This command maps in the contents of the screen.

21. The file control READ UPDATE reads the file using the number from the last invocation of this transaction of this program, which is stored in the COMMAREA.

22. The fields from the last invocation are checked against those on the current file record.

23. A message and attribute bytes are moved.

24. Map XDFHCMB is sent to the terminal.

25. The message 'FILE UPDATED' is moved to MESSAGES.

26. The message 'RECORD ADDED' is moved to MESSAGES.

27. Any required editing steps should be inserted here. A suitable form of editing should be used here to ensure that valid records are placed on the file.

28. The record to be written to the file is created.

29. The record fields, date, time, and terminal identification are moved to the transient data area.

30. This record is written to a transient data file.

31. The updated record is rewritten to the file.

32. The record to be added is written to the file.

33. An error message is moved.

34. Fields on map XDFHCMB which are to be sent back to the screen have the modified data tag set on so they will still contain data if the contents are not altered, when the screen is mapped in.

35. The contents of the map XDFHCMB are sent to the screen.

36. Control is returned to CICS along with the name of the transaction to be invoked when an attention key is pressed at the terminal, and data associated with this transaction is returned in the COMMAREA.

37. If no fields were modified, the message 'FILE NOT MODIFIED' is moved to MESSAGES.

38. If a duplicate record condition exists, the message 'DUPLICATE RECORD' is moved to MESSAGES.

39. If the file record was not found, the message 'INVALID NUMBER - PLEASE REENTER' is moved to MESSAGES.

40. On an error (notes 5, 11, 18, 20, 21, 24, 30, 31, 32, 35, and 42) a dump is taken, and the message 'TRANSACTION TERMINATED' is moved to MESSAGES.

41. The bright attribute is turned on, and MESSAGES is moved to the map area.

42. The screen is erased, and map XDFHCMA is transmitted to the screen.

43. The program ends.

MAP XDFHCMA (MENU SCREEN)

Map Definition

```
MAPSET   DFHMSD TYPE=&SYSPARM,MODE=INOUT,CTRL=(FREEKB,FRSET),          *
               LANG=COBOL,TIOAPFX=YES,EXTATT=MAPONLY,COLOR=BLUE
XDFHCMA  DFHMDI SIZE=(12,40)
         DFHMDF POS=(1,10),LENGTH=21,INITIAL='OPERATOR INSTRUCTIONS',  *
               HILIGHT=UNDERLINE
         DFHMDF POS=(3,1),LENGTH=27,INITIAL='OPERATOR INSTR - ENTER CMN*
               U'
         DFHMDF POS=(4,1),LENGTH=38,INITIAL='FILE INQUIRY   - ENTER CIN*
               Q AND NUMBER'
         DFHMDF POS=(5,1),LENGTH=38,INITIAL='FILE BROWSE    - ENTER CBR*
               W AND NUMBER'
         DFHMDF POS=(6,1),LENGTH=38,INITIAL='FILE ADD       - ENTER CAD*
               D AND NUMBER'
         DFHMDF POS=(7,1),LENGTH=38,INITIAL='FILE UPDATE    - ENTER CUP*
               D AND NUMBER'
MSG      DFHMDF POS=(11,1),LENGTH=39,INITIAL='PRESS PA1 TO PRINT--PRESS*
                CLEAR TO EXIT'
         DFHMDF POS=(12,1),LENGTH=18,INITIAL='ENTER TRANSACTION:'
         DFHMDF POS=(12,20),LENGTH=4,ATTRB=IC,COLOR=GREEN,HILIGHT=REVER*
               SE
         DFHMDF POS=(12,25),LENGTH=6,INITIAL='NUMBER'
KEY      DFHMDF POS=(12,32),LENGTH=6,ATTRB=NUM,COLOR=GREEN,HILIGHT=REVE*
               RSE
         DFHMDF POS=(12,39),LENGTH=1
         DFHMSD TYPE=FINAL
         END
```

Menu Screen Layout

```
+-----------------------------------------+
|                                         |
|         +OPERATOR INSTRUCTIONS          |
|+OPERATOR INSTR - ENTER CMNU             |
|+FILE INQUIRY   - ENTER CINQ AND NUMBER  |
|+FILE BROWSE    - ENTER CBRW AND NUMBER  |
|+FILE ADD       - ENTER CADD AND NUMBER  |
|+FILE UPDATE    - ENTER CUPD AND NUMBER  |
|                                         |
|                                         |
|+PRESS PA1 TO PRINT--PRESS CLEAR TO EXIT |
|+ENTER TRANSACTION:+XXXX+NUMBER+XXXXXX+  |
+-----------------------------------------+
```

Figure 2.3-1. Menu Screen Layout (as seen on 40-character Screen)

MAP XDFHCMB (FILE SCREEN)

Map Definition

```
MAPSET   DFHMSD TYPE=&SYSPARM,MODE=INOUT,CTRL=(FREEKB,FRSET),          *
               LANG=COBOL,TIOAPFX=YES,EXTATT=MAPONLY
XDFHCMB  DFHMDI SIZE=(12,40)
TITLE    DFHMDF POS=(1,15),LENGTH=12
         DFHMDF POS=(3,1),LENGTH=8,INITIAL='NUMBER:',COLOR=BLUE
NUMB     DFHMDF POS=(3,10),LENGTH=6
         DFHMDF POS=(3,17),LENGTH=1
         DFHMDF POS=(4,1),LENGTH=8,INITIAL='NAME:  ',COLOR=BLUE
NAME     DFHMDF POS=(4,10),LENGTH=20,ATTRB=(UNPROT,IC)
         DFHMDF POS=(4,31),LENGTH=1
         DFHMDF POS=(5,1),LENGTH=8,INITIAL='ADDRESS:',COLOR=BLUE
ADDR     DFHMDF POS=(5,10),LENGTH=20,ATTRB=UNPROT
         DFHMDF POS=(5,31),LENGTH=1
         DFHMDF POS=(6,1),LENGTH=8,INITIAL='PHONE:  ',COLOR=BLUE
PHONE    DFHMDF POS=(6,10),LENGTH=8,ATTRB=UNPROT
         DFHMDF POS=(6,19),LENGTH=1
         DFHMDF POS=(7,1),LENGTH=8,INITIAL='DATE:  ',COLOR=BLUE
DATE     DFHMDF POS=(7,10),LENGTH=8,ATTRB=UNPROT
         DFHMDF POS=(7,19),LENGTH=1
         DFHMDF POS=(8,1),LENGTH=8,INITIAL='AMOUNT:  ',COLOR=BLUE
AMOUNT   DFHMDF POS=(8,10),LENGTH=8,ATTRB=NUM
         DFHMDF POS=(8,19),LENGTH=1
         DFHMDF POS=(9,1),LENGTH=8,INITIAL='COMMENT:',COLOR=BLUE
COMMENT  DFHMDF POS=(9,10),LENGTH=9,ATTRB=UNPROT
         DFHMDF POS=(9,20),LENGTH=1
MSG1     DFHMDF POS=(11,1),LENGTH=39
MSG3     DFHMDF POS=(12,1),LENGTH=39
         DFHMSD TYPE=FINAL
         END
```

File Screen Layout

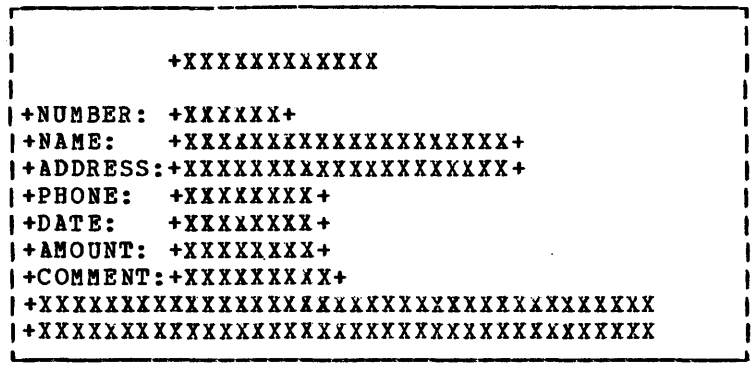


Figure 2.3-2. File Screen Layout (as seen on 40-character Screen)

Chapter 2.4. PL/I Programming

CICS/VS acts as an interface between PL/I language application programs and the operating system. When a PL/I language application program is designed to run under CICS/VS, certain commands can be replaced by CICS/VS commands. This substitution is usually mandatory. In particular, applications must always use CICS/VS commands to perform input and output operations.

The first part of this chapter describes the use of CICS/VS commands in PL/I language programming, and lists some programming rules. The second part demonstrates the principles by describing, listing, and analysing a sample program.

Translator Invocation

For details of translator invocation, see Chapter 3.5, "Preparation of Application Programs."

Translator Options

The translator provides optional facilities, which can be requested by job control statements. Some of the options have default values.

Translator options are specified in the *PROCESS job control statement, and within the XOPTS keyword option. For example:

```
*PROCESS  XOPTS(FLAG(W) DEBUG)
```

The options can be specified in any order, and can be separated by commas or blanks. If the options are coded in the EXEC job control statement, the XOPTS keyword (and its associated parentheses) is unnecessary. Only options for the translator are permitted.

Note: For compatibility with existing coding, CICS/VS-ELS 1.5 will continue to recognize the job control keyword "CICS", which has been replaced by keyword "XOPTS".

The *PROCESS statement can also contain options which apply to the following compiler. These options will be ignored by the translator (that is, they will not be checked for validity), but will be copied through into the output data set. For example, a program preceded by:

```
*PROCESS XOPTS (SOURCE),ATTRIBUTES
```

will be passed to the compiler preceded by:

```
*PROCESS ATTRIBUTES
```

Translator Options	Abbreviations	Default
CICS	-	-
DLI	-	-
DEBUG|NODEBUG	-	NODEBUG
FE		-
FLAG[(I|W|E|S)]	F[(I|W|E|S)]	FLAG (I)
LINECOUNT (n)	LC (n)	LINECOUNT (55)
MARGINS (m,n[,c])	MAR (m,n[,c])	MARGINS (2,72,0)
NOSPIE	-	-
OPMARGINS (m,n[,c])	OM (m,n[,c])	OPMARGINS (2,72,0)
OPSEQUENCE (m,n) | NOOPSEQUENCE	OS (m,n) |NOS	OS (73,80)
OPTIONS|NOOPTIONS	OP|NOP	OPTIONS
SEQUENCE (m,n) | NOSEQUENCE	SEQ (m,n) |NSEQ	SEQUENCE (73,80)
SOURCE|NOSOURCE	S|NS	SOURCE
XREF|NOXREF	X|NX	NOXREF

CICS

Specifies that the program contains "EXEC CICS..." commands.

This option is assumed as a default if * PROCESS CICS (.....) is coded, but not if * PROCESS XOPTS (.....) is coded.

DLI

Specifies that the program contains "EXEC DLI..." commands.

DEBUG|NODEBUG

specifies whether the execution diagnostic facility (EDF) is to display the line numbers of commands as shown in the source listing. (EDF is described in Chapter 4.1.)

FE

produces translator informatory messages, which print (in hexadecimal notation) the bit pattern corresponding to the first argument of the translated call. This bit pattern forms a code that the EXEC interface program uses to determine which function is required and which options are specified. If FE is specified, all diagnostic messages are listed, whatever the FLAG option specifies.

FLAG[(I|W|E|S)]
specifies the minimum severity of error that requires a message to be listed.

FLAG(I)	List all messages
FLAG or FLAG(W)	List all except informatory messages
FLAG(E)	List all except warning and informatory messages
FLAG(S)	List only severe error and unrecoverable error messages

LINECOUNT(n)
specifies the number of lines to be included in each page of translator listing, including heading and blank lines. The value of n must be an integer in the range 1 to 32767; if n is less than 5, only the heading and one line of listing will be included on each page.

MARGINS(m,n[,c])
specifies the extent of the part of each input line or record that contains PL/I statements. The translator does not process data that is outside these limits (but it does include it in the source listings).

The option can also specify the position of an American National Standard (ANS) printer control character to format the listing produced if the SOURCE option applies; otherwise the input records will be listed without any intervening blank lines.

"m" is the column number of the left-hand margin.

"n" is the column number of the right-hand margin. It must be greater than "m".

"c" is the column number of the ANS printer control character. It must be outside the values specified for "m" and "n". A zero value for"c" specifies that there is no printer control character.

Only the following printer control characters can be used:

(blank)	Skip one line before printing
-	Skip two lines before printing
+	No skip before printing
1	Start new page

NOSPIE
is used to prevent the translator trapping unrecoverable errors; instead, a dump is produced.

OPMARGINS(m,n[,c])
specifies the translator output margins, that is, the margins of the input to the PL/I compiler. Normally these will be the same as the input margins. For the meaning of "m", "n" and "c", see MARGINS.

OPSEQUENCE(m,n)|NOOPSEQUENCE
specifies the position of the sequence field in the output records. For the meaning of "m" and "n", see SEQUENCE.

OPTIONS|NOOPTIONS
specifies whether the translator is to include in the listing a list of all the translator options used during this translation.

SEQUENCE(m,n)|NOSEQUENCE
specifies the extent of the part of each input line or record that contains a sequence number. This number is included in the source listing and used in the error message and cross-reference listings. No attempt is made to sort the input lines or records into sequence. If NOSEQUENCE is specified, the translator creates and prints in the source listing its own sequence numbers; this is necessary so that the error messages and cross-reference listings can refer to a particular line in the source listing.

"m" specifies the column number of the left-hand margin

"n" specifies the column number of the right-hand margin.

The extent specified must not overlap with the source program (as specified in the MARGINS option).

The maximum length for the sequence field is 8 characters.

SOURCE|NOSOURCE
specifies whether the translator is to include in the listing a listing of the source program.

XREF|NOXREF
specifies whether the translator is to include in the listing a list of all the CICS/VS commands used in the program together with the sequence numbers of the lines in which they are used.

Command Syntax

The format of a CICS/VS command is as follows:

1. The keyword EXECUTE or its abbreviation, EXEC
2. The identifier CICS
3. The function keyword
4. Possibly a sequence of options, can be written in any order.

5. The statement terminator ';'

The general format is:

```
{EXECUTE|EXEC} CICS function [option]...  ;
```

General Rules for PL/I Programming

Restrictions

The following PL/I features cannot be used in an application program designed for use with CICS/VS (refer to the DOS PL/I Optimizing Compiler Programmer's Guide for more information):

1. The multitasking built-in functions: COMPLETION, PRIORITY, STATUS.
2. The multitasking options: EVENT, PRIORITY, TASK.
3. The PL/I statements: READ, WRITE, GET, PUT (a limited form is permitted), OPEN, CLOSE, DISPLAY, DELAY, REWRITE, LOCATE, DELETE, UNLOCK, STOP, HALT, EXIT, FETCH, and RELEASE. (CICS/VS commands are provided for the storage and retrieval of data, and for communication with terminals.)
4. PL/I sort/merge.
5. Static storage (except for read-only data).

OPTIONS(MAIN) Specification

If OPTIONS(MAIN) is specified in an application program, that program can be the first program of a transaction, or control can be passed to it by means of a LINK or XCTL command.

If OPTIONS(MAIN) is not specified, the program cannot be first in a transaction, nor have control passed to it by a LINK or XCTL command, but it can be link-edited to a main program.

The definition of the EXEC interface block (EIB) is generated only in main programs. If fields in the EIB are referred to in an external procedure for which OPTIONS(MAIN) is not specified, either the address of the EIB, or the necessary fields themselves, must be passed to the external procedure as a parameter to the CALL statement that invokes the external procedure.

Program Segments

Segments of programs can be translated by the command language translator, stored in their translated form, and later included in the program to be compiled.

Description of UPDATE Sample Program

The update sample program combines the facilities of file update, file add, and file inquiry.

The update program maps in the account number and reads the file record. The required fields from the file area, and a title depending on the invoking transaction identifier, are moved to the map area. In the case of the file add function being required, the number entered onto map XDFHPMA and a title are moved to the map area of XDFHPMB. Then XDFHPMB containing the record fields, is displayed at the terminal. If the function of this transaction is file inquiry, the program ends here.

The update program then reads and maps in the record to be added or updated, and edits the fields. The sample program only suggests the type of editing that might be done. The user should insert editing steps needed to ensure valid changes to the file. Those fields which have been changed are moved to the file area. Log information is moved to the transient data area. The file record is then either added or updated, depending on the function required of the program. Either the message 'FILE UPDATED' or 'RECORD ADDED' is inserted in XDFHPMA and the map is transmitted to the terminal.

Note: This program demonstrates a pseudo-conversational programming technique, where control is returned to CICS/VS along with a transaction identifier whenever a response is requested from the operator. Associated with each return of control to CICS/VS is a storage area containing details associated with the previous invocation of this transaction.

List of UPDATE Sample Program and Maps

```
PALL:    PROC(COMPOINT)  OPTIONS(MAIN);
         DCL MESSAGES CHAR(39);
         DCL COMLEN FIXED BIN(15);
         DCL KEYNUM PICTURE '(6)9';
         %INCLUDE XDFHPMA;
         %INCLUDE XDFHPMB;
         %INCLUDE FILEA;
         %INCLUDE LOGA;
         %INCLUDE DFHBMSCA;
         DCL CHSTR CHAR(256) BASED;
         DCL COMPOINT PTR;
         DCL COMMAREA LIKE FILEA  BASED(COMPOINT);
  1      IF EIBCALEN¬=0 THEN GO TO READ_INPUT;
  2      EXEC CICS HANDLE CONDITION ERROR(ERRORS) MAPFAIL(PMNU);
         ALLOCATE COMMAREA;
  3      EXEC CICS RECEIVE MAP('XDFHPMA');
         IF KEYL=0 THEN GO TO NOTFOUND;
  4      KEYNUM=KEYI;
         SUBSTR(ADDR(XDFHPMBO)->CHSTR,1,STG(XDFHPMBO))
                 =LOW(STG(XDFHPMBO));
  5      IF EIBTRNID='PADD' THEN
            DO;
              TITLEO='FILE ADD';
              MSG3O='ENTER DATA AND PRESS ENTER KEY';
  6           NUMBO,COMMAREA.NUMB=KEYI;
  7           AMOUNTA='J';
              AMOUNTO='¬0000.00';
              COMLEN=7;
              CALL MAP_SEND;
              GO TO CICS_CONTROL;
            END;
          ELSE
          IF EIBTRNID='PINQ'
           | EIBTRNID='PUPD' THEN
            DO;
  8           EXEC CICS HANDLE CONDITION NOTFND(NOTFOUND);
  9           EXEC CICS READ DATASET('FILEA') INTO(FILEA)
                                                RIDFLD(KEYNUM);
  10          IF FILEA.STAT=HIGH(1) THEN GO TO NOTFOUND;
              IF EIBTRNID='PINQ' THEN
                 DO;
  11               TITLEO='FILE INQUIRY';
                   MSG3O='PRESS ENTER TO CONTINUE';
                   CALL MAP_BUILD;
                   CALL MAP_SEND;
  12               EXEC CICS RETURN TRANSID('PMNU');
                 END;
              ELSE
                 DO;
  13               TITLEO='FILE UPDATE';
                   MSG3O='CHANGE FIELDS AND PRESS ENTER';
  14               COMMAREA.FILEREC=FILEA.FILEREC;
                   CALL MAP_BUILD;
                   CALL MAP_SEND;
                   COMLEN=80;
                   GO TO CICS_CONTROL;
                 END;
            END;
         ELSE
            GO TO ERRORS;
MAP_BUILD:   PROC;
         NUMBO=FILEA.NUMB;
```

```
        NAMEO=FILEA.NAME;
        ADDRO=FILEA.ADDRX;
  15    PHONEO=FILEA.PHONE;
        DATEO=FILEA.DATEX;
        AMOUNTO=FILEA.AMOUNT;
        COMMENTO=FILEA.COMMENT;
        RETURN;
END;
MAP_SEND:    PROC;
  16    EXEC CICS SEND MAP('XDFHPMB') ERASE;
        RETURN;
END;
READ_INPUT:
  17    EXEC CICS HANDLE CONDITION MAPFAIL(NOTMODF) DUPREC(DUPREC)
                                   ERROR(ERRORS) NOTFND(NOTFOUND);
  18    EXEC CICS RECEIVE MAP('XDFHPMB');
        IF EIBTRNID='PUPD' THEN
           DO;
  19         EXEC CICS READ UPDATE DATASET('FILEA') INTO(FILEA)
                                                RIDFLD(COMMAREA.NUMB);
  20         IF STRING(FILEA.FILEREC)¬=STRING(COMMAREA.FILEREC) THEN
                DO;
                  MSG1O='FILE ALREADY UPDATED - REENTER';
                  MSG1A=DFHBMBRY;
  21              MSG3A=DFHBMDAR;
                  CALL MAP_BUILD;
  22              EXEC CICS SEND MAP('XDFHPMB') DATAONLY;
                  COMMAREA.FILEREC=FILEA.FILEREC;
                  COMLEN=80;
                  GO TO CICS_CONTROL;
                END;
              ELSE
                 DO;
                   FILEA.STAT='U';
  23               MESSAGES='FILE UPDATED';
                 END;
           END;
        ELSE
         IF EIBTRNID='PADD' THEN
             DO;
               FILEA.STAT='A';
  24           MESSAGES='RECORD ADDED';
             END;
        ELSE
           GO TO ERRORS;
        IF NAMEL=0 &
           ADDRL=0 &
  25       PHONEL=0 &
           DATEL=0 &
           AMOUNTL=0 &
           COMMENTL=0 THEN
           GO TO NOTMODF;
        IF EIBTRNID='PADD' THEN
           IF VERIFY(NAMEI,'ABCDEFGHIJKLMNOPQRSTUVWXYZ .')¬=0 THEN
              GO TO DATA_ERROR;
        IF EIBTRNID='PUPD' THEN IF NAMEL¬=0 THEN
           IF VERIFY(NAMEI,'ABCDEFGHIJKLMNOPQRSTUVWXYZ .')¬=0 then
              GO TO DATA_ERROR;
        IF EIBTRNID='PADD' THEN
           FILEA.NUMB=COMMAREA.NUMB;
        IF NAMEL¬=0 THEN FILEA.NAME=NAMEI;
        IF ADDRL¬=0 THEN FILEA.ADDRX=ADDRI;
  26    IF PHONEL¬=0 THEN FILEA.PHONE=PHONEI;
        IF DATEL¬=0 THEN FILEA.DATEX=DATEI;
        IF AMOUNTL¬=0 THEN FILEA.AMOUNT=AMOUNTI;
```

```
        IF COMMENTL¬=0 THEN FILEA.COMMENT=COMMENTI;
        LOGREC=FILEA.FILEREC;
        LDAY=EIBDATE;
  27    LTIME=EIBTIME;
        LTERML=EIBTRMID;
  28    EXEC CICS WRITEQ TD QUEUE('LOGA') FROM(LOGA) LENGTH(92);
        IF EIBTRNID='PUPD' THEN
  29       EXEC CICS REWRITE DATASET('FILEA') FROM(FILEA);
        ELSE
  30       EXEC CICS WRITE DATASET('FILEA') FROM(FILEA)
                                              RIDFLD(COMMAREA.NUMB);
        GO TO PMNU;
DATA_ERROR:
        MSG3A=DFHBMBRY;
  31    MSG3O='DATA ERROR - CORRECT AND PRESS ENTER';
  32    NAMEA, ADDRA, PHONEA, DATEA, AMOUNTA, COMMENTA=DFHBMFSE;
  33    EXEC CICS SEND MAP('XDFHPMB') DATAONLY;
        IF EIBTRNID='PADD' THEN COMLEN=7;
        ELSE COMLEN=80;
CICS_CONTROL:
  34    EXEC CICS RETURN TRANSID(EIBTRNID) COMMAREA(COMMAREA)
                                           LENGTH(COMLEN);
NOTMODF:
  35    MESSAGES='FILE NOT MODIFIED';
        GO TO PMNU;
DUPREC:
  36    MESSAGES='DUPLICATE RECORD';
        GO TO PMNU;
NOTFOUND:
  37    MESSAGES='INVALID NUMBER - PLEASE REENTER';
        GO TO PMNU;
ERRORS:
  48    EXEC CICS DUMP DUMPCODE('ERRS');
        MESSAGES='TRANSACTION TERMINATED';
PMNU:
        SUBSTR(ADDR(XDFHPMAO)->CHSTR,1,STG(XDFHPMAO))
                =LOW(STG(XDFHPMAO));
  39    MSGA=DFHBMBRY;
        MSGO=MESSAGES;
  40    EXEC CICS SEND MAP('XDFHPMA') ERASE;
  41    EXEC CICS RETURN;
END;
```

Program Notes

1. The length of the COMMAREA is tested.
2. The program exits are set up.
3. Map XDFHPMA is received.
4. The account number is saved.
5. If the program was invoked by the transaction identifier 'PADD', a title and command message are moved to the title area.
6. The record key is moved to the map area and to the COMMAREA.

7. In the case of the PADD transaction, the amount field has the modified data tag and the numeric attribute byte set on so only numeric data can be entered, and if no data is entered, the field contains the original data if it has not been modified when the contents of map XDFHPMB are mapped in.

8. The exit for the record not found condition is set up.

9. The file control READ reads the file record into the file area.

10. If the record is coded as deleted, it is treated as not found.

11. If the program was invoked by the transaction identifier 'PINQ' a title and command message are moved to the map area.

12. This invocation of the program ends.

13. If the program was invoked by the transaction identifier 'PUPD' a title and command message are moved to the map area.

14. The file record is moved to COMMAREA and the length of the COMMAREA to be returned is set up.

15. The fields from the file area are moved to the map area.

16. The screen is erased and the map XDFHPMB is sent to the terminal.

17. The program exits are set up.

18. This command maps in the contents of the screen.

19. The file control READ UPDATE reads the file using the number from the last invocation of this program which is stored in COMMAREA.

20. The fields from the last invocation are checked against those on the current file record.

21. A message and attribute bytes are moved.

22. The contents of the map XDFHPMB are sent to the terminal.

23. The message 'FILE UPDATED' is moved to MESSAGES.

24. The message 'RECORD ADDED' is moved to MESSAGES.

25. Any required editing steps should be inserted here. A suitable form of editing should be used here to ensure valid records are placed on the file.

26. The record to be written to the file is created.

27. The record fields, date, time and terminal identification are moved to the transient data area.

28. This record is written to a transient data file.

29. The updated record on the file is rewritten.

30. The added record is rewritten to the file.

31. An error message is moved.

32. The fields from the map have the modified data tag attribute set so that data is still in those fields when the map is received.

33. The contents of the map XDFHPMB are sent to the screen.

34. Control is returned to CICS along with the name of the transaction to be invoked when an attention key is pressed at the terminal, and data associated with this transaction is returned in the COMMAREA.

35. If no fields were modified, the message 'FILE NOT MODIFIED' is moved to MESSAGES.

36. If a duplicate record condition exists, the message 'DUPLICATE RECORD' is moved to MESSAGES.

37. If the file record was not found, the message 'INVALID NUMBER — PLEASE REENTER' is moved to MESSAGES.

38. On an error (notes 3, 9, 12, 16, 18, 22, 28, 29, 30, 33, 34, and 40) a dump is taken and the message 'TRANSACTION TERMINATED' is moved to messages.

39. The bright attribute is turned on, and MESSAGES is moved to the map area.

40. The screen is erased, and map XDFHPMA is transmitted to the screen.

41. The program ends.

MAP XDFHPMA (MENU SCREEN)

Map Definition

```
MAPSET   DFHMSD TYPE=&SYSPARM,MODE=INOUT,CTRL=(FREEKB,FRSET),LANG=PLI, *
               STORAGE=AUTO
XDFHPMA  DFHMDI SIZE=(12,40)
         DFHMDF POS=(1,10),LENGTH=21,INITIAL='OPERATOR INSTRUCTIONS',
               HILIGHT=UNDERLINE
         DFHMDF POS=(3,1),LENGTH=27,INITIAL='OPERATOR INSTR - ENTER PMN*
               U'
         DFHMDF POS=(4,1),LENGTH=38,INITIAL='FILE INQUIRY   - ENTER PIN*
               Q AND NUMBER'
         DFHMDF POS=(5,1),LENGTH=38,INITIAL='FILE BROWSE    - ENTER PBR*
               W AND NUMBER'
         DFHMDF POS=(6,1),LENGTH=38,INITIAL='FILE ADD       - ENTER PAD*
               D AND NUMBER'
         DFHMDF POS=(7,1),LENGTH=38,INITIAL='FILE UPDATE    - ENTER PUP*
               D AND NUMBER'
MSG      DFHMDF POS=(11,1),LENGTH=39,INITIAL='PRESS PA1 TO PRINT--PRESS*
                CLEAR TO EXIT'
         DFHMDF POS=(12,1),LENGTH=18,INITIAL='ENTER TRANSACTION:'
         DFHMDF POS=(12,20),LENGTH=4,ATTRB=IC,COLOR=GREEN,HILIGHT=REVER*
               SE
         DFHMDF POS=(12,25),LENGTH=6,INITIAL='NUMBER'
KEY      DFHMDF POS=(12,32),LENGTH=6,ATTRB=NUM,COLOR=GREEN,HILIGHT=REVE*
               RSE
         DFHMDF POS=(12,39),LENGTH=1
         DFHMSD TYPE=FINAL
         END
```

Menu Screen Layout

```
+OPERATOR INSTRUCTIONS
+OPERATOR INSTR - ENTER PMNU
+FILE INQUIRY   - ENTER PINQ AND NUMBER
+FILE BROWSE    - ENTER PBRW AND NUMBER
+FILE ADD       - ENTER PADD AND NUMBER
+FILE UPDATE    - ENTER PUPD AND NUMBER


+PRESS PA1 TO PRINT--PRESS CLEAR TO EXIT
+ENTER TRANSACTION:+XXXX+NUMBER+XXXXXX+
```

Figure 2.4-1. Menu Screen Layout (as seen on 40-character Screen)

MAP XDFHPMB (FILE SCREEN)

Map Definition

```
MAPSET   DFHMSD TYPE=&SYSPARM,MODE=INOUT,CTRL=(FREEKB,FRSET),LANG=PLI, *
               STORAGE=AUTO,EXTATT=MAPONLY
XDFHPMB  DFHMDI SIZE=(12,40)
TITLE    DFHMDF POS=(1,15),LENGTH=12
         DFHMDF POS=(3,1),LENGTH=8,INITIAL='NUMBER:',COLOR=BLUE
NUMB     DFHMDF POS=(3,10),LENGTH=6
         DFHMDF POS=(3,17),LENGTH=1
         DFHMDF POS=(4,1),LENGTH=8,INITIAL='NAME:  ',COLOR=BLUE
NAME     DFHMDF POS=(4,10),LENGTH=20,ATTRB=(UNPROT,IC)
         DFHMDF POS=(4,31),LENGTH=1
         DFHMDF POS=(5,1),LENGTH=8,INITIAL='ADDRESS:',COLOR=BLUE
ADDR     DFHMDF POS=(5,10),LENGTH=20,ATTRB=UNPROT
         DFHMDF POS=(5,31),LENGTH=1
         DFHMDF POS=(6,1),LENGTH=8,INITIAL='PHONE:  ',COLOR=BLUE
PHONE    DFHMDF POS=(6,10),LENGTH=8,ATTRB=UNPROT
         DFHMDF POS=(6,19),LENGTH=1
         DFHMDF POS=(7,1),LENGTH=8,INITIAL='DATE:   ',COLOR=BLUE
DATE     DFHMDF POS=(7,10),LENGTH=8,ATTRB=UNPROT
         DFHMDF POS=(7,19),LENGTH=1
         DFHMDF POS=(8,1),LENGTH=8,INITIAL='AMOUNT: ',COLOR=BLUE
AMOUNT   DFHMDF POS=(8,10),LENGTH=8,ATTRB=NUM
         DFHMDF POS=(8,19),LENGTH=1
         DFHMDF POS=(9,1),LENGTH=8,INITIAL='COMMENT:',COLOR=BLUE
COMMENT  DFHMDF POS=(9,10),LENGTH=9,ATTRB=UNPROT
         DFHMDF POS=(9,20),LENGTH=1
MSG1     DFHMDF POS=(11,1),LENGTH=39
MSG3     DFHMDF POS=(12,1),LENGTH=39
         DFHMSD TYPE=FINAL
         END
```

File Screen Layout

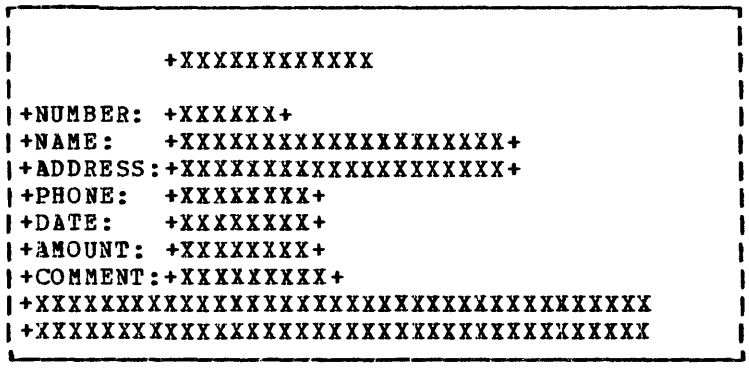


Figure 2.4-2. File Screen Layout (as seen on 40-character Screen)

Chapter 2.5. RPG II Programming

Translator Invocation

For details of translator invocation, see Chapter 3.5, "Preparation of Application Programs."

Translator Options

The translator provides a number of optional facilities, for example, to specify a listing of the source program on SYSLIST. These options are specified in the // OPTION job control statement and their defaults are as specified at system generation. The options are as follows:

Translator Options	Abbreviations	Default
DUMP|NODUMP	-	
LIST|NOLIST		-
LISTX|NOLISTX		

DUMP/NODUMP
: DUMP is used to prevent the translator from trapping unrecoverable errors; instead, a dump is produced. NODUMP must be used if trapping is required.

LIST/NOLIST
: specifies whether or not the translator is to produce a listing of the source program on SYSLIST.

LISTX/NOLISTX
: LISTX produces translator information messages which print (in hexadecimal) the bit patterns corresponding to the first argument of the translated call. NOLISTX causes no messages to be produced.

DEBUG Option

The DEBUG option, specified by including "D" in position 15 of the H-Spec, causes the execution diagnostic facility (EDF) to display the line numbers of commands as shown in the source listing. EDF is described in Chapter 4.1.

Command Syntax

A CICS/VS command for use in an RPG II application program consists of an EXEC statement followed by one or more ELEM statements. The commands are always coded on an RPG Calculation Specifications Form.

Figure 2.5-1 shows the RPG format of the command:

```
EXEC CICS READ INTO (RECORD) FILE ('MASTER') UPDATE
```

The following points should be noted:

1. The CICS/VS command-keyword (READ) is specified in an EXEC operation (line 01).
2. Operand keywords (INTO, etc.) are specified in ELEM operations.
3. Literal arguments for operands are specified in columns 33-42 (line 03). A literal argument may be a numeric or alphameric literal constant as defined in IBM DOS/VSE RPG II Language.
4. Variable-name arguments for operands are specified in columns 43-48 (line 02). The variable name may be the name of an RPG variable that describes an RPG II field, subfield, array, subarray, array-element (fixed subscript), subarray-element (fixed subscript), or data structure.
5. An indicator is required in columns 56-57 of the EXEC operation; but this indicator should not be tested. If it is omitted, an indicator of 13 is provided by the translator.
6. The "line" and "comments" fields may be used in the normal manner.

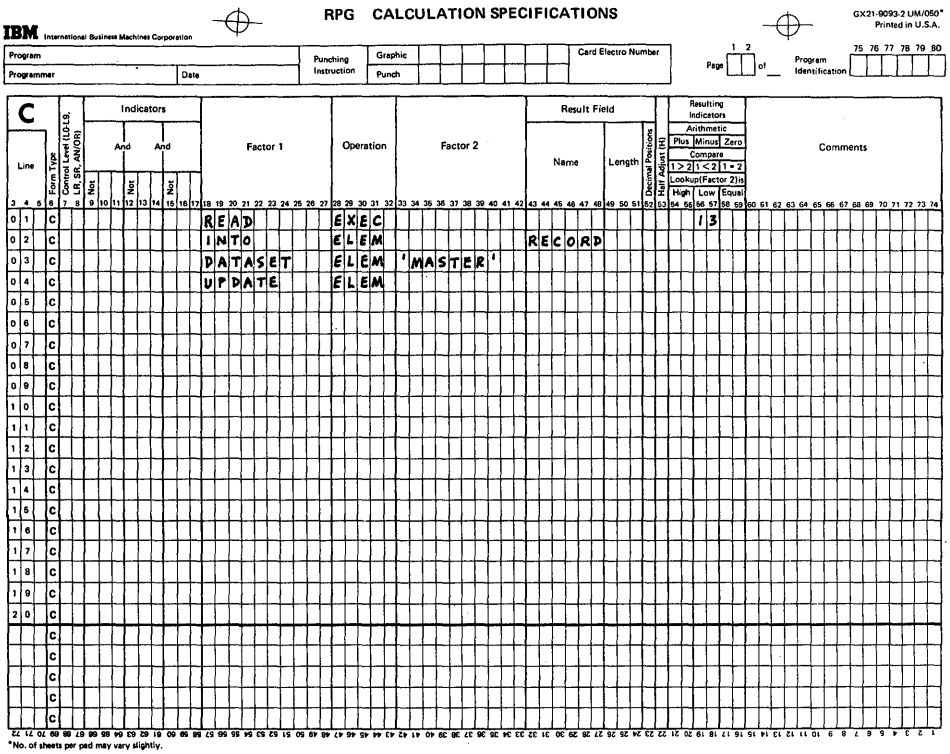


Figure 2.5-1. RPG II Format of a CICS/VS Command

General Rules for RPG II Programming

An RPG II application program using CICS/VS commands to request CICS/VS services is written in the same way as a batch RPG II program, except for the restrictions listed later in this section. CICS/VS commands can be included anywhere in the C-Specifications.

Order of Specifications

To satisfy the compiler requirements, the order of the specifications must be H, F, E, L, I, C, and O. An H-Specification must always be first and it must specify "main" or "subprogram" in position 55, and "C" in position 56. If the H-Spec is omitted, NOTRANSLATE is assumed and /INSERT statements only are processed by the translator.

Inserted Text

The /INSERT statement can be used in any of the specifications to include program pieces, data structures and maps that are held in libraries. The inserted text must be untranslated source, and must not itself contain /INSERT statements.

The format of a /INSERT statement is as follows:

Position	Contents
1–5	(See note)
6	H|F|E|L|I|C|O
7–13	/INSERT
14	blank
15	sublibrary-name|blank
16	. (period)
17–24	book-name
25–49	blank
50–74	comment
75–80	(See note)

(Note: the contents of these positions are standard RPG II entries as defined in IBM DOS/VSE RPG II Language.)

An explanation of the contents of the column positions in the above, apart from the obvious ones, or those referred to in the note, is as follows:

"sublibrary-name"
name of the sublibrary from which the insertion is to be made. If omitted, the name R is assumed.

"book-name"
name of the sublibrary member to be inserted.

The contents of the specified sublibrary book are generally inserted in place of the /INSERT statement. There are some exceptions, as follows:

- If the text that replaces a /INSERT statement in an E-Spec contains I-Specs the inserted text is placed in the I-Specs in the correct position. If maps are to be included, they should be inserted in this way.
- /INSERT statements for the data structures DBMSCA, DFHAID, and D2980 must be specified in the E-Specs. Each data structure is initialized during compilation by an array that overlays it. For this purpose, the required data is generated after the O-Specs following a ** statement.

*ENTRY PLIST Statement

The *ENTRY PLIST statement must be specified explicitly only if options that require pointer reference arguments are used in a command, or if DL/I is to be used.

If an *ENTRY PLIST statement is not specified explicitly, the translator generates an *ENTRY PLIST with the following PARM statements in the order shown:

```
PARM  DFHEIB
PARM  DFHCOM
PARM  DFHDUM
```

If specified explicitly, an *ENTRY PLIST must be the first C-Spec. The parameters are passed unchanged but the three required parameters (DFHEIB, DFHCOM, and DFHDUM) are always inserted at the beginning of the parameter list, if they have not been specified explicitly.

Restrictions

The following RPG II features cannot be used in an application program designed for use with CICS/VS.

1. Files, other than DB and DC files. A DC file may be any data set relevant to CICS/VS; it does not necessarily have to be associated with a terminal. (DB and DC files are defined in F-Specifications that have DB or DC starting in position 40).
2. RPG II statements requesting file operations. (CICS/VS commands are provided for the storage and retrieval of data in files.)
3. Tables and arrays loaded or saved at execution time.
4. Program error subroutines, except those that process the program errors "invalid index", "negative square root", and "error return from a CALL".
5. The op-codes CHAIN, DEBUG, DSPLY, DUMP, DUMPF, ERPGC, EXCPT, EXTCV, FORCE, KEYCV, READ, RPGCV, SETLL, and TIME.
6. File I/O errors. (CICS/VS commands are provided for CICS/VS error handling.)
7. Formatted dump. (CICS/VS commands are provided for obtaining CICS/VS dumps.)
8. AN or OR lines with CICS/VS commands.
9. Special Features: SORT

Map Length Restriction

The length of a map used with DC files and referred to in I-Specs and O-Specs must be less than, or equal to, 256, because the complete map (mapname concatenated with 'X') must be specified as a field, and a field length in RPG II cannot be greater than 256.

Description of BROWSE Sample Program

The browse program sequentially retrieves a page or set of records for display, starting at a point in a file specified by the terminal operator. Depressing the PF1 key or typing in F causes retrieval of the next page or paging forward. If the operator wishes to re-examine the previous records displayed, depressing the PF2 key or typing B allows paging backward.

To start a browse, the account number is mapped in and stored in a four entry key table in working storage. To retrieve a page, the key of the first record of that page is all that need be maintained in the table. The values in the key table are shifted right, so that the table is primed for the next page. A map area is obtained to move the fields from each record. The starting point of the browse is then established, the first record is read, and its fields are moved to the map area. As many successive records as can be shown on the screen are then read and set up. The sample program shows four records to a page (four lines). If conditions dictate displaying other than four lines, READNEXT and

associated commands should be added or deleted. If only one record can be accommodated, browse is still possible.

Following the last line, the key of the last record obtained is stored by ISAM in FLDA. The program must increase this by one, as this number becomes the search argument or key for the next page, if the browse is continued forward. As the browse operates on finding the first record of a page, where the record key is equal to or higher than the search argument, increasing the last record key by one assures that the next record, whatever its key values, will be retrieved when the browse is continued.

After viewing the first page, the operator may indicate page forward through the PF1 key or by typing F. The program proceeds directly to building the next page, as the key table is already conditioned. The browse may continue for as long as is desired (or until the end of the file is reached).

If the operator wishes to page backward with the PF2 key or by typing B, the key table entries are shifted left, so that the previous page is retrieved. The program resets the browse starting position and branches back to the main routine to construct a page. The backward browse depends on the number of keys that may be stored in the key table. If more than two page backwards in a sequence are required, the four entry key table should be expanded.

The operator may cancel a browse at any time by depressing the clear key.

Key Table example

The following are the field functions:

```
FLDA   - Next page forward
FLDB   - Current page being viewed
FLDC   - Previous page
FLDD   - Page before previous page

( + additional backward paging keys, if needed)
```

Assume that the file contains the following records, and there will be two records to a page for display:

```
| 14 | 17 | 18 | 20 | 25 | 28 | ....|....
```

The operator keys in 15, indicating that the browse should start with the first record equal to or greater than 15. The program stores 15 in FLDA and FLDB.

```
| 15 | 15 | 0  | 0  |
  FLDA  FLDB  FLDC  FLDD
```

The program reads records 17 and 18 from the file and displays them at the terminal. The last record (18) is increased by one and stored in FLDA, to be ready for a page forward.

```
| 19 | 15 | 0  | 0  |
  FLDA  FLDB  FLDC  FLDD
```

The operator presses PF1 or types F to page forward and display the next page. The program uses FLDA (19) to retrieve records 20 and 25.

These are displayed after the keys are shifted right. The last record read (25) is increased by one and stored in FLDA.

```
| 26  | 19  | 15  | 0   |
  FLDA  FLDB  FLDC  FLDD
```

Additional page forward requests would cause the table entries to be shifted right, and a new entry stored in FLDA. Entries in FLDD are dropped during the shift right.

The operator presses PF2 or types B to page backward and display the previous page of two records. The keys are shifted left to place the starting key of the previos page displayed (15) in FLDA and FLDB. FLDD is moved to FLDC, and zeros are moved to FLDD.

```
| 15  | 15  | 0   | 0   |
  FLDA  FLDB  FLDC  FLDD
```

The program uses FLDA to retrieve records 17 and 18, which are then displayed. The last record (18) is increased by one and stored in FLDA for the next page forward.

```
| 19  | 15  | 0   | 0   |
  FLDA  FLDB  FLDC  FLDD
```

The operator is viewing the first page that was requested, after paging forward one page and then paging backward to the starting page. The sample program does not permit paging beyond the starting page, so that the operator may only page forward at this point or cancel the browse by pressing the clear key. Although browse permits paging forward to the end of the file, paging backward is limited by the number of table entries. The four-entry table allows going back two pages. If this is insufficient, a larger table will allow further backward paging.

The source listing is shown in the following RPG II specifications.

Listing of BROWSE Sample Program

RPG CONTROL AND FILE DESCRIPTION SPECIFICATIONS

GX21-9092 UM/050* Printed in U.S.A.

IBM International Business Machines Corporation

Program	BROWSE	Punching Instruction	Graphic		Card Electro Number
Programmer	Date		Punch		

Page 01 of 09 Program Identification BROWSE

Control Specifications

H

Line	Form Type	Size to Compile	Object Output	Listing Options	Size to Execute	Debug	MFCM Stacking Sequence	Date Format	Date Edit	Inverted Print	360/20 2501 Buffer	Number Of Print Positions	Alternate Collating Sequence	Model 20: Address to Start	Model 20: Work Tapes	Overlay Open	Overlap Printer	Binary Search	Tape Error	2152 Checking	Inquiry	Model 20: Read/Write/Compute	Keyboard Output	Sign Handling	1P Forms Position	Indicator Setting	File Translation	Punch MFCU Zeros	Numeric Characters	Table Load Halt	Shared I/O	Field Print	Formatted Dump	RPG to RPG II Conversion	Number of Formats	Refer to the specific System Reference Library manual for actual entries.
01	H																																		C

File Description Specifications

F

Line	Form Type	Filename	File Type	File Designation	End of File	Sequence	File Format	Block Length	Record Length	Mode of Processing	Length of Key Field or of Record Address Field	Record Address Type	Type of File Organization or Additional Area	Overflow Indicator	Key Field Starting Location	Extension Code E/L	Device	Symbolic Device	Labels S/N/E/M	Name of Label Exit	Extent Exit for DAM	Storage Index	File Addition/Unordered	Number of Tracks for Cylinder Overflow	Number of Extents	Tape Rewind	File Condition U1-U8	Continuation Lines: K	Option	Entry
02	F	FILEA	I	D			F		80							DC														
03	F																													
04	F																													
05	F																													
06	F																													
07	F																													
08	F																													
09	F																													
10	F																													
	F																													
	F																													

RPG EXTENSION AND LINE COUNTER SPECIFICATIONS

Form X21-9091-2 Printed in U.S.A.

IBM International Business Machines Corporation

Program	BROWSE	Punching Instruction	Graphic		Card Electro Number
Programmer	Date		Punch		

Page 02 of 09 Program Identification BROWSE

Extension Specifications

E

Line	Form Type	Record Sequence of the Chaining File	Number of the Chaining Field	From Filename	To Filename	Table or Array Name	Number of Entries Per Record	Number of Entries Per Table or Array	Length of Entry	P/B/L/R	Decimal Positions	Sequence (A/D)	Table or Array Name (Alternating Format)	Length of Entry	P/B/L/R	Decimal Positions	Sequence (A/D)	Comments
01	E					M1		1	39									
02	E					M2		1	39									
03	E					M3		1	39									
04	E																	
05	E																	
06	E																	
07	E																	
08	E																	
	E																	
	E																	

Line Counter Specifications

L

Line	Form Type	Filename	1: Line Number	1: FL or Channel Number	2: Line Number	2: OL or Channel Number	3: Line Number	3: Channel Number	4: Line Number	4: Channel Number	5: Line Number	5: Channel Number	6: Line Number	6: Channel Number	7: Line Number	7: Channel Number	8: Line Number	8: Channel Number	9: Line Number	9: Channel Number	10: Line Number	10: Channel Number	11: Line Number	11: Channel Number	12: Line Number	12: Channel Number
11	L																									
12	L																									
	L																									

*No. of sheets per pad may vary slightly.

RPG INPUT SPECIFICATIONS

IBM International Business Machines Corporation

GX21-9094-0 U/M050*
Printed in U.S.A.

Program	BROWSE	Punching Instruction	Graphic		Card Electro Number
Programmer		Date	Punch		

Page 03 of 09 — Program Identification BROWSE

Line	Form Type	Filename	Sequence (AND)	Number (1/N)	Option (O), U	Record Identifying Indicator: 1, 2, or DS	Record Identification Codes 1: Position	1: Not (N)	1: C/Z/D	1: Character	2: Position	2: Not (N)	2: C/Z/D	2: Character	3: Position	3: Not (N)	3: C/Z/D	3: Character	Sequence Select	P/B/L/R	Field Location From	Field Location To	Decimal Positions	Field Name	Control Level (L1-L9)	Matching Fields or Chaining Fields	Field Record Relation	Field Indicators: Plus	Field Indicators: Minus	Field Indicators: Zero or Blank
01	I	/INSERT R.XDRMA																												
02	I	/INSERT R.XDRMC																												
03	I	/INSERT R.FILEA																												
04	I	/INSERT R.DBMSCA																												
05	I	FAREA DS																												
06	I																				L	60	FLDA							
07	I																				7	120	FLDB							
08	I																				13	180	FLDC							
09	I																				L9	240	FLDD							
10	I																				25	300	STARTN							
11	I																													
12	I																													
13	I																													
14	I																													
15	I																													
16	I																													
17	I																													
18	I																													
19	I																													
20	I																													
	I																													
	I																													
	I																													
	I																													
	I																													

RPG CALCULATION SPECIFICATIONS

IBM International Business Machines Corporation

GX21-9093-2 UM/050*
Printed in U.S.A.

Program	BROWSE	Punching Instruction	Graphic		Card Electro Number
Programmer		Date	Punch		

Page 04 of 09 — Program Identification BROWSE

Line	Form Type	Control Level (L0-L9, LR, SR, AN/OR)	Indicators: And Not	And Not	And Not	Factor 1	Operation	Factor 2	Result Field: Name	Length	Decimal Positions	Half-Adjust (H)	Resulting Indicators: Arithmetic Plus	Arithmetic Minus	Arithmetic Zero	Compare 1>2	Compare 1<2	Compare 1=2	Lookup(Factor 2) is High	Lookup(Factor 2) is Low	Lookup(Factor 2) is Equal	Comments	
01	C						MOVEL	' '	MESSAG	39													
02	C						Z-ADD	0	FLDA														
03	C						Z-ADD	0	FLDB														
04	C						Z-ADD	0	FLDC														
05	C						Z-ADD	0	FLDD														
06	C					HANDLE	EXEC													L3			1
07	C					CONDITION	ELEM																
08	C					ERROR	ELEM		ERRORS														
09	C					MAPFAIL	ELEM		MENU														
10	C					ENDFILE	ELEM		ENDFIL														
11	C					NOTFND	ELEM		NOFILE														
12	C					HANDLE	EXEC													L3			3
13	C					AID	ELEM																
14	C					CLEAR	ELEM		MENU														
15	C					PF1	ELEM		PFORW														
16	C					PF2	ELEM		PBACKW														
17	C					RECEIVE	EXEC													L3			2
18	C					MAP	ELEM	'XDRMA'															4
19	C						MOVE	KEY1	FLDA														5
20	C					STARTBR	EXEC													L3			
	C																						
	C																						
	C																						
	C																						
	C																						

*No. of sheets per pad may vary slightly.

RPG CALCULATION SPECIFICATIONS

IBM International Business Machines Corporation

GX21-9093-2 UM/050* Printed in U.S.A.

Program BROWSE | Punching Instruction | Graphic Punch | Card Electro Number

Programmer | Date

Page 05 of 09 | Program Identification BROWSE

Line	Form Type	Control Level (L0-L9, LR, SR, AN/OR)	Indicators: And/Not	Factor 1	Operation	Factor 2	Result Field: Name	Length	Decimal Positions	Half Adjust (H)	Resulting Indicators: Arithmetic (Plus, Minus, Zero); Compare (1>2, 1<2, 1=2); Lookup(Factor 2)is (High, Low, Equal)	Comments	
01	C			DATASET	ELEM	'FILEA'							
02	C			RIDFLD	ELEM		FLDA						
03	C			PFORW	TAG								
04	C				Z-ADD	FLDD							6
05	C				Z-ADD	FLDC							
06	C				Z-ADD	FLDA							
07	C			BUILD	TAG								
08	C				Z-ADD	1							
09	C				MOVE	*ZERO	XDFHCO						
10	C			NXTLINE	TAG								
11	C			READNEXT	EXEC						13		7
12	C			INTO	ELEM		FILEA						
13	C			DATASET	ELEM	'FILEA'							
14	C			RIDFLD	ELEM		FLDA						
15	C				TESTB	'01234567'	STAT				10		8
16	C		10		GOTO	NXTLINE							
17	C			1	COMP	1					101010		
18	C				GOTO	L1							
19	C				MOVE	NUMB	NUMB10						9
20	C				MOVE	NAME	NAME10						

*No. of sheets per pad may vary slightly.

RPG CALCULATION SPECIFICATIONS

IBM International Business Machines Corporation

GX21-9093-2 UM/050* Printed in U.S.A.

Program BROWSE | Punching Instruction | Graphic Punch | Card Electro Number

Programmer | Date

Page 06 of 09 | Program Identification BROWSE

Line	Form Type	Control Level (L0-L9, LR, SR, AN/OR)	Indicators: And/Not	Factor 1	Operation	Factor 2	Result Field: Name	Length	Decimal Positions	Half Adjust (H)	Resulting Indicators: Arithmetic (Plus, Minus, Zero); Compare (1>2, 1<2, 1=2); Lookup(Factor 2)is (High, Low, Equal)	Comments	
01	C				MOVE	AMOUNT	AMOT10						
02	C				GOTO	L4							
03	C			L1	TAG								
04	C			2	COMP	1					101010		
05	C		10		GOTO	L2							
06	C				MOVE	NUMB	NUMB20						10
07	C				MOVE	NAME	NAME20						
08	C				MOVE	AMOUNT	AMOT20						
09	C				GOTO	L4							
10	C			L2	TAG								
11	C			3	COMP	1					101010		
12	C		10		GOTO	L3							
13	C				MOVE	NUMB	NUMB30						
14	C				MOVE	NAME	NAME30						
15	C				MOVE	AMOUNT	AMOT30						
16	C				GOTO	L4							
17	C			L3	TAG								
18	C			4	COMP	1					101010		
19	C		10		GOTO	L4							
20	C				MOVE	NUMB	NUMB40						

*No. of sheets per pad may vary slightly.

IBM International Business Machines Corporation

RPG CALCULATION SPECIFICATIONS

GX21-9093-2 UM/050* Printed in U.S.A.

Program	BROWSE	Punching Instruction	Graphic		Card Electro Number
Programmer	Date		Punch		

Page 07 of 09 Program Identification BROWSE

Line	Form Type	Control Level (LO L9, LR, SR, AN/OR)	Indicators: And / Not	Factor 1	Operation	Factor 2	Result Field: Name	Length	Decimal Positions	Half Adjust (H)	Resulting Indicators: Arithmetic Plus / Minus / Zero; Compare 1>2 / 1<2 / 1=2; Lookup(Factor 2)is High / Low / Equal	Comments	
01	C				MOVE	NAME	NAME40						
02	C				MOVE	AMOUNT	AMOT40						
03	C			L4	TAG								
04	C			L	ADD	1	1						
05	C			5	COMP	1					10 10		
06	C		10		GOTO	NXTLINE							
07	C			DISPREC	TAG								
08	C			SEND	EXEC						13		11
09	C			MAP	ELEM	'XDRMC'							
10	C			ERASE	ELEM								
11	C			REPEAT	TAG								
12	C			RECEIVE	EXEC						13		12
13	C			MAP	ELEM	'XDRMC'							
14	C			DIR1	COMP	'F'					10		
15	C		10		GOTO	PFORW							
16	C			DIR1	COMP	'B'					10		
17	C		10		GOTO	PBACKW							
18	C				GOTO	MENU							
19	C			ENDFIL	TAG								
20	C				MOVEAM1		MSG10						13
	C												
	C												
	C												
	C												
	C												

*No. of sheets per pad may vary slightly.

IBM International Business Machines Corporation

RPG CALCULATION SPECIFICATIONS

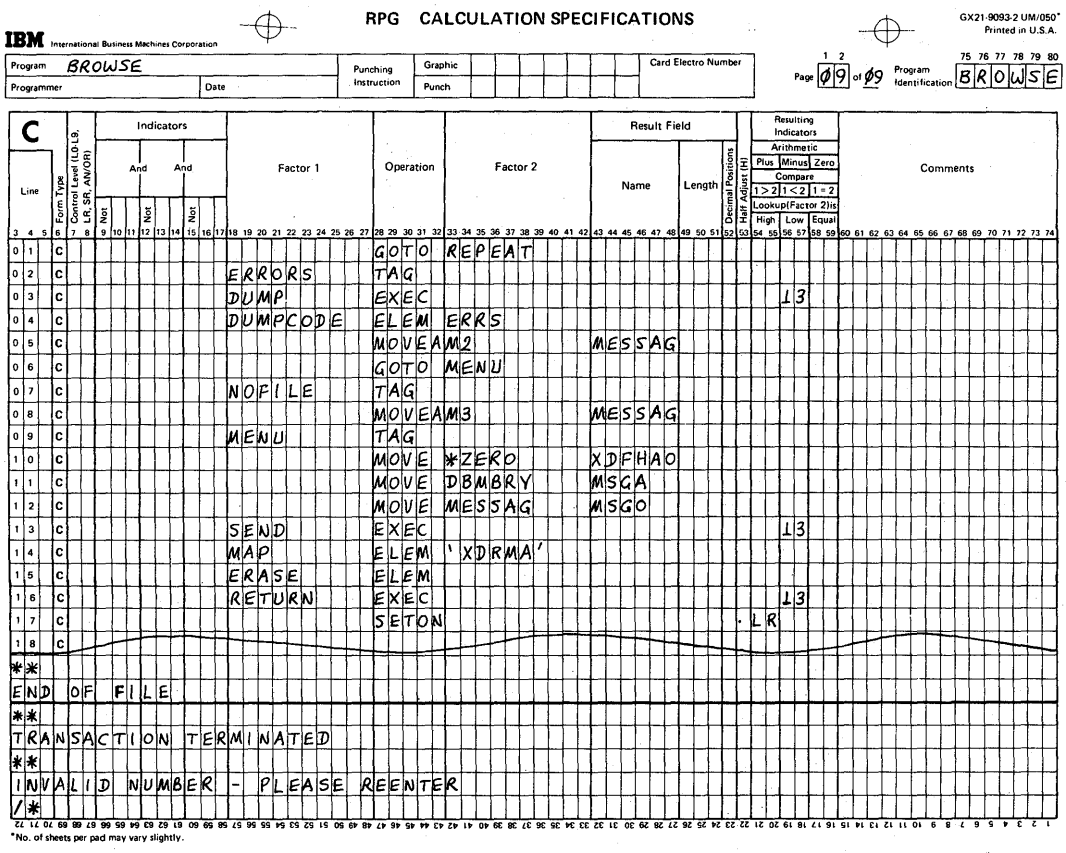
GX21-9093-2 UM/050* Printed in U.S.A.

Program	BROWSE	Punching Instruction	Graphic		Card Electro Number
Programmer	Date		Punch		

Page 08 of 09 Program Identification BROWSE

Line	Form Type	Control Level (LO L9, LR, SR, AN/OR)	Indicators: And / Not	Factor 1	Operation	Factor 2	Result Field: Name	Length	Decimal Positions	Half Adjust (H)	Resulting Indicators: Arithmetic Plus / Minus / Zero; Compare 1>2 / 1<2 / 1=2; Lookup(Factor 2)is High / Low / Equal	Comments	
01	C				MOVE	DBMBRY	MSG2A						
02	C				GOTO	DISPREC							
03	C			PBACKW	TAG								
04	C			0	COMP	FLDC					10		14
05	C				GOTO	TOOFAR							
06	C				Z-ADD	FLDC	FLDA						15
07	C				Z-ADD	FLDC	FLDB						
08	C				Z-ADD	FLDD	FLDC						
09	C				Z-ADD	0	FLDD						
10	C			STARTN	COMP	FLDA					10 10		
11	C		10	1	ADD	FLDA	FLDA						
12	C			RESETBR	EXEC						13		
13	C			DATASET	ELEM	'FILEA'							
14	C			RIDFLD	ELEM		FLDA						
15	C				GOTO	BUILD							
16	C			TOOFAR	MOVE	DBMBRY	MSG1A						16
17	C				MOVE	DBMDAR	MSG2A						
18	C			SEND	EXEC						13		17
19	C			MAP	ELEM	'XDRMC'							
20	C			DATAONLY	ELEM								
	C												
	C												
	C												
	C												
	C												

*No. of sheets per pad may vary slightly.



RPG CALCULATION SPECIFICATIONS

IBM International Business Machines Corporation

GX21-9093-2 UM/050* Printed in U.S.A.

Program	BROWSE	Punching Instruction	Graphic	Card Electro Number
Programmer	Date		Punch	

Page 09 of 09 — Program Identification BROWSE

Line	Form Type	Control Level (LO,L1,L2,L3,L4,L5,L6,L7,L8,L9, LR, SR, AN/OR)	Indicators: And / Not	Factor 1	Operation	Factor 2	Result Field: Name	Result Field: Length	Result Field: Decimal Positions	Result Field: Half Adjust (H)	Resulting Indicators: Arithmetic: Plus / Minus / Zero	Resulting Indicators: Compare: 1>2 / 1<2 / 1=2	Resulting Indicators: Lookup(Factor 2)is: High / Low / Equal	Comments
01	C				GOTO	REPEAT								
02	C			ERRORS	TAG									
03	C			DUMP	EXEC								L3	19
04	C			DUMPCODE	ELEM	ERRS								
05	C				MOVEAM2		MESSAG							
06	C				GOTO	MENU								
07	C			NOFILE	TAG									
08	C				MOVEAM3		MESSAG							18
09	C			MENU	TAG									
10	C				MOVE	*ZERO	XDFHAO							20
11	C				MOVE	DBMBRY	MSGA							
12	C				MOVE	MESSAG	MSGO							
13	C			SEND	EXEC								L3	21
14	C			MAP	ELEM	'XDRMA'								
15	C			ERASE	ELEM									
16	C			RETURN	EXEC								L3	22
17	C				SETON									
18	C										LR			
**														
END OF FILE														
**														
TRANSACTION TERMINATED														
**														
INVALID NUMBER - PLEASE REENTER														
/*														

*No. of sheets per pad may vary slightly.

Program Notes

1. The program exits are set up.
2. This command maps in the account number.
3. The exits for each of the defined function keys are set up.
4. The starting key is stored in field A in the key table.
5. This command establishes the browse starting point.
6. The keys in the table are shifted right in anticipation of a continuation of a browse.
7. The READNEXT reads the first record into the file area.
8. If the record is flagged as deleted, the program reads the next record.
9. The required fields are moved from the file area to the map area.
10. The same basic commands are repeated to read and set up the next three lines. The same file area is used and, therefore, the fields must be reused after each READNEXT.
11. The screen is erased and the page is displayed at the terminal.

12. The browsing command (CLEAR, PF1, or PF2 key, or 'F' or 'B') is read from the terminal, and control is passed according to the operator response (see note 3).

13. If the end of file is reached on any READNEXT, any records read to that point are displayed, together with the message 'END OF FILE'. The label to which this routine branches allows the operator to restart the browse at a different point. The bright attribute for the page backward message is turned on.

14. If the PF2 key is depressed or B typed in, indicating page backward, and FLDC contains zeros, further backward paging is not possible. The program branches to TOOFAR (see note 17).

15. If not, the key fields are shifted left to retrieve the previous page and the starting point for the browse reset accordingly.

16. The table limit is exceeded. An output map area is acquired, the bright attribute for the page forward message is turned on, and a dark attribute is moved to the page backward message.

17. An error message is written to the terminal.

18. On the record NOTFND condition, the message 'INVALID NUMBER — PLEASE REENTER' is moved to MESSAGES.

19. On an error (notes 2, 5, 7, 11, 12, 17, 19 or 21) a dump is taken and the message 'TRANSACTION TERMINATED' is moved to MESSAGES.

20. The map area is cleared. This is also the entry point if the clear key was depressed. The bright attribute to highlight the message is turned on, and the message 'TRANSACTION TERMINATED' or the default message is moved to MESSAGES.

21. The screen is erased and map XDRMA is displayed.

22. The program ends.

Menu Screen Layout

```
+------------------------------------------+
|                                          |
|         +OPERATOR INSTRUCTIONS           |
|+OPERATOR INSTR - ENTER RMNU              |
|+FILE INQUIRY   - ENTER RINQ AND NUMBER   |
|+FILE BROWSE    - ENTER RBRW AND NUMBER   |
|+FILE ADD       - ENTER RADD AND NUMBER   |
|+FILE UPDATE    - ENTER RUPD AND NUMBER   |
|                                          |
|                                          |
|+PRESS PA1 TO PRINT--PRESS CLEAR TO EXIT  |
|+ENTER TRANSACTION:+XXXX+NUMBER+XXXXXX+   |
+------------------------------------------+
```

Figure 2.5—2. Menu Screen Layout (as seen on 40—character Screen)

Part 3. System Programming

Chapter 3.1. CICS/VS System Design

This chapter provides an introduction to CICS/VS program logic. Other information relating to system design will be found in Chapter 3.8, "Performance." More detailed information can be found in the CICS/VS Introduction to Program Logic.

Introduction to CICS/VS Program Logic

The entry level system contains the following components:

Terminal control
— controls all terminal activity.

Task control
— controls all CICS/VS tasks.

Program control
— manages CICS/VS application programs.

Basic mapping support
— handles terminal data formatting.

File control
— controls file I/O operations.

Transient data control
— controls sequential data files and intrapartition data.

Trace control
— provides a trace facility.

Dump control
— provides dumps to aid testing.

Temporary storage control
— provides for temporary data storage.

Storage control
— obtains working storage areas.

Interval control
— provides time-dependent facilities.

TRANSACTION FLOW

The following is a description of the use of the CICS/VS components in a typical inquiry application. In the application described, a terminal operator enters the transaction code INQY and an account number, signifying that he wants a particular record to be displayed at the terminal. The application program retrieves this record, selects fields required, and displays them at the terminal.

Terminal Control

Assume the terminal operator has entered the transaction code and an account number. Terminal control reads this input message into a terminal input/output area (TIOA).

Terminal control consists of a terminal control program (TCP for BTAM and ZCP for VTAM), a terminal control table (TCT), and terminal error routines.

The terminal control program controls terminal operations through BTAM or VTAM. Terminal control's primary functions are polling and addressing. Polling checks all remote terminals periodically to determine whether any have input to transmit, and is an invitation to send input to the application program. Addressing is having the computer check to see if a terminal is ready to receive output. With basic mapping support, terminal control provides an application program with the ability to communicate with a terminal. Terminal control also handles I/O errors, and keeps track of which task is associated with which terminal.

The terminal control table aids in controlling terminal operations, in that it specifies the communication line characteristics, the types of terminals, special features, and terminal priorities. The terminal control program refers to this table in performing its functions. The polling sequence, that is, the order in which the terminals should be polled, is defined in the TCT. Also stored is operational data, such as an indication that a certain terminal is temporarily out of service. This terminal is excluded from polling until it is put back in service.

CICS/VS provides several terminal error routines. When unrecoverable I/O errors occur, terminal control uses a terminal abnormal condition program (TACP for BTAM, NACP for VTAM) to analyze the condition. Statistics are maintained, and an error message is sent to a transient data file.

Task Control

Terminal control passes control to task control, which creates a task for the inquiry transaction. A terminal can have only one transaction associated with it at a given time, and the terminal is locked until the program makes a response to the terminal.

Task control consists of a task control program (KCP) and a program control table (PCT).

The task control program keeps track of the status of many tasks being processed concurrently (multitasking). Transactions are not usually processed through to completion in a single, uninterrupted operation. A transaction may be processed up to a file I/O instruction, for instance, whereupon another waiting task receives control. Therefore, there may be many incomplete tasks which task control must control simultaneously.

Task control validates transactions by checking the program control table, which lists all valid transaction codes and the associated programs, so that control may be transferred to the correct program. If an operator entered an invalid transaction code, task control would not find it in the program control table, and an error message would automatically be sent to the terminal.

To control each task, task control acquires a task control area (TCA) through storage control (see the section "Storage Control" later in this chapter). If desired, this area may be extended to include a transaction work area (TWA), which may be used by an application program during the life of a transaction. The TCA and TWA are released when the task terminates.

Program Control

Task control passes control to program control, which keeps track of the locations of the application programs.

Program control consists of a program control program (PCP), a processing program table (PPT), and an abnormal condition program (ACP).

The program control program manages application programs that are stored in the DOS/VS core image library. All programs are loaded into virtual storage when CICS/VS is initialized and are resident throughout CICS/VS operation except for RPG II programs, which must be reloaded each time they are used (see Chapter 3.4).

The processing program table is used by the program control program to determine a program's location in virtual storage during CICS/VS operation. Programs are relocatable, and may be in different main storage locations from run to run. The PPT contains the program size, source language, and other program information.

Under certain conditions, such as an unrecoverable I/O error, the application program may wish to end the task. However, if a program check occurs, this type of abnormal end is treated differently. If a program terminates abnormally in a batch system, the operating system may purge the job in that partition and schedule another. CICS/VS should clearly not be purged because of one application program's exception condition. Therefore, CICS/VS intercepts these program checks and terminates that task only.

User Application Program

Program control passes control to the application program, in this example to the program that handles the transaction code INQY.

In summary, before passing control to the application program, CICS/VS has read the input (INQY and account number) into a terminal I/O area, validated the transaction code, and initiated a transaction. The application program may now process this input and issue commands to request services needed in handling the transaction.

Basic Mapping Support — Input

Basic Mapping Support (BMS) is a CICS/VS feature that allows the user to define layouts of 3270 screens. Its use and functions are described in more detail in Chapter 2.1 and 3.5 of this manual. In the context of this transaction flow description, the next step is as follows:

A BMS command is issued by the application program to format and move the account number from the terminal I/O area to a symbolic description

map. Control passes to the BMS module to perform this service and returns to the application program.

Basic mapping support consists of the BMS modules and BMS physical maps.

BMS uses physical maps as requested by the application program to control the formatting (or mapping) of terminal I/O data. A physical map is defined for each 3270 screen layout used in the application program. The physical map contains such information as the terminal position and length of each data field, 3270 field attribute characters, extended attributes (if any), and constant data for headings and keywords.

File Control

The application program issues a file control command to retrieve a record from a file or a data base. File control reads the record into a file area, and returns control to the program.

File control consists of the file control program (FCP) and the file control table (FCT).

The file control program provides file services or file management. It supports read only (inquiry), update, add, and VSAM browse functions, and provides a file protection function called exclusive control. This function is invoked by the file control program if several tasks request the same record for updating. Exclusive control places all tasks, with the exception of the first, into a wait queue, so that only one task at a time updates that record and returns it to the file before another task may access the record.

The file control table contains, for each file, user-supplied file characteristics including the access method, record format and length, and block size. The FCT also specifies what operations can be performed on each file. A file may be online and yet effectively protected against modifications by specifying read only. CICS/VS does not allow any program to update such a file. New files may be added, old files deleted, and characteristics such as blocking factors modified, without necessarily having to change the application programs.

Transient Data Control

When a file is changed by updating a record or adding a new one, a record of the change can be logged through a transient data command. This provides an audit trail and allows reconstruction of records, if necessary. Inquiries do not change records and are not normally logged, but, in this example, the record is logged because statistics are wanted. Control returns to the application program following logging.

Transient data control consists of the transient data control program (TDP) and the destination control table (DCT).

The transient data control program is a queuing facility which stores records in the order received on a sequential DASD or tape file. These may be records for later batch processing, audit records, statistics, or error messages. Sequential input files may also be read by issuing transient data control commands.

The destination control table contains information used by the transient data control program to direct data to the correct file. This includes the file name, and record and file descriptions.

Trace Control

The trace control program is a CICS/VS debugging aid which may be used to trace the processing path of an application program. This module is invoked for each CICS/VS command if trace is active, and after execution of the trace control program, control returns to the application program.

Trace control consists of the trace control program (TRP) and the trace table (TRT) and, optionally, the auxiliary trace file.

Whenever a trace command is encountered, CICS/VS makes an entry in the trace table. CICS/VS also records in the trace table which CICS/VS components have been used by the application program.

The trace table resides in virtual storage, and a copy is provided if the transaction is abnormally terminated or requests a transaction dump. It is also provided together with a formatted dump.

Dump Control

If an unusual condition occurs during processing, the application program may issue a dump control command to write all transaction-related storage areas to a dump file. After the dump has been taken, control returns to the application program.

Dump control consists of the dump control program (DCP) and the dump utility program (DUP).

Dump control commands may be inserted at strategic points in a program to facilitate debugging. The dump control program dumps storage areas to a sequential file (DASD or tape) for subsequent printing. The dump control commands should be removed from the application program when it has been debugged.

Dump control commands may also be used to provide printed records of certain error conditions, such as unrecoverable I/O errors. Used in this way, the command is regarded not as a debugging aid, but as a permanent part of the program.

The dump utility program supplied with CICS/VS is a batch program that formats and prints the dump file.

Temporary Storage Control

The application program may need to store information for later retrieval by another task. The temporary storage control program allows the program to store such data in virtual storage before the next instruction is executed.

Temporary storage control consists of the temporary storage control program (TSP).

An application program may issue commands requesting that information be stored for subsequent retrieval by assigning a record to a named queue. Such records are queued and may be retrieved later in the same sequence in which they were stored, or randomly by entry number. All temporary storage records, whether single or queued, are retained by CICS/VS until purged by an application program.

Interval Control

The application program may need to start a task at some future time (possibly to process the data stored by the temporary storage control program). The interval control program allows the application program to start a task after a certain interval of time has expired, or at a certain time of day.

Interval control consists of the interval control program (ICP).

Basic Mapping Support — Output

The application program must now extract the necessary fields from the record in the file area, and set up a symbolic description map to be written to the terminal.

A BMS command formats the record fields for transmission to the terminal. BMS moves the data from the symbolic description map to a terminal I/O area, and it is then written to the terminal by terminal control. The terminal operator may view the record for as long as desired, and then initiate a new transaction.

Ending the Transaction

The transaction is terminated by issuing a program control RETURN command. All storage, except the TIOA, allocated to this transaction is released and made available for use by other transactions. Upon completion, control returns to task control which deletes this transaction from its transaction list. Task control then indicates to terminal control that no task is attached to the terminal. If input data is received, it will initiate another transaction. The TIOA is released only when terminal control has finished.

CICS/VS OPERATING ENVIRONMENT

The previous discussion considered only one terminal and one application program to be active. In reality, many terminals and programs may be operating. With CICS/VS, three types of multiple operations usually exist: multiprogramming via the operating system, multitasking within the CICS/VS partition, and multithreading of CICS/VS application programs; these operations are described next.

Multiprogramming

If CICS/VS is executed in a multipartition environment, it is usually in one of the highest priority partitions. Batch partitions receive control from VSE only when CICS/VS has no dispatchable transactions. Thus, as long as there is a transaction ready for processing, CICS/VS maintains system control. Control is released to VSE for continuation of a job in another partition only when there are no ready transactions. CICS/VS regains control as soon as any previously waiting CICS/VS transaction is ready to continue, or, if all active transactions are in wait state, as soon as a new transaction code is entered at a terminal.

Multitasking

Much as VSE controls concurrent execution of application programs across partitions (multiprogramming), CICS/VS controls concurrent execution of CICS/VS application programs within its partition. This is called multitasking. Whenever one task has to wait for the completion of an I/O operation, for example, CICS/VS assigns the processing unit to some other task that is ready to use it.

In the inquiry transaction flow described earlier, control was returned to the task after completion of file control or transient data I/O operations. When several tasks are running concurrently, a task that issues an I/O command, and thus has to wait for completion, will be suspended and another task that is ready for processing is dispatched. This overlapping of I/O operations and processing unit usage between several tasks is called task switching.

Multithreading

In CICS/VS, several transactions running concurrently may require the same application program. Rather than have more than one copy of a program in storage at the same time, one copy is used by various transactions. This process is called multithreading. An application program, especially one with several I/O operations, may have many transactions associated with it, some having gone through the first portion of the program, or up to the first I/O instruction, some the second, and so on. With only one copy of the program in storage, each transaction would be waiting its turn to continue through the next portion of the program, and on to completion. To control multithreading, task control uses the task control area for each transaction. This allows task control to determine whereabouts in a program each transaction is, or where it should return to resume processing when it receives control.

As programs may be used by more than one transaction, programs must not modify themselves. This requires that the code be left in its original condition, so that each transaction may be processed in exactly the same manner. Thus, any intermediate values or variables must be stored in a separate area for each transaction. With multithreading, each transaction must also have its own I/O areas, thus allowing each to issue I/O instructions independently of other transactions. For COBOL programs using the command-level interface, CICS/VS makes a copy of the user-defined working storage for each transaction so that the programs can be used in a multithreading environment.

SYSTEM CONTROL FUNCTIONS

To control overall system, the following functions are included in CICS/VS:

System initialization
: The system initialization programs initialize CICS/VS for operation by loading the CICS/VS components according to requirements specified in startup override parameters. This process allows the CICS/VS system to be tailored for different requirements, such as weekends, second shift, or testing.

Master terminal
: The master terminal program allows control over the system during CICS/VS operation. Any terminal may be designated temporarily as a master terminal, and an individual may be authorized to perform master terminal functions through a security key. The master terminal functions include enabling or disabling programs, files, transactions, or terminals, and requesting shutdown.

System termination
: The system termination program is invoked through the master terminal and shuts down the system; and, if requested, it does so only after all previously accepted transactions have been processed. It prevents further terminal entries, closes files, finalizes CICS/VS statistics, and sends shutdown messages to the console operator.

Sign-on
: The sign-on program and table optionally provide security by restricting the personnel allowed to operate terminals. All operators may be required to sign on before being allowed to operate terminals, and sign off when finished. Through security keys only selected operators may be allowed to use restricted programs and have access to confidential files.

System statistics
: Various modules accumulate statistics that are useful in analyzing and improving performance. The statistics include the number of transactions initiated, the number of times a program was used, and the number of file operations.

CICS/VS Command Language Translators
: These translators permit the use of CICS/VS commands in assembler language, COBOL, PL/I, or RPG II programs, making possible the use of these languages with CICS/VS.

Chapter 3.2. Supervisor Generation

CICS/VS 1.5 ELS operates under a default VSE supervisor for which default values of the BGPGR and FnPGR operands of IOTAB have been overridden as described in note 3 (below).

Notes:

1. The VSE supervisor generation parameters applicable to CICS/VS are:

   ```
   SUPVR    MODE={370|E}
   ```

2. The number of programmer logical units in the system is defined during supervisor generation, and the default number is 10.

   The jobstream generated by DFHJCELS, for use with the sample tables, references programmer logical units numbered up to SYS014, but can be adapted to run user applications, referencing logical units with numbers outside this range. The default of 10 should thus be overridden, at supervisor generation time, by a value of at least 14, as shown below:

   ```
   IOTAB    BGPGR=14
            FnPGR=14
   ```

Chapter 3.3. Installation of Distribution Volume

The machine-readable material for CICS/DOS/VS, including the entry level system, is shipped on one or more Distribution Tape Reels (DTR). The recording density is 1600 or 6250 bpi, as ordered. The tape reel is termed a distribution volume. The information in this chapter, briefly describes the contents of the volumes.

For a full description of how to process the machine-readable material, the user should refer to the Program Directory supplied with the distribution volume.

Establishing Standard Labels

When VSE private libraries are used for CICS/DOS/VS, their DASD label information is stored in the standard label track or partition standard label area. These libraries are used throughout the CICS/DOS/VS system installation and servicing process. The label information should be added to existing standard or partition standard label information.

The file identification and date of the DLBL statement and the EXTENT information are those used when the libraries are restored from the distribution volume. See the Program Directory supplied with the volume for details of space required.

Contents of the Distribution Volumes

The files on the distribution volumes include the following modules:

- A private core-image library containing the entry level system.
- A private source-statement library, called the primary source-statement library, containing CICS/DOS/VS source code needed for application program and map preparation, table generation, and servicing. This library also includes the Z sub-library books, which contain job streams and data to run the sample application programs. Note that this library does not contain the source code for most of the CICS/DOS/VS control programs. Nor does it contain the A sub-library form of the CICS/DOS/VS macros, the source code for the command-level language translators, or the source code for the execution diagnostic facility (not normally needed online).
- Two private source-statement libraries containing further CICS/DOS/VS source. One of these libraries contains copycode and E.macros used by sysgen and E.macros for the macro interface; the other contains source for sysgen and maintenance modules.
- A private relocatable library used to build the entry level system. It contains certain CICS/DOS/VS object modules and VSE logic modules needed for CICS/DOS/VS table generation.

- A private source-statement library containing the A sub-library form of the CICS/DOS/VS macros. The library is not normally needed online.

The private core-image library contains sample application programs, maps, tables, and the CICS/DOS/VS entry level programs.

Processing the Distribution Volumes

Full instructions for processing the distribution tape are contained in the Program Directory. The information given includes the following:

- Descriptions of the layout and content of the files on the tape.
- Disk space requirements for the distributed tape files.
- Job control statements for restoring the tape files.

Bring Up the System to Run the Sample Program

Sample application programs written in Assembler language, COBOL, PL/I, and RPG II are provided in the entry level system private core-image library. The relevant books, both of which are in the primary source library, are:

Z.DFHSTAP
: containing the source of the sample application programs.

Z.DFHSTMP
: containing the maps and data area descriptions used by the sample application programs.

To bring up the system to run the sample programs, the DFHJCELS macro is provided. This macro generates a jobstream which on execution allows the sample transactions to be tried.

THE DFHJCELS MACRO - CICS/DOS/VS-ELS INSTALLATION AID

This macro generates a jobstream with which the sample applications may be run and upon which the installation jobstream for bringing up CICS/DOS/VS-ELS may be based.

Input: The macro is coded with up to eight operands which describe the environment in which the entry level system will be running.

Output: A jobstream which should suit most installation configurations.

Running the Generated Jobstream

Disk Space: The extents of the files required to run the sample applications are calculated in the macro, based upon the SFILE and SAMPVOL operands. The macro assumes that there are 20 contiguous cylinders (or equivalent) on a pack designated for sample application testing.

Partition Standard Labels and ASSGN Statements

The first job to be generated by DFHJCELS is a partition standard label job, and is intended to be merged with existing partition standard labels in the partition to be used for CICS/VS. The macro will generate DLBL and EXTENT information for the following:

1) CICS Private Core Image Library (Operand: CICSCIL)
2) VSAM.SPACE (If SFILE=VSAM) (Operand: SFILE)
3) SAMPLE.TEST.FILEA (A subset of VSAM.SPACE if SFILE=VSAM)
   (Operand: SFILE,SAMPVOL)
   CICS.INTRA (Operand: SFILE)
4) CICS.DUMPA
5) CICS.DUMPB
   dump data Sets
6) CICS.AUXTRACE auxiliary trace data sets
7) CICS.MSGUSR
8) CICS.LOGUSR
   sample application extrapartition data sets
9) CICS.INTRA sample application intrapartition dataset.

In the partition standard label job are permanent ASSGN statements for the following:

1) Private Core-Image Library - SYSCLB (Operand: CICSCIL)
2) Sample Application Testing Pack (Operand: SAMPVOL)
3) Local Screens L77A and L77B (Operand: SYS011&12)
4) BTAM Local Printer L860 (Operand: SYS013)
5) BTAM Remote Control Unit (Operand: SYS014)

Running with VSAM

When the operand SFILE=VSAM is used, it refers to the access method of the sample application testing file (//DLBL FILEA) and the transient data (intrapartition) file (//DLBL DFHTRA). The second job contains a step which defines a VSAM space "VSAM.SPACE", in which "SAMPLE.TEST.FILEA" and "CICS.INTRA" will reside.

Sample Application Test Data

The second job contains a step which will set up a "SAMPLE.TEST.FILEA", using whichever access method was selected by the SFILE operand.

Running with VTAM

The generated jobstream is intended only for use with BTAM; both the jobstream and initialization parameters would need modification for VTAM.

ELS Initialization Parameters

These are contained in a SYSIPT data stream and are generated to suit the environment described by the macro operands; however certain assumptions have been made which may not suit your installation, and the opportunity is provided for overriding the generated options.

UPSI 1 indicates to DFHSIP that a SYSIPT data stream is to be expected.

UPSI 001 indicates to DFHSIP that overrides and options will be input from the VSE console.

The jobstream is generated with both UPSI options in effect. There are two initialization commands which can switch the source of these input options during initialization and these are used to provide the user with an "initialization conversation" at bring-up time, as follows:

The SYSIPT data stream provides a selected CICS/DOS/VS-ELS initialization option, plus a narrative of any alternatives. The command 'CN' follows and there is a wait on the DOS/VSE console for the user's reply. If the given option is acceptable, the user replies 'SI'; this command causes the SYSIPT data stream to be resumed.

If, however, the user wishes to override the given option, he replies by typing in the option he wants on the VSE console typewriter, followed by ',SI', and the SYSIPT data stream is resumed.

Having established the particular set of initialization options for a standard CICS/DOS/VS-ELS bring-up, the user should remove the following from the jobstream

1) Narrative comments
2) CN commands (except the final one)

DFHJCELS Macro Specification

```
DFHJCELS  [JOB=ELS|name]
          [,SFILE=ISAM|VSAM]
          [,SAMPVOL=(volid, volsys, voldev, volstart, voldensity)]
          [,CICSCIL=(cicsname, cicsvol)]
          [,SYS011=cuu]
          [,SYS012=cuu]
          [,SYS013=cuu]
          [,SYS014=cuu]
```

JOB

generates a prefix for numbered jobs in the generated jobstream. ELS is the default, but any valid jobname of one through seven characters in length may be specified.

SFILE
defines the access method of the sample application testing file and the intrapartition data set. The allowed values are ISAM or VSAM (only VSAM for 3350 and FBA devices).

SAMPVOL
supplies information about the Sample Application Testing Pack.

volid
represents the six-character serial number of the volume on which the sample application testing file and other data sets will be created.

volsys
is the three digit system number which is used in generating the permanent ASSGN for the testing pack "volid". The default is 008, and the value specified should not be 009 through 014, as this conflicts with values coded in the various pre-generated CICS/VS programs and tables. A full list of the usage of 'SYS' numbers 9 through 14 is given in the generated jobstream.

voldev
describes the device type on which the sample application testing file and other CICS/VS datasets are to be held. The device type may be 2314, 3330, 3340, 3350 or FBA, but note that neither 3350 nor FBA is allowed unless SFILE=VSAM.

volstart
is the number of the cylinder, relative to zero, on which the sample application testing file and other data sets start. For FBA devices it is the relative block number (between 2 and 99,000,000).

voldensity
may be either SINGLE or DOUBLE. It should be coded DOUBLE only if 'voldev' is 3340 and the 3348 model 70/70F or 'voldev' is 3330 and the 3336 Model 11 is being used. SINGLE is the default.

CICSCIL
this supplies information for the DLBL and ASSGN for the CICS/VS Private Core Image Library.

cicsname
is the filename on the DLBL card; it may be coded in quotes.

cicsvol
is the six-character volume identifier of the pack on which the CICS/VS PCIL resides, even if it is the same as "volid" as coded on SAMPVOL.

```
*SYS011=  only code this if you will use L77A local BTAM 3270
*SYS012=  only code this if you will use L77B local BTAM 3270
*SYS013=  only code this if you will use L86O local BTAM 3286
*SYS014=  only code this if you will use remote BTAM line.
```

* Any combination of these may be coded, but at least one must be coded. These are the system numbers and channel unit addresses of the terminals used while running the sample applications.

Job Control Required to Run an Entry Level System

Figure 3.3-1 shows an example of a jobstream used to define permanent ASSGNs and standard label information in the VSE partition designated for CICS/VS use. With the exception of cards 19 through 22 and 31 through 33, the example is part of the output from assembling the DFHJCELS macro. The example contains all of the different categories of JCL definition required to bring up an entry level system, as outlined below.

CICS/VS Library: Cards 1, 2, and 23 in the example define the private core-image library in which CICS/VS nucleus modules, sample applications, and tables reside.

Files Used by CICS/VS File Control: In the example, FILEA is the only file to which CICS/VS File Control refers in response to CICS/VS commands in the sample application programs. In this case, FILEA resides in the VSAM data space (SPACE), and cards 3 through 6 and 24, together with cards 19 through 22, define FILEA and its associated catalogs and data space. DL/I data bases used by CICS/VS File Control would also be defined here.

Files Used by CICS/VS Transient Data Control: MSGUSR, LOGUSR, and DFHNTRA are all examples of transient data files to which the sample programs refer; they are defined by cards 13 through 18, and 24. DFHNTRA contains all of the intrapartition data, but each extrapartition data set is defined separately.

CICS/VS Trace and Dump Files: Cards 7 through 12 define an auxiliary trace file (not opened unless auxiliary trace is set on by a master terminal command) and a switchable dump file. The switchable dump file is always required; it allows CICS/VS operations to continue in the event of a large amount of transaction dump output, when a master terminal command can be used to switch the output to an alternative file while the full one is being printed down by the DFHDUP utility program.

As the DFHDUP and DFHTUP utility programs respectively open the dump and trace files for output, a retention period of zero days should be specified in the DLBLs, and full extent information, including start and size, should be specified in the EXTENT statement.

BTAM Device ASSGNs: Cards 27 through 30 define the terminal configuration for BTAM devices. With VTAM, the assignments would occur in the VTAM partition.

Note: Logical units SYS009 and SYS010 are reserved for trace and dump data sets respectively. SYS011 through SYS014 are used by the sample application table definitions.

```
                                                              Card Number
// DLBL IJSYSCL,'CICS.PCIL'                                          1
// EXTENT SYSCLB                                                     2
*
// DLBL SPACE,'SAMPLE.TEST.SPACE',0,VSAM                             3
// EXTENT SYS008,SAMPLE,,,2,1200                                     4
*
// DLBL FILEA,'SAMPLE.TEST.FILEA',0,VSAM                             5
// EXTENT SYS008,SAMPLE                                              6
*
// DLBL DFHNTRA,'CICS.INTRA',0,VSAM                                 17
// EXTENT SYS008,SAMPLE                                             18
*
* DFHDMPA IS FOR :- DUMP DATA SET WHEN DUMPDS=A
// DLBL DFHDMPA,'CICS.DUMPA',0,SD,CISIZE=2048                        7
// EXTENT SYS010,SAMPLE,,,1202,2000                                  8
*
* DFHDMPB IS FOR :- DUMP DATA SET WHEN DUMPDS=B
// DLBL DFHDMPB,'CICS.DUMPB',0,SD,CISIZE=2048                        9
// EXTENT SYS010,SAMPLE,,,3202,2000                                 10
*
// DLBL DFHAUXT,'CICS.AUXTRACE',0 AUXILIARY TRACE DATA SET          11
// EXTENT SYS009,SAMPLE,,,5202,2000                                 12
*
* MSGUSR IS FOR :- PL/I MESSAGES, PL/I DUMPS, AND
*               :- CSSL SHUT-DOWN STATISTICS
*               :- CEMT OPERATOR STATISTICS
*               :- CSML SIGN-OFF STATISTICS
*               :- CSTL TERMINAL I/O STATS
// DLBL MSGUSR,'CICS.MSGUSR',0 SAMPLE EXTRA TRANS.DS                13
// EXTENT SYS008,SAMPLE,,,7202,400                                  14
*
* LOGUSR KEEPS UPDATE LOG FOR SAMPLE APPLICATION
// DLBL LOGUSR,'CICS.LOGUSR',0 FILEA UPDATE LOG                     15
// EXTENT SYS008,SAMPLE,,,7602,400                                  16
*                                                    ┌──────┐
* DLBL and EXTENT information supplied by the user...│      │
*                                                    │      │
// DLBL IJSYSCT,'VSAM.MASTER.CATALOG',,VSAM          │      │      19
// EXTENT SYSCAT,SYSRES                              │      │      20
// DLBL IJSYSUC,'CICS.ELS.UCAT',,VSAM                │      │      21
// EXTENT SYS008,SAMPLE                              │      │      22
*                                                    └──────┘
* C.I.C.S. PERMANENT ASSGNS
*
ASSGN SYSCLB,DISK,VOL=SAMPLE,SHR                                    23
ASSGN SYS008,DISK,VOL=SAMPLE,SHR SAMPLE APPL. PACK                  24
ASSGN SYS009,DISK,VOL=SAMPLE,SHR AUXILIARY TRACE DATA SET           25
ASSGN SYS010,DISK,VOL=SAMPLE,SHR BOTH DUMP DATA SETS                26
ASSGN SYS011,X'0A1' SCREEN L77A                                     27
ASSGN SYS012,IGN SCREEN L77B                                        28
ASSGN SYS013,X'0A4' PRINTER L860                                    29
ASSGN SYS014,IGN REMOTE                                             30
*                                                    ┌──────┐
* ASSGN information supplied by the user...          │      │
*                                                    │      │
ASSGN SYSPCH,X'01B'                                 │      │      31
ASSGN SYSIPT,X'01A'                                 │      │      32
ASSGN SYSLST,X'011'                                 │      │      33
                                                    └──────┘
```

Figure 3.3-1. Example of job control required to run CICS/DOS/VS-ELS

Estimating DASD Space for CICS/VS Files

Dump Files: Formatted and partition dumps requested implicitly at transaction abnormal termination, or explicitly by master terminal or program commands, can cause a considerable volume of output; therefore a minimum of one million bytes of DASD space is recommended for each of DFHDMPA and DFHDMPB.

Auxiliary Trace File: Each trace entry causes a 20-byte record (as part of a 2040-byte block) to be output. Insufficient DASD space will cause data to be lost; after the operator has been warned, the auxiliary trace file is closed, and the auxiliary trace option is set off.

Transient Data Intrapartition Data Set: A minimum of two tracks is required to hold intrapartition control data for all queues. In addition, variable-length messages with an average of four bytes additional overhead are written, unblocked, when requested by program commands. Insufficient DASD space will cause an excessive number of ERROR conditions to be encountered by application programs unable to write messages to full queues.

EXEC Statement to Bring Up CICS/VS

Phase Name: DFHSIP is invoked by the VSE EXEC statement to bring up CICS/VS.

Partition Size: The SIZE parameter of the VSE EXEC statement must be used to specify the virtual address space in which CICS/VS is to run. The difference between the EXEC size and the ALLOC size is the GETVIS size. Two examples of cases where GETVIS storage is required are:

1. Where VSAM is being used.
2. Where rotational position sensing (RPS) is being used.

The size of the virtual address space specified on the EXEC SIZE parameter depends on:

1. The total size of the CICS/VS nucleus modules selected as a result of function required by initialization options.
2. The total size of the application modules in the user's environment.
3. The subpool size required by both (1) and (2) above; this subpool, or dynamic storage area (DSA), is controlled by DFHSCE, the storage control program for CICS/DOS/VS-ELS.

After the jobstream created by the DFHJCELS macro has been executed, that is, when CICS/VS is running, users can enter the channel unit addresses of the terminals that invoke the sample programs; any of the following transaction identifiers may be entered, where "x" is the initial letter of the programming language being used (for example, AMNU displays the Assembler language transaction identifiers). Note that the xINQ, xADD, xUPD, and xBRW transactions must be entered via the menu screen displayed by xMNU. The xORD and xCOM transactions need both a screen and a printer for correct execution.

xMNU
— display the sample transaction identifiers.

xINQ
— display a file entry

xADD
— add a file entry

xUPD
— update a file entry

xBRW
— browse (VSAM only)

xORD
— order entry

xCOM
— print order entry queue

The first four transactions listed above are provided by the UPDATE sample program, described and listed in Part 2 of this manual, in the appropriate language-specific chapters for Assembler, COBOL, and PL/I. (The VSAM browse transaction is described and listed in Chapter 2.5 for RPG II users. All the sample programs in each of the languages are described and listed in the appropriate CICS/VS Application Programmer's Reference Manual. The program descriptions should be read before any attempt is made to run the sample programs.)

If Release 5 of the PL/I Optimizing Compiler Transient Library is not available on the private core-image library being used, messages will be produced during startup indicating that application programs with names beginning with "IBM" cannot be found. The messages should be ignored if the intention is to run the COBOL, Assembler language or RPG II sample programs.

Running User Applications

The sample programs are supplied complete with the maps and tables needed to run them. To tailor the system to the user's own application programs, suitable maps and tables will have to be generated. As a short cut, it may well be possible to modify the supplied sample programs and maps to fit in with required user applications. In any case, the following list enumerates the operations that will be necessary, and shows where to find the information about them:

1. Prepare the application programs (see Part 2 of this manual).

2. Prepare the tables (see Chapter 3.4).

3. Prepare the maps (see Chapter 3.5).

4. Bring up the system using startup overrides where required (see Chapter 3.6).

Chapter 3.4. Table Generation

CICS/VS is a table-driven system. All information regarding the terminals, files (permanent and temporary), programs, transactions, and operator identifications is contained in these tables. The tables are specified in table generation macros, which conform to assembler language macro syntax.

CICS/VS tables have to be prepared, assembled, link-edited, and cataloged in the core image library. Each table is created separately, and may be recreated at any time prior to CICS/VS initialization. More than one table of each type (identified by unique suffixes) can be maintained and used. This allows the user to maintain special tables for testing, and other tables for normal operation.

CICS/DOS/VS-ELS contains examples of all the tables. They have been provided mainly to support the sample applications. However, the user may be able to use some without change. A summary of the function provided by the tables is given in the following pages. For the complete details, see the source used to generate the tables in the source statement library (see Appendix A).

Organization of this Chapter

The remainder of this chapter consists of general information on the table generation macros and how to assemble them, followed by summaries of the macros in alphabetic order. Each summary includes the syntax definition and an alphabetic list of operands with their descriptions. Where possible, default parameters are indicated by underscoring. An example of the use of the macros is given for each table.

Table Generation Procedures

The following table generation macros need to be coded and assembled:

1. DFHDCT - Destination control table
2. DFHFCT - File control table
3. DFHPCT - Program control table
4. DFHPPT - Processing program table
5. DFHSNT - Sign-on table
6. DFHTCT - Terminal control table

The TYPE=INITIAL and TYPE=FINAL macros are required for each table; they establish the beginning and end of the table. Each table assembly run must be terminated by the assembler END statement.

To assemble tables, the source statement library containing the VSE data management macro instructions is also needed. To link-edit the tables, the relocatable library containing logic modules is needed. (See the System Summary in Appendix A.)

The output of each assembly contains the linkage-editor control statements (PHASE and INCLUDE) required to link-edit the table into the VSE core image library.

The following is an example of the assembly, link-edit, and catalog of CICS/VS control tables, showing the standard sequence of macro types:

```
// JOB CICSTAB
*  CICS/DOS/VS-ELS CONTROL TABLE GENERATION
// OPTION  CATAL,NOXREF,ALIGN
// ASSGN   SYSSLB,DISK,VOL=CICSCS,SHR
// EXEC    ASSEMBLY,SIZE=64K
         PRINT NOGEN
         DFHxxx TYPE=INITIAL,SUFFIX=..
         DFHxxx TYPE=...
         .
         .
         .
         (macros defining table xxx)
         .
         .
         .
         DFHxxx TYPE=FINAL
         END
/*
// EXEC LNKEDT
/&
```

In the above illustration, xxx represents the type of table as listed above.

The tables are named as follows:

Table	Name
Destination control table	DFHDCTyy
File control table	DFHFCTyy
Program control table	DFHPCTyy
Processing program table	DFHPPTyy
Sign-on table	DFHSNT
Terminal control table	DFHTCTyy

The first six characters of the name are standard for each of the tables. Except for the sign-on table, the last two characters (yy) may be specified by the user through the SUFFIX operand to allow several versions of a table to exist; any one or two characters (other than NO, #, and $) are valid. The suffix determines which version of that table is to be loaded during system initialization. Startup override parameters are used to specify which version is used (see Chapter 3.6).

DCT — Destination Control Table

The DCT describes sequential files used for transient data, and associates the files with the appropriate destination names.

```
        DFHDCT  TYPE=INITIAL
                [,SUFFIX=xx]
                [,DEVICE={2314|3330|3340|3350}]

        DFHDCT  TYPE=SDSCI
                ,DEVICE=device
                ,DSCNAME=name
                [,BLKSIZE=length]
                [,BUFNO={1|2}]
                [,DEVADDR=symbolic address]
                [,ERROPT={IGNORE|SKIP}]
                [,RECFORM={FIXUNB|FIXBLK|VARUNB|VARBLK}]
                [,RECSIZE=length]
                [,REWIND={UNLOAD|NORWD}]
                [,TYPEFLE={INPUT|OUTPUT}]
                [,TPMARK=NO]

        DFHDCT  TYPE=EXTRA
                ,DESTID=name
                ,DSCNAME=name

        DFHDCT  TYPE=INDIRECT
                ,DESTID=name
                ,INDDEST=name

        DFHDCT  TYPE=INTRA
                ,DESTID=name
                [,DESTFAC={TERMINAL|FILE}]
                [,TRANSID=name]
                [,TRIGLEV={1|number}]

        DFHDCT  TYPE=FINAL
```

The DFHDCT macro instruction types are:

1. DFHDCT TYPE=INITIAL
2. DFHDCT TYPE=SDSCI: data set control information
3. DFHDCT TYPE=EXTRA: extrapartition destinations
4. DFHDCT TYPE=INDIRECT: indirect data destinations
5. DFHDCT TYPE=INTRA: intrapartition destinations
6. DFHDCT TYPE=FINAL

DFHDCT TYPE=SDSCI — Data Set Control Information

This macro generates the data set control block (DTF) necessary to process a single transient data file. It is needed only for extrapartition transient data, and a DFHDCT TYPE=EXTRA macro instruction must be associated with it by specifying the same file name in the DSCNAME operand. All DFHDCT TYPE=SDSCI macros must be issued immediately after the DFHDCT TYPE=INITIAL macro and before any DFHDCT TYPE=EXTRA, DFHDCT TYPE=INTRA, or DFHDCT TYPE=INDIRECT macros.

DFHDCT TYPE=EXTRA — Extrapartition Destinations

Destinations external to the system are specified using the DFHDCT TYPE=EXTRA macro instruction. This macro instruction must be generated once for each extrapartition destination.

Note: The DFHDCT TYPE=INDIRECT macro instruction should be used when multiple extrapartition destinations share the same data set.

DFHDCT TYPE=INDIRECT — Indirect Data Destinations

The DFHDCT TYPE=INDIRECT macro instruction is used for data that is to be routed to a file already described by a DFHDCT TYPE=EXTRA or TYPE=INTRA macro. A separate DTF is not required.

DFHDCT TYPE=INTRA — Intrapartition Destinations

The DFHDCT macro must be coded once for each intrapartition destination. If automatic transaction initiation is required, this macro sets the trigger level for the destination and identifies the transaction to be initiated.

DFHDCT Operands

BLKSIZE=length
: specifies the length, in bytes, of the block (the maximum length for variable-length records including four bytes for the LLØØ block length prefix). For DASD output, add eight bytes for the record count field.

  Example: BLKSIZE=128 (120-byte block + 8 for DASD count field)

BUFNO=1|2
: specifies the number of buffers (one or two) to be provided. Two should be specified for high-usage files or printers.

DESTFAC=
: specifies the type of destination that the queue represents.

TERMINAL
— the transient data destination is to be associated with a specific terminal. If automatic transaction initiation is used, the specified terminal must be available before the transaction can be initiated.

FILE
— the transient data destination is to be used as a file not associated with a particular terminal. Automatic transaction initiation does not require a terminal to be available.

DESTID=name
specifies the symbolic name of the destination. The symbolic name is the same as that used in transient data commands (such as WRITEQ TD) that specify the destination.

Any destination identification (DESTID) of more than four characters is truncated on the right. Names starting with the letter "C" are reserved for CICS/VS.

If the ultimate destination of intrapartition data is a terminal and if automatic transaction initiation is associated with the destination, the name specified in the DESTID operand must be the same as the name specified in the TRMIDNT operand of the DFHTCT TYPE=GPENTRY macro. It may be convenient to use the same naming convention for terminal destinations and data set destinations, regardless of whether automatic transaction initiation is requested.

Example: DESTID=CSMT

DEVADDR=symbolic address
specifies the symbolic unit address. This operand is not required for disk data sets when the symbolic address is provided through the VSE EXTENT statement. It is required for tape and unit record devices.

Example: DEVADDR=SYS041

DEVICE=device
specifies the type of input/output device.

Type=INITIAL
specifies the device to be used for intrapartition data sets. Valid device types are: 2314 (the default), 3330, 3340, and 3350. If the data is VSAM intrapartition transient data, the default device is used regardless of the device specified.

Type=SDSCI
specifies the device to be used for extrapartition data sets. Valid device types are: 1403, 1404, 1443, 1445, 2314, 3203, 3211, 3330, 3340, 3350, FBA, 5203, and TAPE.

DSCNAME=name
specifies the one- to seven-character file name used in the DLBL statement. The name should not start with the letters "DFH", which are reserved for use by CICS/VS. TYPE=EXTRA macros are associated with TYPE=SDSCI macros by specifying the same file name in the DSCNAME operand.

ERROPT=
specifies the error option to be performed.

IGNORE
— accept the block that caused the error.

SKIP
— skip the block that caused the error.

INDDEST=name
identifies an intrapartition or extrapartition destination. This identification must be the same as the DESTID of the actual destination. If the name specified is not defined in the DCT, an assembly error will result.

Example: INDDEST=CSMT

RECFORM=
specifies the record format.

FIXUNB
— fixed unblocked records.

FIXBLK
— fixed blocked records.

VARUNB
— variable unblocked records.

VARBLK
— variable blocked records.

RECSIZE=length
specifies the record length (the maximum length for variable length records including four bytes for the LLØØ record length prefix). This parameter is needed only if RECFORM=FIXBLK or RECFORM=VARBLK is specified.

REWIND=
specifies the disposition of a tape data set.

UNLOAD
— rewind and unload the current volume

NORWD
— do not rewind

TPMARK=NO
specifies that no tapemark is to be written at the beginning of the file.

TRANSID=name
— identifies the transaction that is to be automatically initiated when the trigger level is reached. The purpose of such initiated transactions is to read records from the destination. If this operand is omitted, or if TRIGLEV=0 is specified, some other means must be employed to schedule transactions to read records from the destinations.

TRIGLEV=number
specifies the number of data records (the trigger level) to be accumulated for a destination before automatically requesting the creation of a task to process these records. The TRIGLEV default value 1 applies when TRANSID is specified without TRIGLEV. The maximum allowed is 32767.

If the DESTFAC operand specifies TERMINAL, the transaction will not be initiated until the associated terminal is available.

During operation, the master terminal operator can use the CEMT transaction to change the trigger level. If the trigger level is reduced to a number equal to or less than the number of records accumulated so far, the task will be initiated when the next record is written to the destination.

TYPEFLE=
specifies the type of data set. The default is TYPEFLE=INPUT.

INPUT
—input data set.

OUTPUT
—output data set.

An extrapartition SDSCI can be either input or output, but not both.

Example of Destination Control Table

```
 // JOB DFHDCT
 // OPTION CATAL,NOXREF,ALIGN
 // ASSGN SYSSLB,DISK,VOL=CICSCS,SHR
 // EXEC ASSEMBLY,SIZE=64K
         PRINT NOGEN
         DFHDCT TYPE=INITIAL,SUFFIX=3E,DEVICE=3330
*
         DFHDCT TYPE=SDSCI,DSCNAME=MSGUSR,BLKSIZE=128,RECFORM=VARUNB,   *
               TYPEFLE=OUTPUT,BUFNO=1,DEVICE=3330
         DFHDCT TYPE=SDSCI,DSCNAME=LOGUSR,BLKSIZE=100,RECFORM=FIXUNB,  *
               TYPEFLE=OUTPUT,BUFNO=1,DEVICE=3330
*
         DFHDCT TYPE=EXTRA,DESTID=SLOG,DSCNAME=MSGUSR
*
*        THE NEXT FOUR ENTRIES ARE CICS/VS-REQUIRED
*
         DFHDCT TYPE=INDIRECT,DESTID=CSMT,INDDEST=SLOG
         DFHDCT TYPE=INDIRECT,DESTID=CSTL,INDDEST=SLOG
         DFHDCT TYPE=INDIRECT,DESTID=CSSL,INDDEST=SLOG
         DFHDCT TYPE=INDIRECT,DESTID=CSML,INDDEST=SLOG
*
         DFHDCT TYPE=EXTRA,DESTID=ULOG,DSCNAME=LOGUSR
         DFHDCT TYPE=INDIRECT,DESTID=LOGA,INDDEST=ULOG
*
         DFHDCT TYPE=INTRA,DESTID=L860,TRIGLEV=30,TRANSID=ACOM,         *
             DESTFAC=TERMINAL
*
         DFHDCT TYPE=FINAL
         END
 /*
 // EXEC LNKEDT
 /&
```

FCT — File Control Table

The file control table identifies all of the user files to be included in the system. ELS supports ISAM and VSAM files.

```
┌──────┬──────┬──────────────────────────────────────────────────────────┐
│      │DFHFCT│ TYPE=INITIAL                                             │
│      │      │ [,SUFFIX=xx]                                             │
│      │      │                                                          │
│      │DFHFCT│ TYPE=DATASET                                             │
│      │      │ ,DATASET=name                                            │
│      │      │ ,ACCMETH=({ISAM|DL/I|VSAM[,KSDS|ESDS|RRDS]})            │
│      │      │ ,SERVREQ=(request[,request],...)                         │
│      │      │ [,RECFORM=([{UNDEFINED|VARIABLE|FIXED}]                  │
│      │      │                [{,BLOCKED|UNBLOCKED}])]                  │
│      │      │                                                          │
│      │      │ ISAM only                                                │
│      │      │                                                          │
│      │      │ ,EXTENT=number                                           │
│      │      │ [,BLKKEYL=length]                                        │
│      │      │ [,BLKSIZE=length]                                        │
│      │      │ [,CYLOFL=number]                                         │
│      │      │ [,DEVICE=(device[,device])                               │
│      │      │ [,INDAREA=symbolic name]                                 │
│      │      │ [,INDSIZE=length]                                        │
│      │      │ [,IOSIZE=length]                                         │
│      │      │ [,LRECL=length]                                          │
│      │      │ [,MSTIND=YES]                                            │
│      │      │ [,NRECDS=number]                                         │
│      │      │ [,RKP=number]                                            │
│      │      │ [,VERIFY=YES]                                            │
│      │      │                                                          │
│      │      │ VSAM only                                                │
│      │      │                                                          │
│      │      │ [,BUFND=number]                                          │
│      │      │ [,BUFNI=number]                                          │
│      │      │ [,BUFSP=number]                                          │
│      │      │ [,PASSWD=password]                                       │
│      │      │ [,STRNO=number]                                          │
│      │      │                                                          │
│      │DFHFCT│ TYPE=SHRCTL                                              │
│      │      │ [,BUFFERS=(size(count),...)]                             │
│      │      │ [,KEYLEN=number]                                         │
│      │      │ [,RSCLMT=number]                                         │
│      │      │ [,STRNO=number]                                          │
│      │      │                                                          │
│      │DFHFCT│ TYPE=FINAL                                               │
│      │      │                                                          │
│      │DFHFCT│ TYPE=LOGICMOD                                            │
│      │      │ [,RPS=SVA]                                               │
└──────┴──────┴──────────────────────────────────────────────────────────┘
```

The DFHFCT macro types are as follows:

- DFHFCT TYPE=INITIAL
- DFHFCT TYPE=DATASET — describes the characteristics of a data set.
- DFHFCT TYPE=FINAL
- DFHFCT TYPE=LOGICMOD — generates an ISAM logic module. If this macro is used, it must be included after DFHFCT TYPE=FINAL.

DFHFCT TYPE=DATASET — Data Set Description

The DFHFCT TYPE=DATASET macro describes the characteristics of a single file. A separate TYPE=DATASET macro must be coded for each file to be included in the system.

If the DL/I DOS/VS facility is to be used, the DFHFCT TYPE=DATASET macro is used to provide information about DL/I data bases. DATASET and ACCMETH are the only operands required for DL/I data bases. Physical characteristics of the DL/I data bases need not be specified.

DFHFCT TYPE=SHRCTL — VSAM Shared Resources Control

The DFHFCT TYPE=SHRCTL macro instruction can be used to control the sharing of VSAM resources by CICS/VS VSAM files. Because both the entry that describes the VSAM data set and the entry that controls the sharing of resources are referred to by the file control program whenever I/O is requested for a data set that is sharing resources, it may be desirable to group all data sets which share resources together in the file control table, along with the entry to control the sharing of resources.

The DFHFCT TYPE=SHRCTL macro should follow the entries for the VSAM data sets that are sharing resources.

If one or more VSAM data sets indicate that they are to share resources and this macro instruction has not been issued prior to the DFHFCT TYPE=FINAL macro instruction, the entry necessary to control the sharing of resources is automatically generated with all values defaulted.

DFHFCT Operands

ACCMETH=
: specifies the access method.

  ISAM
  : Indexed Sequential Access Method.

  DL/I
  : DL/I DOS/VS

  VSAM
  : Virtual Storage Access Method.

    KSDS
    : key-sequenced data set. KSDS is the default when ACCMETH=VSAM.

    ESDS
    : entry-sequenced data set.

    RRDS
    : relative-record data set

BLKKEYL=length
specifies a physical key length.

BLKSIZE=length
specifies the physical block length. For undefined blocks, the length should be the maximum user-defined blocksize plus 8. This operand is not required for VSAM.

BUFFERS=size
is used to override part of the CICS/VS resource calculation. Each pair of values specifies a buffer size and a number of buffers of this size to be allocated. Each buffer size must be a power of 2, at least 512, or if greater than 2048, a multiple of 4096. The number of buffers of each size must be at least 3 and less than 32768. If a given buffer size is not defined and it is required, the next larger buffer size will be used. When this parameter is specified, it overrides all of the buffer requirement calculation. What is specified in this parameter is exactly what will be passed to VSAM when the request is made to build the resource pool. If this parameter is not specified, CICS/VS will determine the buffer sizes required and the maximum number of buffers of each size and allocate the percentage specified or implied via the RSCLMT parameter.

BUFND=number (VSAM only)
specifies the number of buffers to be used for data. The minimum specification is the number of strings plus one (see the STRNO operand).

BUFNI=number (VSAM only)
specifies the number of buffers to be used for the index. The minimum specification is the number of strings specified in the STRNO operand.

BUFSP=number (VSAM only)
specifies the size in bytes of the area to be reserved for buffers for this data set within the CICS/VS partition. If less than the minimum is specified, VSAM will not open the data set. If this operand is not specified, VSAM OPEN will obtain a minimum size area. The minimum size must be large enough to hold one data and one index buffer per string (see the STRNO operand) plus one data buffer.

The BUFSP value should be chosen with great care. While the file is open, this storage space is controlled exclusively by VSAM; it will be used only for buffers and only for the specified file.

CYLOFL=number (ISAM only)
specifies the number of tracks per cylinder reserved for overflow records. CYLOFL=0 is invalid; if no cylinder overflow space is to be reserved the operand should be omitted completely.

DATASET=name
specifies the one- to seven-character file name, which must be the same as the "filename" operand in the CICS/VS startup DLBL statement.

If VSAM alternate index support is used to access a base cluster via an alternate index path, the data set name must be the same as the name of the alternate index path. This is specified on the VSE file name used in the job control statement defining the path. No entry is required in the file control table for the alternate index that is used to access the base data set. The link between the alternate index and the base data set is established when the path is defined using Access Method Services.

Note: The data set name should not start with characters "DFH" or "FCT", both of which strings are reserved.

For a DL/I data base, the DATASET operand must specify the same data base name as was specified by the NAME operand when generating the DBD.

DEVICE= (device[,device]) (ISAM only)
specifies the type of device on which the prime data area resides, followed by that of the high-level index. The applicable devices are 2314, 3330, and 3340. The default is DEVICE=(2314,2314).

EXTENT=number (ISAM only)
specifies the number of extents required for the file. This operand is required if ACCMETH=ISAM. The minimum is two (one for the prime data area and one for the cylinder index).

INDAREA=symbolic name (ISAM only)
specifies the unique symbolic name of a virtual storage resident cylinder index. A storage area within the FCT is generated automatically, to contain all or part of the cylinder index. This operand is required only if the cylinder index is to be processed in dynamic storage.

Example: INDAREA=CYLINDA

INDSIZE=length (ISAM only)
If INDAREA is specified, the size is calculated as follows:

(M + 4) * (keylength + 6).

where
M = number of prime data cylinders, and
keylength = keylength in BLKKEYL operand.

Example: INDSIZE=392 (for 24 prime data cylinders, keylength 8)

IOSIZE=length (ISAM only)
specifies the size of the area to be used when adding records to a file. This operand should be used only when SERVREQ=NEWREC is also specified.

Selection of the size should depend on the number of ISAM ADDs expected. The operand should be omitted for a system with few ISAM ADDs, but a large area should be specified for a file with many ISAM ADDs, or where ISAM ADD response time is critical. A large volume will improve ISAM ADD performance, but will need additional real storage. The size is calculated as follows:

```
size = m (BLKKEYL + BLKSIZE + 40) + 24
```

where m is the number of blocks that may be read or written at one time. The size thus calculated must at least equal (BLKKEYL + BLKSIZE + 74).

As a guideline, the user with limited real storage should select a size of less than 1000 bytes. Other users may increase this amount to that represented by the maximum number of blocks on a track.

Example:
The ISAM file has a moderate number of ADDs, minimal storage, a key length of 8, and a blocksize of 250 bytes. Then

```
size = 3 (8 + 250 + 40) + 24 = 918

IOSIZE=918
```

KEYLEN=number
is used to override part of the CICS/VS resource calculation. It specifies the maximum key length of any of the data sets that are to share resources. If not specified, CICS/VS will determine the maximum key length.

LRECL=length (ISAM only)
specifies the size of logical records in a block. For variable-length records within fixed-length blocks, the specified record length, multiplied by the NRECDS value, must equal the actual block size.

MSTIND=YES (ISAM only)
specifies that a master index is used.

NRECDS=number (ISAM only)
specifies the number of logical records in a block. For variable-length records within fixed-length blocks, the number specified, multiplied by the LRECL parameter, must equal the block size.

Note: NRECDS=1, LRECL=blocksize, is not allowed. The most advantageous specification is NRECDS=n, LRECL=(blocksize/n) where n is some decimal value greater than 1.

PASSWD=password (VSAM only)
specifies a one- to eight-character password for access to the data set. If less than eight characters are specified, the password will be padded to the right with blanks. If this operand is omitted for a password-protected file, the console operator may be asked to provide the appropriate password.

The password must differ from the name in the DATASET operand, and must be unique within the FCT.

RECFORM=
describes the record format. The default is UNDEFINED for ISAM and (VARIABLE,BLOCKED) for VSAM.

FIXED
fixed-length records.

VARIABLE
variable-length records.

UNDEFINED
undefined records.

BLOCKED
blocked records.

UNBLOCKED
unblocked records.

Notes:

1. BLKSIZE must include an additional eight bytes for the count field when NEWREC is specified for undefined records.

2. BLOCKED or UNBLOCKED must be specified for all ISAM files of FIXED or VARIABLE format.

3. UNBLOCKED indicates ISAM compatibility for a VSAM file.

RKP=number (ISAM only)
specifies the starting position of the key field in the record, relative to the beginning of the record (position one). For variable-length records, the number must include the four-byte LLbb field at the beginning of each logical record. RKP is required for data sets with embedded keys.

For unblocked files with RKP, ISAM will issue the following MNOTE: "0, KEYLOC INVALID, PARAMETER IGNORED".

Example: RKP=6 (key starting in position 2 of variable-length record)

RPS=SVA
specifies that the logic module to be generated will use Rotational Position Sensing. For further information on this option, see the CICS/VS System Programmer's Reference Manual.

RSCLMT=number

CICS/VS will calculate the maximum amount of resources required by the VSAM data sets that are to share resources. Because these resources are to be shared, some percentage of this maximum amount of resources must be allocated. This parameter is used to specify the percentage of the maximum amount of VSAM resources to be allocated. If this parameter is omitted, 50 percent of the maximum amount of resources will be allocated. If both the STRNO and BUFFERS parameters are specified, RSCLMT will have no effect.

SERVREQ=

defines the types of service request to be allowed for the file.

GET
: records may be read.

PUT
: records may be written.

UPDATE
: records may be updated. If UPDATE is specified, both GET and PUT are implied, and need not be specified.

NEWREC
: records may be added. NEWREC implies that PUT was also specified.

BROWSE (VSAM only)
: records may be sequentially retrieved.

DELETE (VSAM key-sequenced files only)
: records may be deleted from this data set. DELETE implies that UPDATE was specified.

SHARE (VSAM only)
: this file is to share resources. This service cannot be requested for a path, or for the base data set of an alternate indexing structure in which there is an upgrade set.

Note: If any output service request option is to be added dynamically through the CEMT transaction, at least one output option (for example, SERVREQ=PUT) must be specified at assembly time. Similarly, for input options to be added with CEMT, at least one input option must have been specified in SERVREQ.

Example: SERVREQ=(GET,PUT,NEWREC)

STRNO=number (VSAM only)

for TYPE=DATASET, specifies a limit to the number of requests that can be processed concurrently for a VSAM file. CICS/VS will automatically queue any requests received while the number of concurrent requests is at the limit. CICS/VS will accumulate statistics that will help the system programmer find the optimum STRNO value. This operand is required for VSAM files, there being no default value.

For TYPE=SHRCTL, STRNO is used to override part of the CICS/VS resource calculation. It specifies the total number of strings to be shared among the data sets that are to share resources. The value must be at least one and not more than 255. If a number is not specified for STRNO, CICS/VS will determine the maximum number of strings and allocate the percentage specified or implied in the RSCLMT parameter.

VERIFY=YES (ISAM only)
specifies that the parity of disk records is to be checked after they are written.

Examples of File Control Table

Example 1 (ISAM):

```
// JOB DFHFCT
// OPTION CATAL,NOXREF,ALIGN
// ASSGN SYSSLB,DISK,VOL=CICSCS,SHR
// EXEC ASSEMBLY,SIZE=64K
        PRINT NOGEN
        DFHFCT TYPE=INITIAL,SUFFIX=3E
        DFHFCT TYPE=DATASET,DATASET=FILEA,ACCMETH=ISAM,                *
              SERVREQ=(GET,NEWREC,UPDATE),BLKSIZE=160,                 *
              RECFORM=(FIXED,BLOCKED),VERIFY=YES,LRECL=80,NRECDS=2,    *
              IOSIZE=240,CYLOFL=3,                                     *
              BLKKEYL=6,RKP=2,DEVICE=(3330,3330),EXTENT=2
        DFHFCT TYPE=FINAL
        DFHFCT TYPE=LOGICMOD
        END
/*
// EXEC LNKEDT
/&
```

Example 2 (VSAM):

```
// JOB DFHFCT
// OPTION CATAL,NOXREF,ALIGN
// ASSGN SYSSLB,DISK,VOL=CICSCS,SHR
// EXEC ASSEMBLY,SIZE=64K
        DFHFCT TYPE=INITIAL,SUFFIX=1E
        DFHFCT TYPE=DATASET,DATASET=FILEB,ACCMETH=(VSAM,KSDS),         *
              SERVREQ=(GET,UPDATE,NEWREC,BROWSE,DELETE),               *
              RECFORM=(FIXED,BLOCKED),                                 *
              BUFSP=4096,                                              *
              BUFNI=4,                                                 *
              BUFNO=4,                                                 *
              STRNO=3
        DFHFCT TYPE=FINAL
        END
/*
// EXEC LNKEDT
/&
```

PCT — Program Control Table

The PCT identifies all valid transaction codes, and establishes the criteria under which the transaction is to be processed.

```
┌──────┬────────┬──────────────────────────────────────────────────────────┐
│      │DFHPCT  │ TYPE=INITIAL                                             │
│      │        │ ,CICS=ELS                                                │
│      │        │ [,SUFFIX=xx]                                             │
│      │        │ [,TRANSEC=([MASTER(nn)][,SVR(nn)][,FE(nn)][,EDF(nn)])]  │
│      │        │           [,INTERPRETER(nn)]                             │
│      │        │                                                          │
│      │DFHPCT  │ TYPE=ENTRY                                               │
│      │        │                                                          │
│      │        │ ,PROGRAM=name                                            │
│      │        │ {,TASKREQ=xxxx|,TRANSID=transaction id}                  │
│      │        │ [,SCRNSZE={DEFAULT|ALTERNATE}]                           │
│      │        │ [,TRANSEC={1|decimal value}]                             │
│      │        │ [,TWASIZE={0|decimal value}]                             │
│      │        │                                                          │
│      │DFHPCT  │ TYPE=GROUP                                               │
│      │        │ ,FN=(function[,...],...)                                 │
│      │        │                                                          │
│      │DFHPCT  │ TYPE=FINAL                                               │
└──────┴────────┴──────────────────────────────────────────────────────────┘
```

The types of DFHPCT macros are as follows:

- DFHPCT TYPE=INITIAL
- DFHPCT TYPE=ENTRY, which specifies the transaction control information
- DFHPCT TYPE=GROUP, which allows the system programmer to code a single keyword, instead of a list of program names, for each CICS/VS supplied function to be included in the system.
- DFHPCT TYPE=FINAL

DFHPCT TYPE=INITIAL — Operands

SUFFIX=xx
specifies a one- or two-character alphameric suffix for the program control table being assemb;ed. This suffix, if specified, is appended to the standard module name (DFHPCT) and is used to name the module on the linkage editor output library. If this operand is omitted, a suffix is not provided.

TRANSEC=([MASTER(nn)][,SVR(nn)][,FE(nn)][,EDF(nn)][,INTERPRETER(nn)])
allows the user to set the transaction security key value for transactions generated by the DFHPCT TYPE=GROUP macro for the following functions:

MASTER — master terminal facility (CEMT)
SVR — supervisor functions (CEST)
FE — field engineering terminal test facility (CSFE transaction) and facility error recognition system (CSFR transaction)
EDF — execution (command level) diagnostic facility
INTERPRETER — command interpreter (CECI)

Each value specified must be in the range 1 through 24

DFHPCT TYPE=ENTRY — Transaction Control Information

The DFHPCT TYPE=ENTRY macro provides information about the transaction. One TYPE=ENTRY macro must be coded for each user transaction.

Pre-defined entries in a GROUP macro (see next section) may be overridden in a DFHPCT TYPE=ENTRY macro coded before the GROUP macro. Otherwise, GROUP and ENTRY macros can be mixed in any order. The "groupable" entries will not be generated twice in an assembly.

DFHPCT TYPE=GROUP — Operands

CICS/VS provides certain transactions, some of which can be invoked directly from the terminal, the rest being used internally by CICS/VS.

Entries for CICS/VS supplied transactions must be made in the PCT. Entries are generated using the DFHPCT TYPE=GROUP macro, in the form DFHPCT TYPE=GROUP,FN=function[,...].

FN=function
indicates the functions for which entries in the PCT must be generated. Any number of functions from the list below can be specified in one DFHPCT TYPE=GROUP macro.

STANDARD
This is always required for ELS internal use. It provides the transaction identifications that are required in the majority of CICS/VS systems. The transaction identifications generated are:

- CSTT — supervisor statistics program
- CSAC — abnormal condition program
- CSTE — terminal abnormal condition program

CONSOLE
This is optional. It permits operators to send messages to the system console. It generates TRANSID=CWTO for processing unit console support in CICS/DOS/VS.

EDF
Generates TRANSID=CEDF for the execution diagnostic facility (EDF).

FE
This is required if the CSFE transaction is to be used. It generates TRANSID=CSFE for the FE terminal test facility, and TRANSID=CSFR for the Facility Error Recognition System

HARDCOPY
This is required unless the PRINT startup override specifies NO. It generates TRANSID=CSPP for the 3270 print support function (BTAM and VTAM).

INTERPRETER
This is required if the command interpreter is to be used. It generates the following transaction identifier:

- CECS – command interpreter, command syntax checker

OPERATORS
This is always required for the master terminal transaction. It provides the following transaction identifications for the master terminal facility:

- CEMT – master terminal functions
- CEST – supervisor terminal functions
- CEOT – terminal operator functions

SIGNON
This must be specified if the sign–on facility is to be used. It generates the transaction identifications associated with the sign–on program. The transaction identifications generated are:

- CSSN – sign–on
- CSSF – sign–off

TIME
This is always required for ELS internal use. It generates TRANSID=CSTA for the time–of–day adjustment program.

VTAM
This is required if VTAM is being used (VTAM=YES was startup override). It generates TRANSID=CSNE for the VTAM node abnormal condition program and CSGM for the good morning sign–on message.

VTAMPRT
This is required if VTAM is being used. It provides the following transaction identifications associated with the VTAM 3270 print function: CSCY, CSPK, and CSRK.

PROGRAM=name
specifies the name of the program that processes this transaction. This program must also be defined in the PPT.

SCRNSZE=
selects one of two screen sizes (defined in the DFHTCT TYPE=GPENTRY macro) to be used for this transaction. This selection operates only when the transaction is attached to a terminal entry that is able to switch screen sizes. This operand also selects the buffer size for printers using a 3270 data stream.

If SCRNSZE=ALTERNATE is specified for the transaction, and the TCT entry for the terminal to which the transaction is attached has the alternate screen size facility, the transaction will run in alternate screen size mode; that is, the screen size used will be that specified in the ALTSCRN parameter of the TCT for the terminal.

If SCRNSZE=DEFAULT is specified for the transaction, and the TCT entry for the terminal to which the transaction is attached has the alternate screen size facility, the transaction will run in default screen size mode; that is, the screen size used will be that specified in the TRMMODL parameter of the TCT for the terminal.

If the TCT entry for the terminal to which the transaction is attached does not have the alternate screen size facility, any SCRNSZE option for the transaction is ignored.

TASKREQ=xxx
: specifies that a terminal operator may use a program access (PA) key, program function (PF) key, light pen attention (LPA) field, operator identification card reader (OPID), or magnetic stripe reader (MSRE) to initiate this task. Valid parameters are: PA1 through PA3, PF1 through PF24, LPA, OPID, and MSRE.

  TASKREQ and TRANSID are mutually exclusive. Only one may be specified per PCT entry.

  If 3270 printer support is used, the PA key associated with the PRINT startup override parameter must not be specified. (See "Startup Override Parameters," in Chapter 3.6.)

TRANSEC=decimal value
: defines the transaction security key. The range of the value is 1 (the default) through 24. No operator without the specified security key can initiate the transaction. See the SCTYKEY operand in the sign-on table.

  When a transaction is automatically initiated, the operator signed on to the terminal must have the same security key as the transaction. To ensure that all automatically initiated transactions can be initiated without a security violation, either the security key of the transaction should be "1" or the operator signed on to the terminal should have a maximum security key.

  Any automatically-initiated transaction associated with a non-operator terminal, such as a 3284 printer, must have the security key "1".

TRANSID=transaction id
: specifies a one- to four-character transaction identifier. The transaction identifier entered by the operator must exactly match that specified in this macro, including upper and lowercase characters. (However, it is possible to specify that lowercase characters keyed in at a terminal are converted to uppercase, using the TRMFEAT=U operand of the DFHTCT TYPE=GPENTRY macro.)

  Note: Transaction identifiers (TRANSIDs) for supplied CICS/VS transactions begin with the letter "C". The TRANSID for a user-written transaction can also begin with "C", but this is not recommended, because there is a danger of duplication of user-defined identifiers by future supplied CICS/VS TRANSIDs.

TWASIZE=decimal value
specifies the size of the transaction work area to be acquired for this transaction. The default is TWASIZE=0. The maximum size is 32767 minus the length of the task control area.

Example of Program Control Table

```
// JOB DFHPCT
// OPTION CATAL,NOXREF,ALIGN
// ASSGN SYSSLB,DISK,VOL=CICSCS,SHR
// EXEC ASSEMBLY,SIZE=64K
        PRINT NOGEN
*
        DFHPCT TYPE=INITIAL,CICS=ELS,SUFFIX=VE,EXTSEC=YES,              *
                     TRANSEC=(MASTER(24),FE(5))
*
* APPLICATION ENTRIES
*
        DFHPCT TYPE=ENTRY,TASKREQ=PA1,PROGRAM=XDFHINST
        DFHPCT TYPE=ENTRY,TRANSID=CMNU,PROGRAM=XDFHINST
        DFHPCT TYPE=ENTRY,TRANSID=CINQ,PROGRAM=XDFHCALL
        DFHPCT TYPE=ENTRY,TRANSID=CADD,PROGRAM=XDFHCALL
        DFHPCT TYPE=ENTRY,TRANSID=CUPD,PROGRAM=XDFHCALL
        DFHPCT TYPE=ENTRY,TRANSID=CBRW,PROGRAM=XDFHBRWS
        DFHPCT TYPE=ENTRY,TRANSID=CORD,PROGRAM=XDFHOREN
        DFHPCT TYPE=ENTRY,TRANSID=CCOM,PROGRAM=XDFHCCOM
*
* IF EDF IS TO BE INVOKED BY A PF KEY INSTEAD OF BY TRANSID CEDF
*
        DFHPCT TYPE=ENTRY,TASKREQ=PF12,PROGRAM=DFHEDFP,TRANSEC=8

*
* GROUP ENTRY
*
        DFHPCT TYPE=GROUP,FN=(FE,OPERATORS,CONSOLE,                     *
              HARDCOPY,SIGNON,TIME,STANDARD)
*
        DFHPCT TYPE=FINAL
        END
/*
// EXEC LNKEDT
/&
```

PPT — Processing Program Table

This table identifies all programs and physical maps to CICS/VS, in the order that they are loaded.

```
┌──────┬────────┬──────────────────────────────────────────────────────┐
│      │DFHPPT  │ TYPE=INITIAL                                         │
│      │        │ ,CICS=ELS                                            │
│      │        │ [,SUFFIX=xx]                                         │
│      │        │                                                      │
│      │DFHPPT  │ TYPE=ENTRY                                           │
│      │        │ ,PROGRAM=name                                        │
│      │        │ [,PGMLANG={ASSEMBLER|COBOL|PL/I|RPG}]                │
│      │        │ [,RELOAD={NO|YES}]                                   │
│      │        │                                                      │
│      │DFHPPT  │ TYPE=GROUP                                           │
│      │        │ ,FN=(function [,function]...)                        │
│      │        │                                                      │
│      │DFHPPT  │ TYPE=FINAL                                           │
└──────┴────────┴──────────────────────────────────────────────────────┘
```

The DFHPPT macro types are as follows:

- DFHPPT TYPE=INITIAL
- DFHPPT TYPE=ENTRY — identifies programs and maps
- DFHPPT TYPE=GROUP — simplifies the specification of application program names required for certain CICS/VS facilities
- DFHPPT TYPE=FINAL

DFHPPT TYPE=ENTRY — Processing Program Description

The DFHPPT TYPE=ENTRY macro must be coded once for each program or map. Multipurpose map tables should be entered first in the table. Programs and map tables used together should be grouped together to reduce VSE" paging. To improve performance, PPT entries should be coded by frequency of usage (high activity programs and maps should appear first).

DFHPPT TYPE=GROUP - Required Entries

The optional DFHPPT TYPE=GROUP macro instruction allows the system programmer to specify the application program names that are required when certain CICS/VS facilities are used, on a functional basis instead of having to specify the PROGRAM=name operands for each individual program being generated in the system.

A pre-defined entry in a TYPE=GROUP macro can be overridden in a DFHPPT TYPE=ENTRY macro coded before the TYPE=GROUP macro. Otherwise, TYPE=GROUP and TYPE=ENTRY macros can be mixed in any order. The "groupable" entries will not be generated twice in an assembly.

Entries for some functions are required and others are optional, as follows:

STANDARD
always required.

OPERATORS
always required.

OPENCLSE
always required.

TIME
always required.

CONSOLE
required if terminal operators are to be able to send messages to the console.

EDF
required if the Execution Diagnostic Facility is to be used.

FE
required if the terminal test facility is to be used.

HARDCOPY
required if the PRINT startup override specifies anything other than NO.

PL/I
required if PL/I application programs are to be used.

SIGNON
required if the sign-on facility is to be used.

VTAM and VTAMPRT
required if VTAM is being used.

DFHPPT Operands

FN=function
indicates the generic function name that generates the entries required in the PPT for the associated facility. Any number of options from the list below can be specified in one DFHPPT TYPE=GROUP macro. The options are:

STANDARD
provides the application program names that are required in the majority of CICS/VS systems. The program names generated are:

- DFHACP - abnormal condition program
- DFHSTP - system termination program
- DFHSTLK - intersystem communication link statistics program
- DFHSTKC - supervisor statistics program
- DFHSTPD - Transaction, program, and dump statistics program
- DFHSTTD - data management statistics program
- DFHSTTR - file and terminal statistics program
- DFHTACP - terminal abnormal condition program
- DFHTEP - terminal error program

The programs generated by FN=STANDARD are low-usage programs and should be generated towards the end of the PPT.

CONSOLE
generates PROGRAM=DFHCWTO for console support in CICS/DOS/VS.

EDF
generates the following program names associated with the execution (command level) diagnostic facility (EDF):

- DFHEDFP - EDF control program
- DFHEDFX - EDF task switch program
- DFHEDFD - EDF display program
- DFHEDFM - EDF map set
- DFHEDFF - EDF function description table
- DFHEDFR - EDF response table

FE
generates PROGRAM=DFHFEP for the terminal test facility, and PROGRAM= DFHFELG for the Facility Error Recognition System.

HARDCOPY
generates PROGRAM=DFHP3270 for the 3270 print allocation program (BTAM and VTAM).

INTERPRETER
generates programs DFHECIP, DFHECSP, and DFHEIND for the command interpreter.

OPENCLSE
generates PROGRAM=DFHOCP for the dynamic open/close function.

OPERATORS
generates programs DFHEMTP, DFHESTP, DFHEOTP, DFHEMTD, and DFHEMA through DFHEME for the master terminal function.

PL/I
generates the programs needed to provide PL/I support in CICS/VS.

SIGNON
generates the program names associated with the sign-on program. The programs generated are:

- DFHSFP - sign-off program
- DFHSNP - sign-on program
- DFHSNT - sign-on table

TIME
generates PROGRAM=DFHTAJP for the time adjustment program.

VTAM
generates PROGRAM=DFHZNAC and DFHZNEP for the VTAM node abnormal condition and node error programs and DFHGMM for the VTAM good morning sign-on message program.

VTAMPRT
generates DFHCPY, DFHEXI, DFHPRK, and DFHRKB for the VTAM 3270 terminal control print key function.

PGMLANG=
specifies the language in which the program is written. This operand should be omitted for PPT entries describing BMS maps.

ASSEMBLER
— Assembler program

COBOL
— COBOL program

PL/I
— PL/I program

RPG
— RPG II program

PROGRAM=name
specifies the program name (one through eight characters) or map name (one through seven characters). The name is that under which the module has been cataloged in the VSE core image library.

RELOAD=YES
specifies that a load request is to bring in a fresh copy of a program. This parameter is required for RPG II programs; it is not allowed for other programs.

Example of Processing Program Table

```
// JOB DFHPPT
// OPTION CATAL,NOXREF,ALIGN
// ASSGN SYSSLB,DISK,VOL=CICSCS,SHR
// EXEC ASSEMBLY,SIZE=64K
        PRINT NOGEN
        DFHPPT TYPE=INITIAL,CICS=ELS,SUFFIX=VE
*
* APPLICATION PROGRAMS AND THEIR MAPS
*
        DFHPPT TYPE=ENTRY,PROGRAM=XDFHCMA
        DFHPPT TYPE=ENTRY,PROGRAM=XDFHINST,PGMLANG=COBOL
        DFHPPT TYPE=ENTRY,PROGRAM=XDFHCMB
        DFHPPT TYPE=ENTRY,PROGRAM=XDFHCALL,PGMLANG=COBOL
        DFHPPT TYPE=ENTRY,PROGRAM=XDFHCMC
        DFHPPT TYPE=ENTRY,PROGRAM=XDFHBRWS,PGMLANG=COBOL
        DFHPPT TYPE=ENTRY,PROGRAM=XDFHCMK
        DFHPPT TYPE=ENTRY,PROGRAM=XDFHOREN,PGMLANG=COBOL
        DFHPPT TYPE=ENTRY,PROGRAM=XDFHCML
        DFHPPT TYPE=ENTRY,PROGRAM=XDFHCCOM,PGMLANG=COBOL
*
* REQUIRED CICS/VS ENTRIES
*
        DFHPPT TYPE=GROUP,FN=(EDF,FE,OPERATORS,                    *
        CONSOLE,HARDCOPY,SIGNON,TIME,OPENCLSE,STANDARD)
        DFHPPT TYPE=ENTRY,PROGRAM=DFHPEP
*
```

```
        DFHPPT TYPE=FINAL
        END
/*
// EXEC LNKEDT
/&
```

SNT — Sign-on Table

The sign-on table contains operator security keys and operator priorities, to provide system security.

```
┌──────┬────────┬──────────────────────────────────────────────────┐
│      │DFHSNT  │ TYPE=INITIAL                                     │
│      │        │                                                  │
│      │DFHSNT  │ TYPE=ENTRY                                       │
│      │        │                                                  │
│      │        │ ,OPIDENT=operator identification                 │
│      │        │ ,OPNAME='operator name'                          │
│      │        │ ,PASSWRD=password                                │
│      │        │ [,SCTYKEY={1|(n1[,n2],...)}]                     │
│      │        │                                                  │
│      │DFHSNT  │ TYPE=FINAL                                       │
└──────┴────────┴──────────────────────────────────────────────────┘
```

If the security key for a transaction in the PCT matches one of the operator's security keys from the SNT, the transaction is allowed to continue. Otherwise, the transaction is terminated and the attempted security violation is recorded. This table is required if the operator SIGN ON/SIGN OFF function is desired.

DFHSNT TYPE=ENTRY — Terminal Operator Description

The DFHSNT TYPE=ENTRY macro must be coded once for each terminal operator.

DFHSNT Operands

OPIDENT=operator identification
: specifies the one- to three-character operator identification code assigned by the system administrator to each operator. This code is placed in the appropriate terminal control table terminal entry (TCTTE) when the operator signs on, and is made available to the master terminal when a security violation is detected.

OPNAME='operator name'
: specifies the name of the terminal operator for this table entry. The operator name may be 1 to 20 characters long and must be unique for each entry. On sign-on, the name keyed in by the operator must exactly match that in the sign-on table.

PASSWRD=password
specifies a four-character password to be keyed in by the operator on sign-on. The password keyed in must exactly match that in the sign-on table. The same password may appear in more than one table entry.

SCTYKEY=number
specifies one or more security key values from 1 to 24. The default is SCTYKEY=1. Each key specified allows access to all transactions specified with that key in the program control table (see the TRANSEC operand of the DFHPCT macro).

Example of Sign-on Table

```
// JOB DFHSNT
// OPTION CATAL,NOXREF,ALIGN
// ASSGN SYSSLB,DISK,VOL=CICSCS,SHR
// EXEC ASSEMBLY,SIZE=64K
        PRINT NOGEN
        DFHSNT TYPE=INITIAL
        DFHSNT TYPE=ENTRY,OPNAME='APPLICATION',PASSWRD=USER,        *
          OPIDENT=USR,SCTYKEY=(1,8)
        DFHSNT TYPE=ENTRY,OPNAME='MASTER OPERATOR',PASSWRD=MSTR,    *
          OPIDENT=MOP,SCTYKEY=(1,8,15),OPPRTY=50
        DFHSNT TYPE=FINAL
        END
/*
// EXEC LNKEDT
/&
```

TCT — Terminal Control Table

The terminal control table provides the information needed by the terminal control program (TCP) to control the CICS/VS terminal network.

```
         DFHTCT   TYPE=INITIAL
                  ,CICS=ELS
                  ,MODNAME=(IJLBTM|name)
                  [,ERRATT={NO|([LASTLINE][,INTENSIFY][,{BLUE|RED|PINK|
                  GREEN|TURQUOISE|YELLOW|NEUTRAL}][,{BLINK|
                  REVERSE|underline}])}]
                  [,SUFFIX=xx]
                  
         DFHTCT   TYPE=GPENTRY
                  ,GPTYPE=type
                  [,ALTSCRN=((lines,columns)[,...],...)]
                  [,ALTSFX=(n[,...],...)]
                  [,CUADDR=(nn[,...],...)]
                  [,CUFEAT=(feature[,...],...)]
                  [,CUPOSN=(nn[,...],...)]
                  [,GPTCU={2701|2702|2703|ICA}]
                  [,LINELST=(nnn[,...],...)]
                  [,LINFEAT=(feature[,...],...)]
                  [,LININL=(number[,...],...)]
                  [,TRMADDR=(nn[,...],...)]
                  [,TRMFEAT=([A][D][S][U][P][T][,...],...)]
                  [,TRMIDNT=(xxxx[,...],...)]
                  [,TRMINL=(number[,...],...)]
                  [,TRMMODL=(number,character[,...],...)]
                  [,TRMPOSN=(nn[,...],...)]
                  [,TRMPRTY=(number[,...],...)]
                  [,TRMUAL=(number[,...],...)]
                  
         DFHTCT   TYPE=FINAL
```

Note: This syntax does not apply to VTAM, for which DFHTCT TYPE=TERMINAL macros are needed. For details, please refer to Appendix C.

DFHTCT TYPE=INITIAL — Operands

MODNAME=(IJLBTM|name)
specifies the name of the BTAM logic module that will be link edited with the table.

IJLBTM is the default name generated by BTAM.

name
is a user chosen name and must be the same as the name used when the BTAM logic module was generated. On the PRLB of the distribution volume there are three BTAM logic modules that are suitable for ELS. They are:

BTMODL#	local 3270s
BTMODR#	remote 3270s
BTMODM#	local and remote 3270s

The source used to generate them (and others in the PRLB) is in Z.DFHSTLM. If you wish to generate logic modules

with different BTAM options, you must use the name you chose in MODNAME.

ERRATT={NO|([LASTLINE][,INTENSIFY][,{BLUE|RED|PINK|GREEN|TURQUOISE|YELLOW|NEUTRAL}][,{BLINK|REVERSE|UNDERLINE}])}

indicates the attributes that are to be associated with error messages that are displayed on all 3270 screens in the terminal control table.

NO
indicates that an error message will be displayed at the current cursor position and without any additional attributes.

LASTLINE
indicates that an error message will be displayed starting at the beginning of the line nearest the bottom of the screen such that the message will fit on the screen.

The other values indicate that one or more of the 3270 attributes are to be used when an error messages is displayed. Specification of any attribute implies LASTLINE. Valid attributes are:

for field intensification:
INTENSIFY

for color:
BLUE
RED
PINK
GREEN
TURQUOISE
YELLOW
NEUTRAL

for highlighting:
BLINK
REVERSE
UNDERLINE

Any attributes specified that are not valid for a particular device will be ignored.

DFHTCT TYPE=GPENTRY — Terminal Line Group Definition

The DFHTCT TYPE=GPENTRY macro instruction may be used with the following device types:

- Local 3270
- Remote 3270
- Processing unit console operating as a terminal

The DFHTCT TYPE=GPENTRY macro allows the system programmer to specify terminal types and device characteristics on a line group basis.

One DFHTCT TYPE=GPENTRY must be coded for each line group. If the network consists of only one type of terminal, at least two line groups

should be established so that if one line group is placed out of service, terminals in the other line group can continue in service and can be used to restore the first line group.

The parameters in each operand of this macro, except for GPTYPE and GPTCU, are positional; for example, LINFEAT=(0,,0) indicates that the first and third terminals in this line group have open polling.

DFHTCT Operands

ALTSCRN=((lines,columns),...)
specifies the alternate screen size for each terminal in the line group. If a terminal does not have the alternate screen size facility, the entry for that terminal should be omitted; this operand specifies the screen size to be used when a transaction with SCRNSZE=ALTERNATE is attached to this terminal.

Example: ALTSCRN=((24,80),(32,80),,(12,80))

| ALTSFX=n
| specifies the one-character suffix to be added by BMS to a mapset name if an alternate page size is being used (that is, if PCT specifies SCRNSZE=ALTERNATE). BMS will try to load the mapset with the correct alternate suffix. If it cannot find the mapset, or is unable to load it, BMS will try to load the mapset which has the correct device suffix (for the device specified in the DFHMSD macro TERM operand). If this also fails, BMS will load the unsuffixed map set.

CUADDR=nn
applies to 3270R only, and indicates the control unit address for each remote 3270 control unit within its line group. The range is 0 through 31.

Example: CUADDR=(0,0,1)

which means that the first control unit is the only one on this line; the second control unit is the first one on the second line; the third control unit is the second one on the second line.

CUFEAT=feature
applies to 3270R only, and specifies the features associated with the control unit. C indicates the COPY feature.

Example: CUFEAT=(C,,)

which means that the first control unit has COPY; the second and third do not.

| CUPOSN=nn
| This operand applies to 3270R only.

| The numbers following the CUPOSN Keyword must be arranged in ascending order. They convey two pieces of information:

(1) Each number has a positional value. Thus, the first number in the list represents control unit 1, the second represents control unit 2, and so on. Up to 40 control units can be represented by the CUPOSN operand.

(2) Each number is a pointer to LINELST, indicating the identity of the line to which the control unit is attached. The value of a number can range from 1 to 31.

Example: CUPOSN=(1,2,2,3)

This means that:

C.U. 1 is one line 1
C.U.s 2 and 3 are on line 2
C.U. 4 is on line 3

Thus, if LINELST=(021,022,023), the configuration can be summarized as follows:

There are four control units; the first connected to line SYS021; the second and third to line SYS022; the fourth to SYS023.

GPTCU=
applies to 3270R terminals, and specifies the transmission control unit attached to the processor. The options are: 2701, 2702, 2703, and ICA. 270x must be specified when a 270x control unit is being emulated by a 370x.

GPTYPE=type
specifies the type of terminal in the line group. One type option may be specified in each DFHTCT TYPE=GPENTRY macro. The options are:

- 3270L — Local 3270
- 3270RA — Remote 3270 with ASCII communications code
- 3270RE — Remote 3270 with EBCDIC communications code
- CONSOLE — Processing unit console

LINELST=nnn (nnn of SYSnnn)
is available for all group types except the console device, and specifies the system logical unit number assigned to each terminal in the line group. A maximum of 31 lines may be defined in this list.

Example: LINELST=(021,022) (SYS021, SYS022)

LINFEAT=feature
applies to 3270R terminals only, and specifies the line features. Wrap-around polling is the default; 0 indicates open polling.

Example: LINFEAT=(,0) (Second line open polling)

LININL=number
applies to 3270L terminals, and specifies the terminal input area length in bytes. The number specified should be large enough to handle 80% of the input messages. If the area is too small, CICS/VS issues GETMAIN macro instructions for more storage and rereads the message.

Example: LININL=50

TRMADDR=nn
applies to 3270R terminals, and specifies the relative address of each terminal within its controller. The range is 0 through 31.

Example: TRMADDR=(0,1,2,0,0)

TRMFEAT=feature
indicates the features for each terminal in the line group. The options are:

- A — audible alarm feature
- D — dual case keyboard
- Q — Programmed Symbols
- H — Highlight
- V — Validation
- E — Extended datastream
- C — Color
- S — selector pen feature
- U — upper case translate
- P — printer (required for 3270 printers). CUFEAT must be specified with C, and the remote 3270 control unit must have the COPY feature, if copying is required. If P is specified, no other features apply.

Example: TRMFEAT=(ADSU,,P,A,DU)

TRMIDNT=xxxx
specifies a four-character terminal identification for each terminal in the line group. CNSL must be specified for the console.

Examples: TRMIDNT=(R77A,R77B,R86A,R75A,R75B)
TRMIDNT=CNSL

TRMINL=number

applies to 3270R only, and specifies a terminal input area length in bytes, large enough to handle 80% of input messages. If the number specified is too small, CICS/VS issues GETMAIN macros to obtain more storage. TRMINL=0 must be specified for printer input areas.

Example: TRMINL=(50,50,0,50,50)

TRMMODL=(number character)

indicates the model number of each terminal in the line group. For terminals having the alternate screen size facility, this parameter is used to specify the default screen size for the terminal. For devices without the alternate screen size facility, the options are:

Device	Buffer size	Model number
3277	480	1A
	1920	2A
3284	480	1B
	1920	2B
3286	480	1C
	1920	2C

For devices with the alternate screen size facility:

Device type	Default screen size	Model number
Display	480	1A
	1920	2A
Printer	480	1B
	1920	2B

Example: TRMMODL=(1A,2A,2C,1B,2B)

TRMPOSN=nn

applies to 3270R terminals, and indicates the relative position of the control unit within the list CUPOSN (CUPOSN= 1 through 40) to which each terminal is attached. A maximum of 40 terminals may be defined.

Example: TRMPOSN=(1,1,1,2,3)

If CUPOSN and LINELST are as shown in the examples, this means:

SYS010 has one 3271 with three terminals;
SYS011 has two 3271s, each with one terminal attached.

TRMPRTY=number

specifies the priority to be assigned for each terminal in the group. The task priority is the sum of the terminal, operator, and transaction priorities, and must not exceed 255.

TRMUAL=number
is available for all device types and indicates, for each terminal in the line group, the size of the terminal control table user area (TCTUA) if this area is to be used by application programs. Any information stored in this area is available to all transactions originated by this terminal. The maximum TRMUAL size is 255 bytes; the default is 0.

Example: TRMUAL=(50,50,0,50,50)

Examples of Terminal Control Table

Example 1:

The table below describes a network of two local 3277's and a 3284 printer:

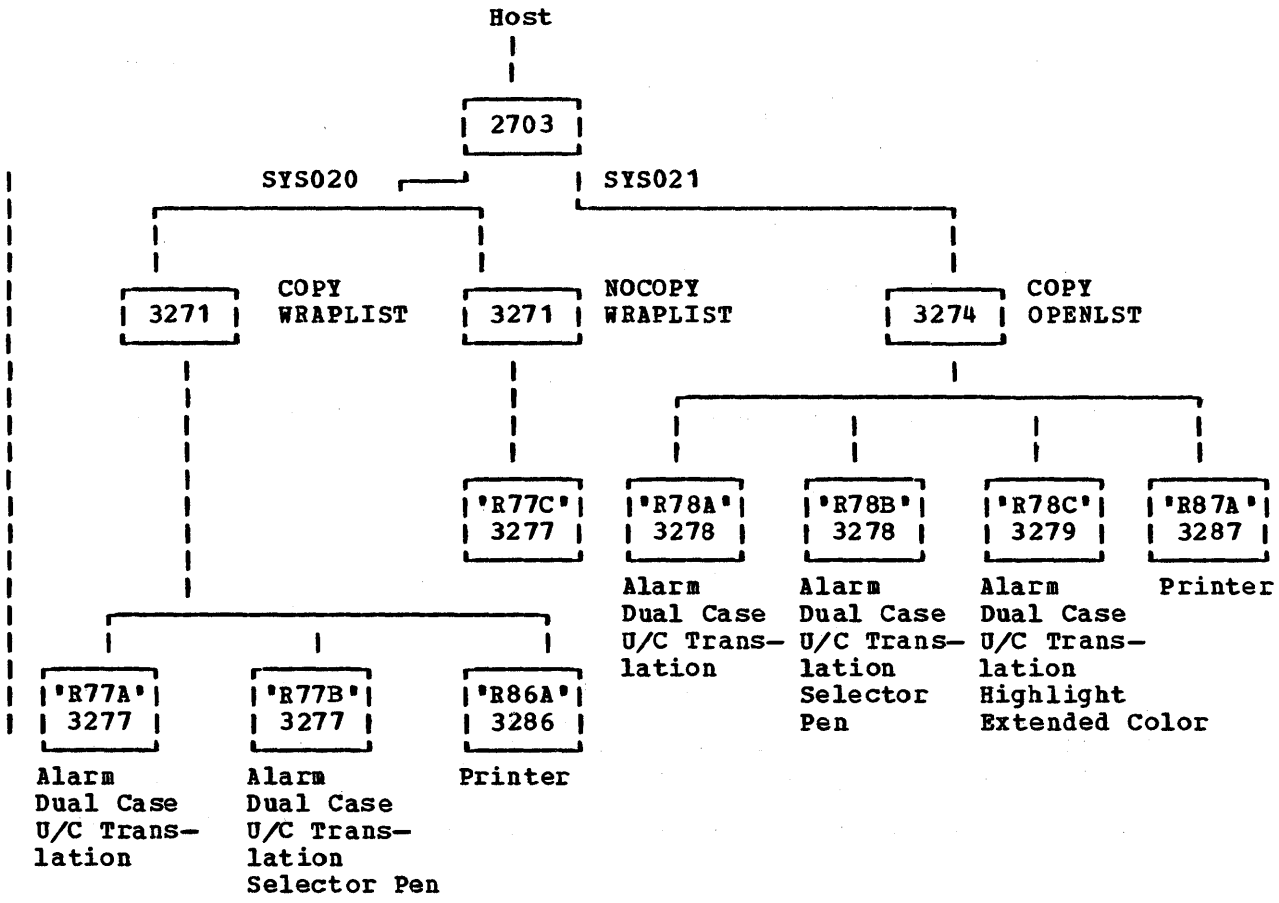
OPERAND	TERMINAL 1	TERMINAL 2	PRINTER
LINELST	010	011	012
TRMMODL	1A	2A	2B
TRMFEAT	ADU	ADSU	P
TRMIDNT	L77A	L77B	L84A
TRMUAL*	0	0	0
TRMPRTY	100	100	100

* TRMUAL defaults to 0, and since all values are zero, this parameter may be omitted.

The following example defines the line group described in the above table:

```
// JOB DFHTCT
// OPTION CATAL,NOXREF,ALIGN
// ASSGN SYSSLB,DISK,VOL=CICSCS,SHR
// EXEC ASSEMBLY,SIZE=256K
         PRINT NOGEN
         DFHTCT TYPE=INITIAL,CICS=ELS,SUFFIX=LC,MODNAME=BTMODL
         DFHTCT TYPE=GPENTRY,GPTYPE=3270L,                          *
               LININL=50,                                           *
               LINELST=(010,011,012),                               *
               TRMMODL=(1A,2A,2B),                                  *
               TRMFEAT=(ADU,ADSU,P),                                *
               TRMIDNT=(L77A,L77B,L84A),                            *
               TRMPRTY=(100,100)
         DFHTCT TYPE=GPENTRY,
               GPTYPE=CONSOLE,     CONSOLE SUPPORT                 *
               TRMIDNT=CNSL,       TERMINAL NAME                   *
               TRMPRTY=50,         TERMINAL PRIORITY               *
               LININL=80           TIOA LENGTH                     *
         DFHTCT TYPE=FINAL
         END
/*
// EXEC LNKEDT
/&
```

Example 2:


Host
2703
SYS020
SYS021
3271
COPY
WRAPLIST
3271
NOCOPY
WRAPLIST
3274
COPY
OPENLST
'R77C'
3277
'R78A'
3278
'R78B'
3278
'R78C'
3279
'R87A'
3287
Alarm
Dual Case
U/C Trans-
lation
Alarm
Dual Case
U/C Trans-
lation
Selector
Pen
Alarm
Dual Case
U/C Trans-
lation
Highlight
Extended Color
Printer
'R77A'
3277
'R77B'
3277
'R86A'
3286
Alarm
Dual Case
U/C Trans-
lation
Alarm
Dual Case
U/C Trans-
lation
Selector Pen
Printer

The following example defines the configuration illustrated above:

```
// JOB DFHTCT
// OPTION CATAL,NOXREF,ALIGN
// ASSGN SYSSLB,DISK,VOL=CICSCS,SHR
// EXEC ASSEMBLY,SIZE=256K
        PRINT NOGEN
        DFHTCT TYPE=INITIAL,
              CICS=ELS,             ELS GENERATION                        *
              MODNAME=BTMODR#       USING 3270 REMOTE BTMOD
        DFHTCT TYPE=GPENTRY,
              GPTYPE=CONSOLE,       CONSOLE SUPPORT                       *
              TRMIDNT=CNSL,         TERMINAL NAME                         *
              TRMPRTY=50,           TERMINAL PRIORITY                     *
              LININL=80             TIOA LENGTH                           *
        DFHTCT TYPE=GPENTRY,
              GPTYPE=3270RE,        REMOTE 3270 EBCDIC                    *
              GPTCU=2703,           CONTROLLER is 2703                    *
              LINELST=(020,021),    ON SYS020 and SYS021                  *
              LINFEAT=(,0),         SYS021 OPEN LIST POLLING              *
              CUADD=(0,1,0),        2 CUS ON LINE 1,1 ON LINE2            *
              CUFEAT=(C,,C),        1ST AND 3RD HAVE COPY                 *
              CUPOSN=(1,1,2),       1 CU ON SYS021,2 ON SYS020            *
              ALTSCRN=(,,,,(12,80),(32,80),(43,80),(43,80)), ALTSCR       *
              TRMADDR=(0,1,2,0,0,1,2,3), 3 TERMS ON 1ST CU, 1 ON 2ND      *
              TRMFEAT=(ADU,ADSU,P,,ADU,ADSU,ADUCH,P), TERMNL FEATURES     *
              TRMIDNT=(R77A,R77B,R86A,R77C,R78A,R78B,R78C,R87A), NAME      *
              TRMINL=(50,50,0,50,100,100,100,0), TIOA SIZES               *
              TRMMODL=(2A,1A,2C,1A,1A,2A,2B,2B), MODEL TYPES              *
              TRMPOSN=(1,1,1,2,3,3,3,3), 3 ON CU 1,1 ON CU2,4 ON CU3      *
              TRMPRTY=(50,50,50,25,100,100,100,100), TRMNL PRIORITIES     *
              TRMUAL=(75,75,75,0,0,0,0,0)           TRMNL TCTUA SIZE      *
        DFHTCT TYPE=FINAL
        END
/*
// EXEC LNKEDT
/&
```

Chapter 3.5. Preparation of Application Programs

This chapter consists of two sections. The first deals with the translation and compilation or assembly of the source programs for each language. The second describes the creation and cataloging of maps.

Program Compilation and Translation

Application programs written to use the command-level interface to CICS/VS or DL/I must be translated by the appropriate CICS/VS Command Translator before being assembled or compiled.

The following sample job streams illustrate how the output from the translator is used as input to the assembler or appropriate compiler. DASD can be used as intermediate storage for translator output only if VSE has been generated to allow system data sets to reside on disk (SYSFIL parameter of FOPT macro).

ASSEMBLER LANGUAGE PROGRAMS

Assembler language programs using the command-level interface must satisfy the following conditions:

- DFHEAI and DFHEAI0 must exist in the relocatable library. (They are supplied in the private relocatable library on the distribution volume.)
- An INCLUDE card for DFHEAI must follow immediately after the PHASE card and before the EXEC ASSEMBLY card.

```
// JOB      jobname
// DLBL     IJSYSPH,'ASSEMBLER TRANSLATION',yy/ddd
// EXTENT   SYSPCH,balance of extent information
ASSGN  SYSPCH,DISK,VOL=volid,SHR
// EXEC     DFHEAP1#,SIZE=64K
    .
    Assembler source deck
    .
/*
CLOSE       SYSPCH,PUNCH
// DLBL     IJSYSIN,'ASSEMBLER TRANSLATION',yy/ddd
// EXTENT   SYSIPT
ASSGN  SYSIPT,DISK,VOL=volid,SHR
// OPTION   SYM,ERRS,NODECK,CATAL
    PHASE    phase-name,*
    INCLUDE DFHEAI
// EXEC     ASSEMBLY,SIZE=64K
// EXEC     LNKEDT
/&
// JOB
// CLOSE
/&
```

Figure 3.5-1. Assembling an Application Program using DASD as Intermediate Storage for the Translator Output

```
// JOB      jobname
// ASSGN    SYSPCH,X'181'
// MTC      REW,SYSPCH
// EXEC     DFHEAP1#,SIZE=64K
    .
    Assembler source deck
    .
/*
// MTC      WTM,SYSPCH,2
// MTC      REW,SYSPCH
// RESET    SYSPCH
// ASSGN    SYSIPT,X'181'
// OPTION   SYM,ERRS,NODECK,CATAL
    PHASE    phase-name,*
    INCLUDE DFHEAI
// EXEC     ASSEMBLY,SIZE=64K
// EXEC     LNKEDT
/&
// JOB      RESET
// RESET    SYSIPT
/&
```

Figure 3.5-2. Assembling an Application Program using Tape as Intermediate Storage for Translator Output

COBOL PROGRAMS

For COBOL programs using the command-level interface the following conditions must be met:

- The LIB option must be specified for the compilation step so that the COPY statements inserted by the translator can be processed correctly.
- DFHECI must exist in the relocatable library. (It is supplied in the private relocatable library on the distribution volume.)
- An INCLUDE card for DFHECI must follow immediately after the PHASE card and before the EXEC FCOBOL card.

```
// JOB      jobname
// DLBL     IJSYSPH,'COBOL TRANSLATION',yy/ddd
// EXTENT   SYSPCH,balance of extent information
ASSGN   SYSPCH,DISK,VOL=volid,SHR
// EXEC     DFHECP1#,SIZE=64K
 CBL LIB
    .
    COBOL source deck
    .
/*
CLOSE       SYSPCH,PUNCH
// DLBL     IJSYSIN,'COBOL TRANSLATION',yy/ddd
// EXTENT   SYSIPT
ASSGN   SYSIPT,DISK,VOL=volid,SHR
// OPTION   SYM,ERRS,NODECK,CATAL
    PHASE    phase-name,*
    INCLUDE DFHECI
// EXEC     FCOBOL,SIZE=64K
// EXEC     LNKEDT
/&
// JOB      RESET
CLOSE       SYSIPT,X'00C'
/&
```

Figure 3.5-3. Compiling COBOL Application Programs using DASD as Intermediate Storage for the Translator Output

```
// JOB      jobname
// ASSGN    SYSPCH,X'181'
// MTC      REW,SYSPCH
// EXEC     DFHECP1#,SIZE=64K
 CBL LIB
    .
    COBOL source deck
    .
/*
// MTC      WTM,SYSPCH,2
// MTC      REW,SYSPCH
// RESET    SYSPCH
// ASSGN    SYSIPT,X'181'
// OPTION   SYM,ERRS,NODECK,CATAL
    PHASE   phase-name,*
    INCLUDE DFHECI
// EXEC     FCOBOL,SIZE=64K
// EXEC     LNKEDT
/&
// JOB      RESET
// RESET    SYSIPT
/&
```

Figure 3.5-4. Compiling COBOL Application Programs using Tape as Intermediate Storage for the Translator Output

PL/I PROGRAMS

PL/I application programs using the command-level interface must satisfy the following conditions:

- Either the INCLUDE or MACRO option must be specified so that the %INCLUDE statement inserted by the translator is processed correctly.
- The modules DFHPL1I and DFHSAP that are supplied with PL/I must be in the relocatable library.
- The module DFHEPI must be in the relocatable library. (It is supplied in the private relocatable library on the distribution volume.)
- An INCLUDE card for DFHPL1I must follow immediately after the PHASE card and before the EXEC PLIOPT card.

```
// JOB      jobname
// DLBL     IJSYSPH,'PL/I TRANSLATION',yy/ddd
// EXTENT   SYSPCH,balance of extent information
ASSGN  SYSPCH,DISK,VOL=volid,SHR
// EXEC     DFHEPP1#,SIZE=64K
*PROCESS    INCLUDE;
    .
    .
    PL/I source deck
    .
    .
/*
CLOSE       SYSPCH,PUNCH
// DLBL     IJSYSIN,'PL/I TRANSLATION',yy/ddd
// EXTENT   SYSIPT
ASSGN  SYSIPT,DISK,VOL=volid,SHR
// OPTION   CATAL
   PHASE    phase-name,*
   INCLUDE  DFHPL1I
// EXEC     PLIOPT,SIZE=64K
// EXEC     LNKEDT
/&
// JOB      RESET
CLOSE       SYSIPT,X'00C'
/&
```

Figure 3.5-5. Compiling PL/I Application Programs using DASD as Intermediate Storage for the Translator Output

```
// JOB      jobname
// ASSGN    SYSPCH,X'181'
// MTC      REW,SYSPCH
// EXEC     DFHEPP1#,SIZE=64K
*PROCESS    INCLUDE;
    .
    .
    PL/I source deck
    .
    .
/*
// MTC      WTM,SYSPCH,2
// MTC      REW,SYSPCH
// RESET    SYSPCH
// ASSGN    SYSIPT,X'181'
// OPTION   CATAL
   PHASE    phase-name,*
   INCLUDE  DFHPL1I
// EXEC     PLIOPT,SIZE=64K
// EXEC     LNKEDT
/&
// JOB      RESET
CLOSE       SYSIPT
/&
```

Figure 3.5-6. Compiling PL/I Application Programs using Tape as Intermediate Storage for the Translator Output

RPG II programs using the command-level interface must satisfy the following conditions:

- The module DFHERI must exist in the relocatable library. (It is supplied in the private relocatable library on the distribution volume.)
- An INCLUDE card for DFHERI must follow immediately after the PHASE card and before the EXEC RPG II card.

```
// JOB      jobname
// DLBL     IJSYSPH,'RPG II TRANSLATION',yy/ddd
// EXTENT   SYSPCH,balance of extent information
ASSGN  SYSPCH,DISK,VOL=volid,SHR
// EXEC     DFHERP1#,SIZE=128K
    .
    .
    RPG II source deck
    .
    .
/*
CLOSE       SYSPCH,PUNCH
// DLBL     IJSYSIN,'RPG II TRANSLATION',yy/ddd
// EXTENT   SYSIPT
ASSGN  SYSIPT,DISK,VOL=volid,SHR
// OPTION   CATAL
    PHASE   phase-name,*
    INCLUDE DFHERI
// EXEC     RPGII,SIZE=64K
// EXEC     LNKEDT
/&
// JOB      RESET
CLOSE       SYSIPT,X'00C'
/&
```

Figure 3.5-7. Compiling RPG II Application Programs using DASD as Intermediate Storage for the Translator Output

```
// JOB      jobname
// ASSGN    SYSPCH,X'181'
// MTC      REW,SYSPCH
// EXEC     DFHERP1#,SIZE=64K
    .
    .
    RPG II source deck
    .
    .
/*
// MTC      WTM,SYSPCH,2
// MTC      REW,SYSPCH
// RESET    SYSPCH
// ASSGN    SYSIPT,X'181'
// OPTION   CATAL
   PHASE    phase-name,*
   INCLUDE  DFHERI
// EXEC     RPG II,SIZE=64K
// EXEC     LNKEDT
/&
// JOB      RESET
CLOSE       SYSIPT
/&
```

Figure 3.5-8. Compiling RPG II Application Programs using Tape as Intermediate Storage for the Translator Output

SUPPLIED CATALOGED PROCEDURES

To facilitate compiling programs using the command-level interface, the source code of example cataloged procedures is supplied in the following books in the CICS/VS source-statement library:

Z.DFHEITAL for Assembler language
Z.DFHEITCL for COBOL
Z.DFHEITPL for PL/I
Z.DFHEITRL for RPG II

Before putting these cataloged procedures into the procedure library the system programmer must modify:

1. The extent information.
2. The volume identifications.

```
// DLBL    IJSYSPH,'ASSEMBLER TRANSLATION',69/365
// EXTENT  SYSPCH,,1,0,399,19
ASSGN  SYSPCH,DISK,VOL=DB0007,SHR
// EXEC    DFHEAP1#,SIZE=64K
CLOSE      SYSPCH,PUNCH
// DLBL    IJSYSIN,'ASSEMBLER TRANSLATION',69/365
// EXTENT  SYSIPT
ASSGN  SYSIPT,DISK,VOL=DB0007,SHR
   INCLUDE DFHEAI
// EXEC    ASSEMBLY,SIZE=64K
CLOSE      SYSIPT,READER
// EXEC    LNKEDT
```

Figure 3.5-9. Example of Cataloged Procedure (DFHEITAL) for Assembling Application Programs using DASD as Intermediate Storage for Translator Output

```
// DLBL    IJSYSPH,'COBOL TRANSLATION',69/365
// EXTENT  SYSPCH,,1,0,399,19
ASSGN  SYSPCH,DISK,VOL=DB0007,SHR
// EXEC    DFHECP1#,SIZE=64K
CLOSE      SYSPCH,PUNCH
// DLBL    IJSYSIN,'COBOL TRANSLATION',69/365
// EXTENT  SYSIPT
ASSGN  SYSIPT,DISK,VOL=DB0007,SHR
   INCLUDE DFHECI
// EXEC    FCOBOL,SIZE=64K
CLOSE      SYSIPT,READER
// EXEC    LNKEDT
```

Figure 3.5-10. Example of Cataloged Procedure (DFHEITCL) for Compiling COBOL Application Programs using DASD as Intermediate Storage for Translator Output

```
// DLBL    IJSYSPH,'PL/I TRANSLATION',69/365
// EXTENT  SYSPCH,,1,0,399,19
ASSGN  SYSPCH,DISK,VOL=DB0007,SHR
// EXEC    DFHEPP1#,SIZE=64K
CLOSE      SYSPCH,PUNCH
// DLBL    IJSYSIN,'PL/I TRANSLATION',69/365
// EXTENT  SYSIPT
ASSGN  SYSIPT,DISK,VOL=DB0007,SHR
   INCLUDE DFHPL1I
// EXEC    PLIOPT,SIZE=64K
CLOSE      SYSIPT,READER
// EXEC    LNKEDT
```

Figure 3.5-11. Example of Cataloged Procedure (DFHEITPL) for Compiling PL/I Application Programs using DASD as Intermediate Storage for Translator Output

```
// DLBL    IJSYSPH,'RPG II TRANSLATION',69/365
// EXTENT  SYSPCH,,1,0,399,19
ASSGN  SYSPCH,DISK,VOL=DB0007,SHR
// EXEC    DFHERP1#,SIZE=64K
CLOSE      SYSPCH,PUNCH
// DLBL    IJSYSIN,'RPG II TRANSLATION',69/365
// EXTENT  SYSIPT
ASSGN  SYSIPT,DISK,VOL=DB0007,SHR
   INCLUDE DFHERI
// EXEC    RPG II,SIZE=64K
CLOSE      SYSIPT,READER
// EXEC    LNKEDT
```

Figure 3.5-12. Example of Cataloged Procedure (DFHEITRL) for Compiling RPG II Application Programs using DASD as Intermediate Storage for Translator Output.

Note: These procedures are valid only if the output from the translation step is written to a disk other than that on which the procedure library resides.

Using Supplied Cataloged Procedures

```
// JOB     jobname
// OPTION  NODECK,CATAL
   PHASE   phase-name,*
// EXEC    PROC=DFHEITAL
   .
   Assembler source deck
   .
/*
/&
```

Figure 3.5-13. Use of Cataloged Procedure for Assembling Application Programs

```
// JOB     jobname
// OPTION  NODECK,CATAL
   PHASE   phase-name,*
// EXEC    PROC=DFHEITCL
 CBL LIB
   .
   COBOL source deck
   .
/*
/&
```

Figure 3.5-14. Use of Cataloged Procedure for Compiling COBOL Application Programs

```
// JOB      jobname
// OPTION   CATAL
   PHASE    phase-name,*
// EXEC     PROC=DFHEITPL
*PROCESS    INCLUDE;
   .
   PL/I source deck
   .
/*
/&
```

Figure 3.5-15. Use of Cataloged Procedure for Compiling PL/I Application Programs

```
// JOB      jobname
// OPTION   CATAL
   PHASE    phase-name,*
// EXEC     PROC=DFHEITRL
   .
   RPG II source deck
   .
/*
/&
```

Figure 3.5-16. Use of Cataloged Procedure for Compiling RPG II Application Programs

Notes:

1. If tape or disk is used as intermediate storage for the translator output, 81 byte records are written out, but only 80 bytes are input to the following compiler or assembler.

2. (PL/I only)

   a. Warning messages issued by the PL/I compiler stating that there are fewer arguments than parameters for the calls to procedure DFHEInn should be ignored.

   b. Weak external references unresolved by the linkage editor should be ignored.

   c. Warning messages from the PL/I compiler, stating that arguments and parameters for calls to procedure DFHEInn do not match, are issued automatically. The user should check that the arguments and parameters specified are those required; otherwise these messages can be ignored.

Map Creation and Cataloging

Basic mapping support (BMS) provides the interface between the user's application program and the CICS/VS terminal control program.

The application program passes data to BMS and receives data from BMS in a device-independent format. The terminal control commands (see Chapter 2.1) issued by the application program use BMS to control formatting of the data and to initiate input from and output to a terminal.

PHYSICAL AND SYMBOLIC DESCRIPTION MAPS

Two forms of map need to be defined by CICS/VS macros and assembled offline in advance of running the application program. The two forms are:

1. A physical map used by BMS to convert data to or from the format required for the application program.

2. A symbolic description map used by the application programmer to symbolically refer to the data in the terminal buffer.

The physical map is a table of information about each field, and is stored in the CICS/VS program library to be loaded by BMS at execution time. The symbolic description map is a set of source statements that are cataloged into the appropriate source library (Assembler language, COBOL, PL/I, or RPG II) and copied into the application program when it is assembled, compiled, or translated.

The programmer defines and provides names for fields and groups of fields that may be written to and received from the devices supported by BMS. The symbolic description map can be copied into each application program that uses the associated physical map. Data can thus be passed to and from the application program under the field names in the symbolic description map. Since the application program is written to manipulate the data under the field names, altering the map format by adding new fields or rearranging old fields does not necessarily alter the program logic.

If the map format is altered, it is necessary in most cases to make the appropriate changes to the macro instructions that describe the map and then reassemble both the physical map and the symbolic description map. The new symbolic description map must then be copied into the application program and the program reassembled. Certain map alterations can be made without reassembling the symbolic description map. In particular, if the only change to the map is addition of extended attributes, it is not necessary to modify the application program. Instead, the operator defines the required extended attributes in the map definition macros (DFHMSD, DFHMDI, and DFHMDF), and specifies EXTATT=MAPONLY.

Map Definition

Maps are defined by three macros: DFHMSD, DFHMDI, and DFHMDF.

The DFHMSD macro

- defines a map set
- specifies the attributes of the map set

- indicates whether a particular set of macros is for a physical map or for a symbolic description map

The DFHMDI macro

- defines a map
- specifies the attributes of symbols used in the map
- specifies the size of the map

The DFHMDF macro

- defines a field within a map
- specifies the position of the field
- specifies the length of the field
- specifies the attributes of symbols which appear in the map.

The formats of these macro instructions are given later in this chapter. Examples of their use are included in the sample applications described and listed in the language specific chapters of Part 2 of this manual.

An attribute of a map is specified by the DFHMDI macro. Its value overrides, for that map, the corresponding attribute of its map set (defined by macro DFHMSD). An individual field within a map can have yet another attribute value if the DFHMDF macro is used to override the value for the whole map.

The map definition macro instructions are assembled twice, once to produce the map used by BMS, and once to produce the symbolic storage definition (or DSECT) that will be copied into the application program.

Copying Symbolic Description Maps

The BMS symbolic description maps must be copied into the application program as shown in the following examples. In these examples, mapsetname1, mapsetname2, and mapsetname3 are the names of members that contain the assembly of a BMS symbolic storage definition.

1. Assembler-language COPY instructions for each symbolic storage definition. If it is required that each definition overlays the same area, the second and subsequent COPY instructions must be preceded by an ORG instruction to reposition the Assembler to the start of the data area.

```
COPY mapsetname1
COPY mapsetname2
COPY mapsetname3
```

2. COBOL COPY statements for each symbolic storage definition. Note that mapname1, mapname2, and mapname3 in this example are the names of the first maps in the map sets.

```
LINKAGE SECTION.
01 mapname1I COPY mapsetname1.
01 mapname2I COPY mapsetname2.
01 mapname3I COPY mapsetname3.
.
.
.
```

3. PL/I %INCLUDE statements.

```
%INCLUDE mapsetname1;
%INCLUDE mapsetname2;
%INCLUDE mapsetname3;
.
.
.
```

MAP DEFINITION MACROS

This section describes the syntax and operands of the three map definition macros (DFHMSD, DFHMDI, and DFHMDF)

DFHMSD Macro — Define a Map Set

All maps must belong to a map set, even if the set contains only one map. BMS generates and stores map sets in the CICS/VS program library under the names selected by the application programmers.

Information pertaining to an entire map set is specified in the DFHMSD macro instruction, which always appears at the beginning and end of each map set generation. The one at the beginning indicates whether physical maps or symbolic description maps are being generated; the one at the end indicates the end of the map set.

All operands other than the TYPE operand of a DFHMSD macro instruction are the same for a physical map generation run and for the corresponding symbolic description map generation run. TYPE=MAP is specified for the former, and TYPE=DSECT for the latter. Alternatively, physical maps and symbolic description maps can be assembled in the same job by the use of job control language options, as described later in this chapter, under "Cataloging Maps."

The format of the DFHMSD macro is as follows:

```
┌──────┬──────┬──────────────────────────────────────────────────────────┐
│      │      │                                                          │
│mapset│DFHMSD│ TYPE={DSECT|MAP|FINAL}                                   │
│      │      │ EXTATT={NO|YES|MAPONLY,}                                 │
│      │      │ COLOR={DEFAULT|BLUE|RED|PINK|GREEN|TURQUOISE|            │
│      │      │        YELLOW|WHITE},                                    │
│      │      │ PS={BASE|psid},                                          │
│      │      │ HILIGHT={OFF|BLINK|REVERSE|UNDERLINE}                    │
│      │      │ VALIDATION={[MUSTFILL][,MUSTENTER]}                      │
│      │      │ ,MODE=INOUT                                              │
│      │      │ ,TIOAPFX=YES                                             │
│      │      │ [,BASE=name]                                             │
│      │      │ [,CTRL=([PRINT][,{L40|L64|L80|HONEOM}]                   │
│      │      │         [,FREEKB][,ALARM][,FRSET])]                      │
│      │      │ [,LANG={ASM|COBOL|PLI|RPG}]                              │
│      │      │ [,STORAGE=AUTO]                                          │
│      │      │                                                          │
└──────┴──────┴──────────────────────────────────────────────────────────┘
```

Operands of DFHMSD Macro

mapset
: is the one- to seven-character name of the map set, to be specified in the MAPSET operand of any terminal control command that refers to the map set. The name must begin with an alphabetic character and must differ from other map names or program names. An entry for the name must be made in the Processing Program Table (PPT) — see Chapter 3.4. The name must be specified in the PHASE card when the physical map is link edited.

TYPE=
: indicates the type of run.

  DSECT
  : indicates that this is a symbolic description map generation run to create the list of field names to be copied into an application program. If a single map set is to be used by application programs written in different languages, all of the macros for the map set must be provided for each language used.

  MAP
  : indicates that this is a physical map generation run to create the control information block used by BMS to perform mapping. This physical map is stored in the CICS/VS program library.

  FINAL
  : must be coded in the DFHMSD macro instruction that marks the end of the map set. If other parameters are coded in the DFHMSD TYPE=FINAL macro instruction, they will be ignored.

  The values of TYPE= can be overridden in the SYSPARM option when the maps are cataloged (see under "Cataloging Maps," later in this chapter).

EXTATT

Indicates whether the extended attributes are supported.

NO indicates that no support for extended attributes is required. This is the default unless COLOR, PS, HILIGHT or VALIDATION is specified on the DFHMSD macro. It results in the same maps and application structures as appear in existing applications.

MAPONLY indicates that the extended field attributes can be specified in the maps within the map set, but that application structures contain no field for the extended attributes. This means that they are the same as generated on CICS/VS Version 1 release 4.0, and the extended attributes can not be dynamically changed. This option can be used to add extended attributes to existing maps, without having to recompile the existing application programs.

YES indicates that the extended field attributes can be specified in the map and can be dynamically modified.

COLOR

specifies the color of characters used in a mpa set defined for a device using the extended data stream.

PS

specifies the symbol set to be used in a mapset using programmed symbols.

HILIGHT

specifies the form of highlighting to be applied to characters in a mapset.

A symbol set identifier (psid) can be expressed as either a single character or as two hexadecimal digits (if there is no EBCDIC equivalent). For example, psid=a or psid=X'74'. Identifiers will be allocated by the installation's system programmer.

BASE=name (COBOL and PL/I only)

is used to indicate that the same storage base will be used for the symbolic description maps from more than one map set. The same name is coded in the BASE operand for each map set that is to share the same storage base. Since all map sets with the same base describe the same storage, data related to a previously-used map set may be overwritten when a new map set is used. Furthermore, maps will overlay other maps from their own map set.

If this operand is omitted and STORAGE=AUTO is not specified, BASE=BMSMAPBR is assumed.

CTRL=

is used to specify device characteristics. Its parameters correspond to similarly named options of the terminal control commands described in Chapter 2.1. (Terminal control command options, when specified, override these parameters.)

PRINT
must be specified if the printer is to be started; if omitted, the data is sent to the printer buffer but is not printed. This operand is ignored if the map set is used with 3275s without the Printer Adapter feature or with 3277s.

L40, L64, L80, HONEOM
are mutually exclusive options that control the line length on the printer. L40, L64, and L80 force a carrier return/line feed after 40, 64, or 80 characters, respectively. HONEOM causes the printer to honor all new-line (NL) characters and the first end-of-message (EM) character that appear in displayable fields of the data stream. If the latter option is specified, the application program must insert the NL and EM characters into the data stream. If the NL character is omitted, a carrier return/line feed occurs at the physical end of the carriage. If the EM character is omitted, printing stops at the end of the 3270 buffer.

These operands are ignored for devices other than printers.

FREEKB
specifies that the keyboard should be unlocked after this map is written out. If omitted, the keyboard remains locked; further data entry from the keyboard is prevented until this status is changed.

ALARM
activates the 3270 audible alarm feature.

FRSET
indicates that the modified data tags (MDTs) of all fields currently in the 3270 buffer are to be reset to a not-modified condition (that is, field reset) before any map data is written to the buffer. This allows the DFHMDF ATTRB specification for the requested map to control the final status of any fields written or rewritten in response to a terminal control command.

LANG=
specifies the language in which the application program referring to a symbolic description map is written and, hence, is applicable for only a DFHMSD TYPE=DSECT macro.

ASM
— Assembler language

COBOL
— COBOL

PLI
— PL/I

RPG
— RPG II

MODE=INOUT
specifies that the map definition may be used for both input and output operations.

STORAGE=AUTO (not valid for RPG II)
specifies, for COBOL programs, that the symbolic storage definitions of the maps in the map set are to be separate (that is, not redefined) areas. This operand is used when the symbolic storage definitions are copied into the WORKING-STORAGE section of a program using the command-level interface and the storage for the separate maps in the map set is to be used concurrently.

specifies, for PL/I programs, that the symbolic storage definitions are to be declared AUTOMATIC. If the operand is omitted,the symbolic storage definitions are declared BASED.

specifies, for assembler language programs, that separate maps within a map set are to occupy separate storage, not to overlay one another.

If STORAGE=AUTO is specified, BASE=name cannot be used.

TIOAPFX=YES
is required for the entry level system (to reserve space for BMS control information).

DFHMDI Macro — Map Definition

The DFHMDI macro instruction is used to name a single map and define its size. When defining more than one map within a map set, the corresponding number of DFHMDI macro instructions must be used. If the maps are for use in a COBOL program, and the BASE operand is coded, they must be specified in descending order of the lengths of the associated COBOL structures. The format of the macro is as follows:

```
┌──────┬────────┬──────────────────────────────────────────────────┐
│      │        │                                                  │
│ map  │DFHMDI  │ SIZE=(lines,columns)                             │
│      │        │ EXTATT={NO|YES|MAPONLY,}                         │
│      │        │ COLOR={DEFAULT|BLUE|RED|PINK|GREEN|TURQUOISE|    │
│      │        │        YELLOW|WHITE},                            │
│      │        │ PS={BASE|psid},                                  │
│      │        │ HILIGHT={OFF|BLINK|REVERSE|UNDERLINE}            │
│      │        │ VALIDATION={[MUSTFILL][,MUSTENTER]}              │
│      │        │                                                  │
└──────┴────────┴──────────────────────────────────────────────────┘
```

Operands of DFHMDI Macro

map
: is the one- to seven-character name of the map, to be specified in the MAP option of any terminal control command that refers to the map. Note, however, that for RPG II programs, the name must not exceed 5 characters.

COLOR
PS
HILIGHT
: The operands have the same meanings as the operands for DFHMSD.

SIZE=
: gives the dimensions of a map in terms of length and width.

  lines
  : is a value from 1 to 240, indicating the length of a map as a number of lines.

  columns
  : is a value from 1 to 240, indicating the width of a map as a number of characters per line. Space for the attribute byte should be included in the column specification.

DFHMDF Macro — Field Definition

The DFHMDF macro is used to define one field in a map. One DFHMDF macro is required for each field, giving information such as symbolic field name, field position, field length, attribute byte, initial constant data, justification of input, and COBOL or PL/I data picture.

The maximum number of named fields that may be specified depends on various factors. For details, see the CICS/VS Application Programmer's Reference Manual (Command Level). (In normal use for the entry level system, the limit will not be exceeded.)

The format of the macro is as follows:

```
|[fld] |DFHMDF | POS=(lines,columns)
|      |       | ,COLOR={DEFAULT|BLUE|RED|PINK|GREEN|TURQUOISE|
|      |       |                 YELLOW|WHITE}
|      |       | ,PS={BASE|psid}
|      |       | ,HILIGHT={OFF|BLINK|REVERSE|UNDERSCORE}
|      |       | [,VALIDATION={[MUSTFILL][,MUSTENTER]}
|      |       | [,ATTRB=([{ASKIP|PROT|UNPROT}[,NUM][,{BRT|NORM|DRK}]
|      |       |          [,DET][,IC][,FSET])]
|      |       | [,DECPOS=number]
|      |       | [,GRPNAME=group-name]
|      |       | [,INITIAL='character data'|XINIT='hexadecimal data']
|      |       | [,JUSTIFY=([{LEFT|RIGHT}][,{BLANK|ZERO}])]
|      |       | [,LENGTH=number]
|      |       | [,OCCURS=number]
|      |       | [,PICIN='value']
|      |       | [,PICOUT='value']
```

Operands of DFHMDF Macro

fld

is the one- to seven-character name of the field, used as a symbolic reference to the field by the application program. Note, however, that for RPG II programs, the field name must never exceed 5 characters, and if OCCURS= is specified, the field name must not exceed 3 characters.

Although specification of a field name is not required for every field within a map, a field name must be specified for at least one field of any map to be compiled under COBOL or PL/I. All fields within a group must have names.

If no name is specified for a field, an application program cannot modify the field attributes or data. For an output map, omitting the field name may be appropriate when the INITIAL operand is used to specify field contents. If a field name is specified and the map that includes the field is used in a terminal control command, any data supplied by the user overlays data supplied by the INITIAL operand of DFHMDF.

POS=
specifies the line and column of the attribute character for this field, relative to line 1, column 1.

COLOR
PS
HILIGHT

These operands have the same meanings as the operands for DFHMSD.

VALIDATION
specifies the kind OS validation to be performed on data entered into an input field. It can be:

MUSTFILL
The system will not accept data unless it fills the input field.

MUSTENTER
The system will not resume execution unless data is entered.

ATTRB=
specifies device-dependent characteristics and attributes, such as the capability of a field to receive data or the intensity to be used when the field is output. If the ATTRB operand is specified within a group of fields, it must be specified in the first field entry. (Since a group of fields appears as one field to the 3270, the ATTRB specification refers to all of the fields in the group.)

ASKIP
prevents data from being keyed into the field and causes the cursor to automatically skip over the field.

PROT
prevents data from being keyed into the field, but no automatic skipping occurs.

UNPROT
specifies that data can be keyed into the field.

NUM
ensures that the data entry keyboard is set to numeric shift for this field unless the operator presses the alpha shift key, and prevents entry of nonnumeric data if the Keyboard Numeric Lock feature is installed.

BRT
specifies that the field display intensity is to be brighter than normal.

NORM
specifies that the field intensity is to be normal.

DRK
specifies that the field is nonprint/nondisplay. DRK cannot be specified if DET is specified.

DET
specifies that the field is potentially detectable by the selector pen or cursor select key.

The first character of a 3270 selector pen detectable field must be a "?", ">", "&", or blank. If the first character is "&" or blank, the field is a selector-pen attention field; if the first character is "?" or ">", the field is a selector-pen selection field. (See the publication IBM 3270 Information Display System Component Description for further details of selector pen detectable fields.) If DET is specified, only one data byte is reserved for each input DET field. This byte is set to X'00' when the field is unselected, or to X'FF' when the field is selected; no other data is supplied, even if the field is a selection field and the ENTER key has been pressed.

IC
indicates that the cursor is to be placed in the first position of this field. The IC attribute for the last field for which it is specified in a map is the one that takes effect. If not specified for any fields in a map, the default location is line 1, column 1. Specifying IC with ASKIP or PROT causes the cursor to be placed in an unkeyable field.

This option may be overridden by specifying the CURSOR option in a SEND command.

FSET
specifies that the modified data tag (MDT) for this field should be set when the field is sent to a terminal.

Specification of FSET causes the 3270 to treat the field as though it has been modified. On a subsequent read from the terminal, this field is read, whether or not it has been modified. The MDT remains set until the field is rewritten without ATTRB=FSET or until a SEND command with the FRSET option is issued.

Either of two sets of defaults may apply when not all parameters are specified. If no ATTRB parameters are specified, ASKIP and NORM are assumed. If any parameter is specified, UNPROT and NORM are assumed for that field unless overridden by a specified parameter.

DECPOS=number (RPG II only)
specifies a value to be inserted into the 'decimal positions' column of the generated RPG input specification for the data area of the field being defined. 'number' may be any value from 0 to 9. This operand has no effect in a DFHMSD TYPE=MAP assembly.

GRPNAME=group name
is the name used to generate symbolic storage definitions and to combine specific fields under one group name. The group name has a maximum length of five characters for RPG II programs, and seven characters for other languages. The fields composing a group must be contiguous, and the same group name must be specified for each field that is to belong to the group. A field name must be specified for every field that belongs to the group, and the POS operand must be also specified to ensure that the fields are contiguous. All the DFHMDF macros defining the fields of a group must be placed together, and in the correct order (upward numeric order of the POS operand). The fields in a group must follow on; there can be intervening gaps between them, but not other fields from outside the group. For example, the first 20 columns of the first six lines of a map can be defined as a group of six fields, so long as the remaining columns on the first five lines are not defined as fields.

The ATTRB operand specified on the first field of the group applies to all of the fields within the group. The lengths of the fields within the group must not collectively exceed 256 bytes. If this operand is specified, the OCCURS operand (below) cannot be specified.

INITIAL='character data'|XINIT='hexadecimal data'
is used to specify constant or default data for an output field. The INITIAL operand is used to specify data in character form; the XINIT operand is used to specify data in hexadecimal form. INITIAL and XINIT are mutually exclusive.

For fields with the DET attribute, initial data that begins with a blank character, "&", ">", or "?" should be supplied (or, for XINIT, the hexadecimal equivalent, '40', '50', '6E', or '6F').

The number of characters that can be specified in the INITIAL operand is restricted to the continuation limitation of the assembler to be used or to the value specified in the LENGTH operand (whichever is the smaller).

Hexadecimal data is written as an even number of hexadecimal digits, for example, XINIT=C1C2. If the number of valid characters is smaller than the field length, the data will be padded on the right with blanks. For example, XINIT=C1C2 might result in an initial field of 'AB

Use XINIT with care. If hexadecimal data is specified that corresponds with line or format control characters, the results will be unpredictable.

JUSTIFY=
specifies field justification for input operations.

LEFT
— data is to be left-justified.

RIGHT
— data is to be right-justified.

BLANK
— blanks are to be inserted in any unfilled data positions.

ZERO
— zeros are to be inserted in any unfilled data positions.

LEFT and RIGHT are mutually exclusive, as are BLANK and ZERO. If certain parameters are specified but others are not, assumptions are made as follows:

Specified	Assumed
LEFT	BLANK
RIGHT	ZERO
BLANK	LEFT
ZERO	RIGHT

If JUSTIFY is omitted, but the NUM attribute is specified, RIGHT and ZERO are assumed. If JUSTIFY is omitted, but attributes other than NUM are specified, LEFT and BLANK are assumed.

LENGTH=number
specifies the field length (from 1 to 256 bytes), not including the attribute character. The sum of the lengths of the fields within a group must not exceed 256 bytes. LENGTH can be omitted if PICIN or PICOUT is specified but is required otherwise.

The field length must not extend beyond the end of the map.

OCCURS=number
specifies that the indicated number of entries for the field are to be generated in such a way that the fields are addressable as entries in a matrix or an array. For non-Assembler applications, this permits several data fields to be addressed by the same name, qualified by subscript. OCCURS and GRPNAME are mutually exclusive.

PICIN='value' (COBOL and PL/I only)
specifies that a user-defined picture, composed of picture characters that are valid in the language being used, is to be used to edit input data.

PICOUT='value' (COBOL and PL/I only)
is like PICIN, but it applies to output fields.

XINIT=hexadecimal data
(See under "INITIAL" above).

CATALOGING MAPS

Physical maps are cataloged in the core image library for use by BMS, and symbolic description maps are cataloged in the source statement library, for the programmer's use.

Both types of maps can be generated using one set of DFHMSD, DFHMDI, and DFHMDF macro instructions. The maps can then be assembled and cataloged in one job stream using the SYSPARM option as the following steps indicate:

1. Catalog the BMS source statements into the source statement library under the name DUMMYMAP.

2. Assemble the BMS source statements to create the physical map by specifying SYSPARM='MAP'.

3. Link-edit and catalog the physical map into the core image library.

4. Assemble the BMS source statements to create the symbolic description map by specifying SYSPARM='DSECT'.

5. Catalog the symbolic storage definition into the source statement library.

6. Delete DUMMYMAP from the source statement library.

Figures 3.5-17 and 3.5-18 are job control examples of how to catalog BMS maps, using a disk or a tape as intermediate storage, respectively. Note that the names in the PHASE cards are the same as specified in the application program:

1. in the INCLUDE/COPY statements that include the symbolic description maps;

2. in the MAPSET option of the SEND and RECEIVE MAP commands.

```
// JOB CICSBMS
*  CICS/DOS/VS ASSEMBLE AND CATALOG BMS MAPS
*  ASSIGN PRIVATE LIBRARIES IF NEEDED
// EXEC MAINT                                          STEP 1
   CATALS  A.DUMMYMAP
   BKEND
   .
   source defining BMS map
   .
*  DO NOT INCLUDE THE MAP END STATEMENT
   BKEND
/*
// OPTION   CATAL,NODECK,SYSPARM='MAP'
   PHASE XDFHCMA,*                    (map set name=XDFHCMA)
// EXEC  ASSEMBLY,SIZE=64K                             STEP 2
   COPY  DUMMYMAP
   END
/*
// EXEC LNKEDT                                         STEP 3
// DLBL  IJSYSPH,'MAP DSECT',0
// EXTENT  SYSPCH,,1,0,2698,19          (sample extent)
   ASSGN   SYSPCH,DISK,VOL=CICSCS,SHR   (assgn SYSPCH to DASD)
// OPTION  DECK,SYSPARM='DSECT'
// EXEC  ASSEMBLY,SIZE=64K                             STEP 4
   PUNCH ' CATALS  XDFHCMA'             (map set name=XDFHCMA)
   COPY  DUMMYMAP
   END
/*
// RESET  SYSPCH
// DLBL  IJSYSIN,'MAP DSECT',0
// EXTENT  SYSIPT
   ASSGN   SYSIPT,DISK,VOL=CICSCS,SHR  ('cuu' from SYSPCH in STEP 4)
// EXEC MAINT                                          STEP 5
/*
// RESET SYSIPT
// EXEC MAINT                                          STEP 6
   DELETS  A.DUMMYMAP
/*
/&
```

Figure 3.5-17. Cataloging Maps using DASD Intermediate Storage

```
// JOB CICSBMS
*  CICS/DOS/VS ASSEMBLE AND CATALOG BMS MAPS
*  ASSIGN PRIVATE LIBRARIES IF NEEDED
// EXEC   MAINT                                          STEP 1
   CATALS A.DUMMYMAP
   BKEND
   .
   source defining BMS map
   .
*  DO NOT INCLUDE THE MAP END STATEMENT
   BKEND
/*
// OPTION     CATAL,NODECK,SYSPARM='MAP'
   PHASE      XDFHCMB,*                (map set name=XDFHCMB)
// EXEC       ASSEMBLY,SIZE=64K                          STEP 2
   COPY       DUMMYMAP
   END
/*
// EXEC       LNKEDT                                     STEP 3
// ASSGN      SYSPCH,X'cuu'            (assign SYSPCH to tape)
// OPTION DECK,SYSPARM='DSECT'
// EXEC       ASSEMBLY,SIZE=64K                          STEP 4
   PUNCH      ' CATALS C.XDFHCMB'       (map set name=XDFHCMB)
   COPY       DUMMYMAP
   END
/*
// MTC        WTM,SYSPCH,2
// MTC        REW,SYSPCH
// RESET      SYSPCH
// ASSGN      SYSIPT,X'cuu'        cuu' from SYSPCH in STEP 4)
// EXEC       MAINT                                      STEP 5
/*
// MTC        REW,SYSIPT
// RESET      SYSIPT
// EXEC       MAINT                                      STEP 6
   DELETS A.DUMMYMAP
/*
/&
```

Figure 3.5–18. Cataloging Maps using Tape as Intermediate Storage

Chapter 3.6. Entry-Level System Execution

This chapter, which assumes that the system has been brought up as described earlier in Chapter 3.3, lists the startup override parameters that can be specified during system initialization, and provides a guide for operating the system. Operating procedures are given for console operations, master terminal operations, and user terminal operations.

Startup Override Parameters

Where a choice of different options may be made at startup time, startup override parameters are used, for example to specify the file access method used, or to specify a choice of tables where more than one table has been generated. Default values are applied when override parameters are not specified.

The parameters can be specified either on SYSIPT or directly from the system console. The choice of either or both of these methods is made through the VSE UPSI job control card. If the UPSI card turns on bit zero, the parameters can be read from SYSIPT; if it turns on bit two, they can be entered from the console. Thus, for example, // UPSI 101 would establish both modes of communication.

In console mode, the message

```
DFH1500 - PARAMETER CHANGES REQUIRED
```

appears at the console. Valid operator responses are:

'ØØ' (EOB)	–	No override parameters
'N '	–	No override parameters
'SI'	–	Override parameters from SYSIPT
'CN'	–	Override parameters follow from system console

The override parameters are separated by commas, and one or more may be entered in each 80-byte fixed-length record, starting in column 1. The end of the list of override parameters is indicated by $END.

The override parameters that may be issued for the entry level system are as follows, the default values being indicated by underscores where applicable.

APPLID= (VTAM only)
: specifies a one- to eight-character application name, defined to VTAM during VTAM system definition, which identifies CICS/DOS/VS-ELS to VTAM as an application program. This override must be used if APPLID=name is not specified in the DFHTCT TYPE=INITIAL macro instruction.

BSCODE=ASCII
: specifies that the ASCII communication code is to be supported for terminals, in addition to EBCDIC. If this override is not specified, only EBCDIC may be used.

BTAM=

specifies that BTAM is to be used as a terminal access method, and specifies whether support is required for local terminals, remote terminals, or both.

LOCAL
— locally attached terminals

REMOTE
— remotely attached terminals

(LOCAL,REMOTE)
— both locally and remotely attached terminals

DATFORM=

specifies the format to be used for the date in messages issued by CICS/VS. Valid parameters are:

MMDDYY

DDMMYY

YYMMDD

DCT=

specifies the destination control table suffix (see Chapter 3.4), or "NO" (the default) if transient data support is not required.

DL1=

indicates whether Data Language/I (DL/I) data bases are to be accessed during execution of CICS/DOS/VS-ELS. This parameter applies only to DL/I DOS/VSE.

NO
— DL/I will not be used.

YES
— DL/I will be used.

DUMPDEV=

specifies the device type to be used for transaction dump output.

TAPE
— tape, which must use logical unit SYS010.

2314

3330

3340

3350

FBA

DUMPDS=

specifies which dump data set is to be used first.

A
— the data set to be used first is DFHDMPA

B
— the data set to be used first is DFHDMPB

EXTRA=
specifies whether transient data extrapartition support is required.

NO

YES

FCT=
specifies the file control table suffix (see Chapter 3.4), or "NO" (the default) if file control support is not required.

FDUMP=
specifies the type of dump that is to be produced if CICS/DOS/VS-ELS terminates abnormally or CEMT SNAP (see "Master Terminal Operations," later in this chapter) is issued. For an abnormal termination, CICS/DOS/VS-ELS terminates after the dump is complete; for CEMT SNAP, it continues normally.

FORMAT
— a formatted dump of the major control blocks, arranged in logical order.

PARTN
— a dump of the CICS/DOS/VS-ELS partition (DOS/VS PDUMP).

FULL
— both of the above types of dump.

NO
— a dump of both the supervisor and the partition, but only for abnormal termination; no dump will be produced for CEMT SNAP.

FERS=
Indicates whether the FACILITY ERROR RECOGNITION SYSTEM (FERS) is to be used to log BTAM terminal errors.

NO

YES

FILE=
specifies which file access methods are to be used.

ISAM

VSAM

(ISAM,VSAM)

INTRA=
specifies whether transient data intrapartition support is required, and whether this is to be via DAM or VSAM.

NO

DAM

VSAM

MSGLVL=

specifies a message level, which controls the generation of messages to the console during system initialization.

2
indicates that all messages are to be printed on SYSLST and SYSLOG.

1
indicates that all messages are to be printed on SYSLOG.

0
indicates that only critical I/O errors or interactive messages are to be printed.

MXT=

specifies the maximum number of tasks that may be executed concurrently. No new tasks are initiated while the number of concurrent tasks is at the specified maximum. The range is from 2 through 999, 5 being the default.

NSD=

specifies the maximum number of nonsequential disk extents that will exist for any data set involved in the execution of ELS. ELS initialisation uses this value to determine how much storage to reserve at the beginning of the partition for label processing when the data sets are opened. The presence of this operand makes it unnecessary to supply a DOS LBLTYP job control statement in the ELS execution deck. The minimum value that may be specified is 1 which is also the default.

PCT=

specifies the program control table suffix (see Chapter 3.4).

PPT=

specifies the processing program table suffix (see Chapter 3.4).

PRINT=

specifies how print requests from 3270 terminals will be initiated.

YES
— print requests from the ISSUE PRINT command.

PA1
— print requests from the PA1 key and from the ISSUE PRINT command.

PA2
— print requests from the PA2 key and from the ISSUE PRINT command.

PA3
-- print requests from the PA3 key and from the ISSUE PRINT command.

TCT=
specifies the terminal control table suffix (see Chapter 3.4).

TRACE=(size,type,device)
specifies the main or auxilary trace facilities required.

size
is a decimal value specifying the number of entries to be provided in the CICS/DOS/VS-ELS main trace table; the default is 500.

type
specifies the type of trace required:

OFF
no trace facilities required.

MAIN
switches on main trace.

AUX
switches on auxiliary trace.

device
specifies the type of device to be used for auxilary trace output:

TAPE
tape, which must use logical unit SYS009

2314

3330

3340

3350

FBA

VTAM=
specifies whether VTAM is to be used as a terminal access method.

NO

YES

WRKAREA=
specifies the size of the common work area. The range is from 0 (the default) through 3584.

Console Operator Procedures

This section contains guidelines for the console operator. It is recommended that the console operator have at hand all the information presented in this section, a listing of the job control statements required to initiate the system, and the publication CICS/VS Messages and Codes.

Before CICS/DOS/VS-ELS can be executed, it must be installed as described in Chapter 3.3. The system, tables, application programs, and maps must be cataloged. (See also Chapters 3.4 and 3.5.)

CICS/VS STARTUP

The general procedure for activating CICS/VS is:

- Ready all required devices and files.
- Choose the startup job stream to be executed. There may be one job stream if startup is always the same, or several job streams containing alternative label sets if log and dump files are to be saved in the event of an abnormal termination. (These jobstreams could be cataloged in the procedure library.)
- Execute the startup job stream in the correct partition, as shown in Figures 3.6-1, 3.6-2, and 3.6-3.

```
Operator hits the external interrupt key
BG ....... enter ASSGN SYSIN,X'cuu' (sequential device)

If the SYSIN file is correct, CICS/VS will take control.
```

Figure 3.6-1. CICS/VS Startup in a Background Partition using a Sequential SYSIN Device

```
Operator hits ATTN key
AR ....... enter        BATCH F1
F1 ....... enter        ASSGN SYSIN,X'cuu'   (sequential device)

If the SYSIN file is correct, CICS/VS will take control.
```

Figure 3.6-2. CICS/VS Startup in a Foreground Partition using a Sequential SYSIN Device

```
Operator hits ATTN key
AR ....... enter        BATCH F1
F1 ....... enter        // DLBL IJSYSIN,'CICS JOB STREAM'
F1 ....... enter        // EXTENT SYSIN
F1 ....... enter        ASSGN SYSIN,DISK,VOL=CICSCS,SHR (disk
                         drive)

If the SYSIN file is correct, CICS/VS will take control.
```

Figure 3.6-3. CICS/VS Startup in a Foreground Partition using a DASD SYSIN Device

CICS/VS TERMINATION

CICS/VS can be terminated by the master terminal operator or may be abnormally terminated by the operating system. The primary consideration in both instances is to save information on SYSLST, the dump file, and the log files. On an abnormal termination, there is also the possibility that any ISAM files that permit the ADD function might have to be recreated.

If the system is to be restarted immediately and SYSLST, the dump file, and the log files are to be saved, an alternative startup job stream should be used to provide a different set of extents. As soon as feasible, SYSLST should be printed, the dump utility program (DFHDUP) should be used to format and list the dump file, and the data on the old log file should be added to a consolidated log file with a user-written batch program. These files are then available for the next CICS/VS startup. In addition, the trace utility program should be used to print the contents of the auxiliary trace data set if applicable.

Figures 3.6-4, 3.6-5, and 3.6-6 are job control examples of how to print the CICS/VS SYSLST file from disk and the CICS/VS dump file from tape and from disk. Figure 3.6-7 shows the job control statements required to consolidate the log file. Figures 3.6-8 and 3.6-9 are job control examples of printing the auxiliary trace data set from tape and disk.

```
// JOB PRINT                                PRINT CICS PRINTER FILE
// ASSGN SYS001,DISK,VOL=CICSCS,SHR         (See Note)
// ASSGN SYS002,X'00E'                      (See Note)
// DLBL OUTPR,'CICS PRINTER FILE'           (See Note)
// EXTENT SYS001                            (See Note)
// EXEC xxxxxxxx                            USER-WRITTEN UTILITY
/&

Note: The device must match the requirements of the
      user-written program.
```

Figure 3.6-4. Job Stream Used to Print the CICS/VS SYSLST File

```
// JOB CICSDUP
// ASSGN SYS011,X'180'                    TAPE UNIT ADDRESS
// EXEC DFHDUP
   DEVICE=TAPE
/*
/&

Note:  SYS011 is required for the tape unit.
```

Figure 3.6–5. Job Stream Used to Print a CICS/VS Dump File from Tape

```
// JOB CICSDUP
// ASSGN SYS011,DISK,VOL=CICSCS,SHR          DASD DEVICE
// DLBL DTF3330,'DUMP FILE',0,SD,CISIZE=2048  DUMP FILE-ID EXAMPLE
// EXTENT SYS011
// EXEC DFHDUP
   DEVICE=3330
/*
/&

Note:  The file name is DTF3330 for a 3330, DTF2314 for a 2314,
       DTF3340 for a 3340, DTF3350 for a 3350 or DTFFBA for an
       FBA device.

The file identification is the same as that used for the primary
or alternative dump file when CICS/VS was active.
```

Figure 3.6–6. Job Stream Used to Print a CICS/VS Dump File from DASD

```
// JOB LOGFILE
// DLBL LOG,'LOG FILE'                      LOG FILE-ID EXAMPLE
// EXTENT SYS011
// ASSGN SYS011,DISK,VOL=CICSCS,SHR         DASD DEVICE
// DLBL CONSLOG,'CONSOLIDATED LOG FILE'     (See Note)
// EXTENT SYS012                            (See Note)
// ASSGN SYS012,X'151'                      (See Note)
// EXEC xxxxxxxx                 USER-WRITTEN CONSOLIDATE PROGRAM
/*
/&

Note:  The file and the symbolic unit must match the requirements
       of the user-written program.
```

Figure 3.6–7. Job Stream Used to Consolidate the Log File

```
// JOB PTRACE
// ASSGN SYS009,X'180'                    TAPE UNIT ADDRESS
// TLBL DFHAUXT,'AUXILIARY TRACE'
// EXEC DFHTUP,SIZE=30K
/*
/&

Note:  SYS009 must be assigned to the tape unit
```

Figure 3.6–8. Job Stream Used to Print the Auxiliary Trace Data Set from Tape

```
// JOB PTRACE
// ASSGN SYS009,DISK,VOL=CICSCS,SHR
// DLBL DFHAUXT,'AUXILIARY TRACE',0,SD
// EXTENT SYS009,...
// EXEC DFHTUP,SIZE=30K
   DEVICE=3330 (See Note)
/*
/&

Note:  DEVICE may be 2314, 3330, 3340, 3350, or FBA.
       It is always required for disk data sets
```

Figure 3.6–9. Job Stream Used to Print the Auxiliary Trace Data Set from DASD

PROCESSOR CONSOLE AS CICS/VS TERMINAL

The processor console can be used as a CICS/VS terminal for the CICS/VS master terminal transaction (CEMT), which may be entered to control and monitor the CICS/VS system. If this is required, the terminal control table must have a TRMIDNT=CNSL entry (see Chapter 3.4).

To initiate the CEMT transaction on the console:

- In the background partition, press the external interrupt key (see Figure 3.6–10).
- In a foreground partition, press the attention key. In reply to the attention routine statement 'READY FOR COMMUNICATIONS', enter MSG Fn (Fn is the partition number: F1, F2, F3, or F4) to request communication with the CICS/VS partition (see Figure 3.6–11).

  When the system responds with the partition number (BG or Fn), enter the CEMT transaction code and data. Up to 80 characters of upper or lower case data, including the transaction code, may be entered. The cancel key may be used to cancel any entered data. CICS/VS processes the transaction and sends the response back to the console, preceded by the partition number (BG or Fn).

```
Operator hits the external interrupt key
BG ....... enter CEMT PERFORM SHUTDOWN
BG DFH1701 - C.I.C.S. IS BEING TERMINATED
BG DFH1799 - TERMINATION OF CICS IS COMPLETE

Note: Use of master terminal transaction (CEMT) is discussed in
      the section "Master Terminal Operations" later in this
      chapter.
```

Figure 3.6—10. Console as a Terminal in the Background Partition

```
Operator hits ATTN key
AR
AR ....... enter MSG F1
F1 ....... enter CEMT PERFORM SHUTDOWN
F1 DFH1701 - C.I.C.S. IS BEING TERMINATED
F1 DFH1799 - TERMINATION OF CICS IS COMPLETE
```

Figure 3.6—11. Console as a Terminal in a Foreground Partition

As the console is shared by all partitions, the lines of output from a CICS/VS transaction may be interspersed with messages from other partitions or from VSE.

The display operator console (DOC) is limited to six lines displayed at a time. Any transactions entered on the display operator console should not require more than six lines of terminal output.

Master Terminal Operations

The master terminal transaction enables the user to modify CICS/VS control parameters, and the operational status of terminals, files, programs, and transactions while CICS/VS is operating. It is also the means for terminating CICS/VS. The master terminal transaction may be entered at any terminal, but its use should be limited to the master terminal operator by sign-on/sign-off and a security key. The master terminal operator must have a thorough understanding of the CICS/VS system and each of the master terminal functions that are described in this section.

A "Master Terminal Operator's Guide" should be established and a reference copy kept at the master terminal. The guide should contain this section, "Master Terminal Operations", and the following:

A list of terminals grouped according to physical location.
The terminal identification and the terminal priority, as specified by the TRMIDNT and TRMPRTY operands in the terminal control table, should be included.

A list of transactions grouped according to functional area.
A brief description and the transaction identification, as specified in the TRANSID operands of the program control table, should be included.

A list of programs grouped according to program identification and function.
The program identification is the one specified in the PROGRAM keyword of the processing program table. Each program should include a list of all transactions that invoke it, so that, if a program is disabled, all of its transactions can also be disabled.

A list of files sequenced by file name as specified in the DATASET operand of the file control table.
A list of all transactions that cause access to each file should be included so that, if a file is disabled, all of its transactions can be disabled.

A list of transient data files sequenced by destination identification, as specified in the DESTID operand of the destination control table.
A list of all transactions that cause access to each destination should be included so that, if a destination is disabled, all of its transactions can be disabled. In an online file-maintenance environment, it is critical that all transactions posting to a log file be disabled if the file is disabled for any reason.

The master terminal transaction is initiated by entering a CICS/VS request which begins with the transaction identifier "CEMT". It continues executing until the operator terminates it by pressing program function key 3 (PF3).

This chapter contains a simplified list of the requests required to control and monitor CICS/VS components. The complete range of CEMT requests is described in the CICS/VS Operator's Guide.

If an operator makes an error when typing a request, the transaction will prompt him for a correct request. The master terminal operator for an entry level system is not expected to use this prompting facility, and should clear the screen and retype the request.

PROGRAM FUNCTION KEYS

When CEMT is executing, the lower part of the display contains a list of IBM 3270 PF keys, and descriptions of the key functions.

CEMT responds to seven function keys. The keys and their functions are as follows:

PF1
Help. Produces a list of PF keys and their functions.

PF3
End session. The operator terminates the CEMT transaction. Other CICS/VS transaction codes can then be entered.

PF7
Scroll up half.

PF8
Scroll down half.

PF9

Expand messages. If several messages have been generated in response to a request, the operator can display all of them by pressing PF key 9.

PF10

Scroll up.

PF11

Scroll down.

SCROLLING

A plus (+) sign at the beginning or end of a data display indicates that there is more data above or below the current display. Scrolling back reveals data above, and scrolling forward reveals data below. A whole screen (PF10 or PF11) or half a screen (PF7 or PF8) can be scrolled.

TASKS

List All CICS/VS Tasks

```
CEMT INQUIRE TASK
```

Abnormally Terminate a Task

```
CEMT SET TASK PURGE
```

Inquire about the maximum number of CICS/VS tasks.

```
CEMT INQUIRE MAXTASKS
```

Change the maximum number of CICS/VS tasks

```
CEMT SET MAXTASKS(x)        (x = new maximum number of tasks)
```

Notes:

1. The maximum number of tasks cannot be raised above that specified in the MXT startup override, described earlier in this chapter.
2. INQUIRE or SET requests for MAXTASKS will produce a list of system parameters and their status. The ELS operator should ignore all parameters except MAXTASKS.

TRACE PROGRAM

Turn Trace On

```
CEMT SET TRACE ON
```

Turn Trace Off

```
CEMT SET TRACE OFF
```

DUMP PROGRAM

Initiate Partition Dump

```
CEMT PERFORM SNAP PARTITION
```

Initiate Formatted Dump

```
CEMT PERFORM SNAP FORMAT
```

(For further information, see the CICS/VS Problem Determination Guide.)

CICS/VS SHUTDOWN

To shut CICS/VS down, the master terminal operator enters one of the following commands:

Let Tasks End - No Dump

```
CEMT PERFORM SHUTDOWN
```

Let Tasks End - Dump

```
CEMT PERFORM SHUTDOWN DUMP
```

Immediate Shutdown - No Dump

```
CEMT PERFORM SHUTDOWN YES
```

Immediate Shutdown - Dump

```
CEMT PERFORM SHUTDOWN YES DUMP
```

Note: Operators should not normally perform immediate shutdowns.

TERMINALS

In the following commands, "termid" represents the 4-character name which identifies a particular terminal.

Inquire About One or More Individual Terminals

```
CEMT INQUIRE TERMINAL(termid[,...])
```

Inquire About all Terminals

```
CEMT INQUIRE TERMINAL
```

Set One or More Individual Terminals in Service

```
CEMT SET TERMINAL(termid[,...]) INSERVICE
```

Set all Terminals in Service

```
CEMT SET TERMINAL INSERVICE
```

Set a Terminal Out of Service

```
CEMT SET TERMINAL(termid[,...]) OUTSERVICE
```

Place one or more individual terminals into transaction mode

```
CEMT SET TERMINAL(termid[,...]) TTI NOATI
```

Place one or more individual terminals into transceive mode

```
CEMT SET TERMINAL(termid[,...]) TTI ATI
```

Transaction Mode: A terminal in transaction mode receives no messages without a terminal request.

Transceive Mode: A terminal in transceive mode receives messages that are sent automatically, as well as those initiated by a terminal request.

Receive Mode: A terminal in receive mode may receive messages but cannot send them. This mode is equivalent to NOTTI ATI.

Note: Sign-off GOODNIGHT places the terminal in receive mode to prevent unauthorized use. The terminal must be returned to transaction or transceive mode by the master terminal operator before it can be used again.

CONTROL UNITS

If a control unit is placed out of service, no terminal on the control unit can be used. To reference a control unit, specify the TERMID of any terminal on the control unit.

Inquire About One or More Individual Control Units

```
CEMT INQUIRE CONTROL(ctid[,...])
```

Place a Control Unit in Service

```
CEMT SET CONTROL INSERVICE
```

Place a Control Unit Out of Service

```
CEMT SET CONTROL OUTSERVICE
```

LINES

If a line is placed out of service, no terminal on that line can be used. To reference a line, specify the TERMID of any terminal on the line.

Inquire About One or More Individual Lines

```
CEMT INQUIRE LINE(lineid[,...])
```

Place a Line in Service

```
CEMT SET LINE INSERVICE
```

Place a Line Out of Service

```
CEMT SET LINE OUTSERVICE
```

DATA BASE FILES

Data bases may be enabled or disabled when required. To prevent transaction ABENDs, all transactions that cause reference to a file should be disabled before the file is disabled, and enabled after the file has been enabled.

'fileid' is the file name specified in the DATASET keyword of the file control table.

Inquire About One or More Individual Data Base Files

```
CEMT INQUIRE DATASET(fileid[,...])
```

Enable a Data Base File

```
CEMT SET DATASET(fileid[,...]) ENABLED
```

Disable a Data Base File

```
CEMT SET DATASET(fileid[,...]) DISABLED
```

DUMP DATA SET

CICS/VS dump data sets may be opened or closed as required. If there are two CICS/VS dump data sets, the SWITCH function closes the current dump data set and opens the other one.

Switch Dump Data Sets

```
CEMT DUMP SWITCH
```

Open Dump Data Sets

```
CEMT SET DUMP OPEN
```

Close Dump Data Sets

```
CEMT SET DUMP CLOSED
```

TRANSIENT DATA FILE

The DESTID is the destination identification specified in the DESTID keyword of the destination control table. All transactions referencing the log file should be disabled before it is disabled, and enabled after it has been enabled.

Enable a Transient Data Queue

```
CEMT SET QUEUE(quident) ENABLED
```

Disable a Transient Data Queue

```
CEMT SET QUEUE(quident) DISABLED
```

PROGRAMS

Programs should be disabled if they are not working properly.

The pgrmid is the program identification specified in the PROGRAM keyword of the processing program table. One or more pgrmids may be specified whenever "pgrmid[,...]" is shown.

Inquire About One or More Programs

```
CEMT INQUIRE PROGRAM(pgrmid[,...])
```

Inquire About all Programs

```
CEMT INQUIRE PROGRAM
```

Disable One or More Programs

```
CEMT SET PROGRAM(pgrmid[,...]) DISABLED
```

Enable One or More Programs

```
CEMT SET PROGRAM(pgrmid[,...]) ENABLED
```

Enable all Programs

```
CEMT SET PROGRAM ALL ENABLED
```

TRANSACTIONS

Transactions should be disabled whenever the files or programs that are used by the transactions are disabled. When the files or programs are returned to service (ENABLE), the transaction should be enabled. tranid is the transaction identification specified by the TRANSID keyword of the program control table. One or more tranids may be entered whenever "tranid=tranid[,...]" is specified.

Disable One or More Individual Transactions

```
CEMT SET TRANSACTION(tranid[,...]) DISABLED
```

Enable One or More Individual Transactions

```
CEMT SET TRANSACTION(tranid[,...]) ENABLED
```

Enable all Transactions

```
CEMT SET TRANSACTION ALL ENABLED
```

User Terminal Operations

Terminal operators must be trained in the use of CICS/VS sign-on/sign-off procedures if these are required by the installation. They must also be trained in the use of installation transactions, and must have access to all documentation related to terminal operations and transaction execution.

It is suggested that each installation establishes a run book, which should include:

- Sign-on/Sign-off Procedure
- 3270 Print Procedure
- CWTO Transaction
- User Transaction Requirements
- Transaction Error Recovery Procedures
- CICS/VS Termination Procedure

Each of these items is described below.

SIGN-ON/SIGN-OFF PROCEDURE

The terminal operator must enter the correct password and operator name, as established by the system programmer in the sign-on table. Only transactions that are assigned one of the current operator security keys may be initiated after sign-on.

Sign-on Procedure

Enter: CSSN

Receive:

```
PLEASE SUPPLY PERSONAL DETAILS

NAME=

PASSWORD=

NEWPASSWORD=
```

Sign-on is performed by positioning the cursor to the "NAME" and "PASSWORD" entry fields, and typing the operator's name and password. The NEWPASSWORD field is ignored by ELS.

The password entry field is a "dark" field. This means that text types into the field remains invisible. Care should be taken when typing into the field, as the text cannot be checked visually.

A name can be up to 20 characters long, and a password can be up to eight characters long.

Note: The name must be entered exactly as it appears in the sign-on table, including blanks, commas, and other punctuation.

Sign-Off Procedure

```
Enter:    CSSF              -    Sign off, leave in transaction mode
          CSSF GOODNIGHT    -    Sign off, change to receive-only mode

Receive:  DFH3506  SIGN OFF IS COMPLETE
```

If CSSF GOODNIGHT is used, the terminal remains signed off until the master terminal operator returns it to the transaction or transceive mode.

3270 PRINT PROCEDURE

To print a 3270 display screen, the operator must depress the PA (program access) key specified during system initialization (or PA1 if defaulted). When the data has been captured by the CPU or the remote COPY has been completed, the keyboard is freed and the data is printed, if, for a BTAM system, there is a printer available on the same control unit and line group. For a VTAM system, the data may be printed on an alternative printer on a different control unit/line group.

Each terminal attached to an IBM 3274 or 3276 control unit has a local print key which can cause the contents of the screen to be printed, without host interaction.

CWTO TRANSACTION

The WRITE TO OPERATOR function gives the terminal operator the ability to send messages to the processing unit console operator.

To send the message, the terminal operator enters the transaction identification "CWTO" followed by the message.

```
Enter:    CWTO   -message to be sent-

Receive:  MESSAGE HAS BEEN SENT
```

Notes:

- The ENTER key can be used to insert blank lines in the message.
- The terminal and operator identifications are appended to the message.

The write-to-operator transaction can be canceled by entering:

```
Enter:    CANCEL as the last six characters of the input.

Receive:  TERMINATED BY OPERATOR
```

If the terminal operator initiates the CWTO transaction without a message text, the program becomes conversational:

```
Enter:    CWTO

Receive:  ENTER MESSAGE

Enter:    good morning          (message to be sent)

Receive:  MESSAGE HAS BEEN SENT
```

USER TRANSACTION REQUIREMENTS

The terminal operator run book should include a list of all valid transaction codes with a narrative describing the transaction, input data, CICS/VS messages, and corrective action to be taken.

TRANSACTION ERROR RECOVERY PROCEDURES

The terminal operator run book should describe the actions to be taken by an operator if the terminal becomes inoperable.

All critical add, delete, or update transactions should be listed.

The recovery procedures to be followed, if a terminal becomes inoperative while a critical transaction is being processed, should be described.

Chapter 3.7. Servicing

Proper servicing of the CICS/VS system is essential. IBM distributes fixes to CICS/VS program errors as an APAR (Authorized Program Analysis Report) fix and later to customers on a preventive service (PUT) tape.

Authorized Program Analysis Report (APAR) Fixes

An APAR fix is intended for corrective, not preventive, maintenance. When a CICS/VS error is found, an APAR describing the problem in detail is submitted to IBM by the customer via the IBM support center. IBM analyzes the problem and, if valid, provides a fix which is tested by the customer who reported the problem. A record of the problem and its solution is kept by the IBM support center in case the same problem occurs at other customers' installations. APAR fixes should only be applied to a particular system if the problem exists, or if it is reasonably certain that the problem will occur.

APAR fixes are applied to the distributed version of CICS/VS using the VSE/Advanced Functions Maintain System History Program (MSHP). For information on using MSHP refer to the VSE/Advanced Functions Maintain System History Program User's Guide.

DFHGEN Macro

The APAR response lists the entry level system programs that have to be reassembled (if any) after a source fix has been applied. To reassemble the programs, use the DFHGEN macro, which has to be coded and assembled with the CICS/DOS/VS primary source library assigned. In the example shown after the list of operands, the COPY DFHSTSG copies in the source used by CICS/VS to generate the ELS core-image library (known as Stage 1 source).

The assembly will punch out a jobstream, which must be run with both the primary source library and the entry level system core-image library permanently assigned.

The format of the DFHGEN macro is as follows:

```
DFHGEN  MOD=(program[,suffix]...[,program[,suffix]] ...),          *
        TYPE=type,                                                 *
        CICS=ELS/FULL,                                             *
        DLI=YES/NO,                                                *
        VSAM=YES/NO,                                               *
        VTAM=YES/NO
```

Operands of DFHGEN Macro

program
— the name of the entry level system program — without the "DFH" — to be reassembled (for example, KCP)

suffix
— a two-character suffix, which may be used to distinguish between different versions of one entry level system program. (Certain programs have a number of versions; the choice between these is determined at startup time by the startup overrides specified.)

The recommended procedure is to reassemble all versions of a program when applying a fix, in which case "ALL" should be specified instead of a two-character suffix; but you may, if you so wish, reassemble only the versions you are currently using by specifying the appropriate suffixes. Appendix A provides, under "Core Image Library," a list of those modules having more than one version, and relates suffixes to startup overrides. You should specify "ALL" for any program not listed in Appendix A as having more than one version.

TYPE=
— determines whether an APAR fix is to be applied permanently or temporarily, or removed from the core image library. Valid parameters are:

PERM
— corresponds to a PER APAR response. It puts a new version of the module into the core image library without saving the old one.

TEMP
— corresponds to a fix test APAR response. It puts a new version of the module into the core image library, after first saving the old module by renaming it.

REM
— removes a previously applied fix by deleting the latest version and reinstating the superseded version, which must have been saved by an earlier run specifying TYPE=TEMP.

FIX
— makes a previous TEMP fix into a permanent fix by deleting the saved superseded version from the core image library.

CICS=
This operand does not have to be coded, as the default is the correct option for the ELS user.

DLI
— indicates whether DL/I support is to be generated for the affected modules. This will override the Stage 1 specification.

VSAM=
— indicates whether VSAM support is to be generated for the affected modules. This will override the Stage 1 specification.

VTAM=
: — indicates whether VTAM support is to be generated for the affected modules. This will override the Stage 1 specification.

The following example shows the use of the DFHGEN macro.

```
//JOB  FIXKCP
//EXEC ASSEMBLY
       DFHGEN VTAM=NO,VSAM=NO,DLI=NO,                              *
              MOD=(KCP,ALL)
       COPY DFHSTSG
       END
/*
/&
```

Preventive Service (PUT) Tape

The preventive service (PUT) tape, distributed to customers on a monthly basis, contains maintenance fixes for all programs for which the customer holds a license, and will include CICS/DOS/VS service. These fixes are tested and should be installed as soon as possible. The contents of the PUT tape should be examined to ensure that all APAR fixes that were previously applied are included. Any APAR fixes not included should be reapplied after the PUT tape has been installed. A customer should apply the current PUT tape immediately if he intends to change his data base structure, code new applications, or introduce new system functions.

Preventive service tapes are installed using the VSE/Advanced Function Maintain System History Program (MSHP). For information in using MSHP refer to the VSE/Advanced Function Maintain System History Program User's Guide.

A record of APAR fixes and PUT tapes applied to the CICS/VS system can be automatically generated in the MSHP history file.

Chapter 3.8. Performance

During the system design process, the required level of performance of the system should be considered, and the system designed with these performance criteria in mind. After a system has been successfully designed and implemented, its performance can be monitored and the system tuned to ensure that efficiency is maintained as changes in the workload occur.

CICS/DOS/VS-ELS is designed to give optimum performance when only one transaction is being processed in the system at any one time. If this criterion is not met, performance may be degraded, and in more complex situations CICS/DOS/VS may out-perform CICS/DOS/VS-ELS.

This chapter discusses the performance aspects of CICS/DOS/VS-ELS. The main topics are:

- Design criteria for CICS/DOS/VS-ELS
- Operating system design from a performance viewpoint
- CICS/VS system design from a performance viewpoint
- Application program design
- Use of CICS/VS Statistics.

Design Criteria

In an online system such as CICS/VS, certain areas of code will be executed thousands of times at comparatively regular intervals. For any installation, there is a fairly well-defined set of program code, data areas, and control blocks that are always in use. This is called the "working set" of the system. To achieve the best response, enough real storage should be available to accommodate the working set. The aim of the system designer should be to ensure that the working set is small enough to reside in real storage, allowing VSE to page in non-working set parts of CICS/VS as necessary.

Consider carefully the implications of frequent use of optional CICS/VS functions, because they will increase the size of the working set. The following is a list of application program functions that utilize code that is not considered part of the minimum CICS/DOS/VS-ELS working set.

- Temporary Storage
- Interval Control
- Dump Control
- Trace Control
- Intrapartition Transient Data
- VSAM for FCT entries

- VTAM support for terminals

Operating System Design from a Performance Viewpoint

The performance objectives in generating a VSE system are to ensure that the supervisor generated occupies the minimum amount of real storage and executes with the minimum pathlengths.

Detailed discussion of the sizes and functions of the various VSE facilities is contained in the DOS/VS System Generation manual. The discussion in this chapter is limited to those functions that could have significant effect in a CICS/VS environment.

SUPERVISOR GENERATION

Job Accounting (JA)

Job accounting can increase processor usage by up to one tenth, so the need to use this facility should be carefully evaluated against cost.

Fast CCW Translate (FASTTR)

Specification of FASTTR=YES can result in a significant reduction in processor usage in heavily loaded systems. However, for the entry level system user with a lightly loaded system, FASTTR is unlikely to provide such a reduction.

Rotational Position Sensing (RPS)

This option is useful only when the channel is heavily loaded and causing delays - an unusual condition in small systems. Selection of rotational position sensing carries an overhead in that the FCT has to contain two ISAM logic modules instead of one, thus increasing the working set by 5K.

TPBAL COMMAND

If CICS/DOS/VS-ELS and a batch partition are running concurrently, TPBAL can be used to bias the system towards the CICS/VS partition. This will release page frames for use by the CICS/VS partition, and reduce the amount of page I/O, thus improving CICS/VS response.

The TPBAL command issued from the processor console is used to tell the operating system to recognize TPIN and TPOUT macros. These macros are issued by CICS/VS when it becomes active and when it goes into a long WAIT. TPIN will deactivate the batch partition, if there is one, and TPOUT will reactivate it.

It is recommended that the TPBAL command is issued when a CICS/VS partition is running concurrently with a batch partition. Do not use TP balancing (that is, issue TPBAL=0) when CICS/VS is running in a dedicated processing unit or when there is sufficient real storage to allow the system to run without causing significant page I/O.

BASIC TELECOMMUNICATIONS ACCESS METHOD (BTAM)

The BTAM load module and control blocks form part of the CICS/VS Terminal Control Table and are loaded into the CICS/VS partition. These areas are fixed in real storage while there is any I/O outstanding. Except at the very lowest transaction rate, this effectively means the areas are permanently fixed. These storage requirements are explicitly listed in the DOS/VS System Generation manual.

If BTAM is used it is necessary to make an allowance when deciding on the number of copy blocks required (specified by the BUFSIZE operand of the VSTAB macro instruction during VSE system generation). As a general guide, four blocks should be allocated for each simultaneous I/O operation to remote devices and three blocks for local devices. The number of simultaneous operations cannot exceed the number of communication lines.

DATA BASE ACCESS METHODS (ISAM AND VSAM)

There are two main goals to aim for in making efficient use of the access method chosen. The first is to try and ensure that the physical I/O operations are performed as efficiently as possible. Two important aims in achieving this are:

1. To equalize the I/O rate (at peak load) across the various I/O devices, by careful positioning of the various files. This helps reduce the time spent waiting for a particular device.

2. Position the datasets on a particular device so that seek times, which are proportional to the distance the disk arm has to move, are minimized. (For example, datasets consistently used by the same transaction could be grouped together.)

The second major consideration is to minimize of the number of physical I/O operations necessary. This is not really practicable for VSAM ESDS files; but for ISAM or VSAM KSDS files, there are two ways of minimizing the number of physical I/O operations.

1. The most effective way of decreasing the number of physical I/O operations that occur for any particular dataset is to maintain the most heavily used part of the dataset in main storage. Where an indexed access method is used, (ISAM, VSAM KSDS), this can be achieved by having in-core indexes.

2. The other method is to increase the size of the physical record in a data set with respect to the logical record (blocking factor).

In both these cases however, there is a trade-off. In the first case, more real storage is necessary; lack of storage required may cause page faults, which could offset the advantage gained. In the second case, high blocking factors could result in increased reponse time and channel utilization.

In general, these considerations apply to all the I/O access methods, including those for DL/I. The parameters that control the changes described above are set either when a dataset is created, or else when CICS/VS tables are created. (These are discussed further in the section that follows.)

CICS/VS System Design from a Performance Viewpoint

This section outlines the performance impact of options that can be specified when CICS/VS tables are generated, and of override parameters that can be specified at system initialization.

DFHFCT TYPE=DATASET, (FOR ISAM files)
INDAREA=,INDSIZE=

Use the cylinder index in virtual storage. This will probably save an I/O operation on all calls.

DFHFCT TYPE=DATASET, (for VSAM files)
BUFSP=, BUFND=, BUFNI=, STRNO=

These parameters are used to determine the amount of buffer space (BUFSP) to be given to VSAM for processing the dataset. The amount required depends on the type and number of concurrent accesses to the dataset, the Control Interval size for the data and index buffers, and to some extent the amount of real storage available.

As a general rule, for KSDS files BUFSP should equal

CI(DATA) x BUFND + CI(INDEX) x BUFNI

where CI(DATA) and CI(INDEX) are the appropriate Control Interval sizes, and BUFND and BUFNI are the number of data buffers and index buffers. The number of index buffers allocated should be at least equal to STRNO, but performance can be improved by specifying BUFNI as (at least) equal to STRNO plus the number of index levels. This will maintain the top level index in core.

If additional buffers are allocated, they will be used by VSAM in an attempt to optimize performance. This means that all the allocated buffer space may become part of the working set, so check that any optimization by VSAM as a result of having access to more buffers is not offset by degradation caused by increased page fault activity.

For ESDS files, no indexes are required, and the number of data buffers should be specified as at least STRNO + 1.

The value of STRNO should be chosen carefully, because if there are several concurrent requests to a file, CICS/VS will automatically queue the number of requests that are in excess of STRNO. This will degrade performance. On the other hand, excessive allocations for STRNO will cause extra demands for working set storage, possibly resulting in greater degradation due to additional page faults. However, if the storage available is limited, it is often preferable to increase BUFNI so that the time the STRING is held by VSAM is reduced, which will reduce the number of simultaneous requests that VSAM is handling, and this means that STRNO can be reduced.

Alternatively, rather than trying to optimize the buffer requirements for each file separately, use the VSAM shared resources facility. This will usually allow much better utilization of buffer space and much easier tuning, since the peaks and troughs of activity on different datasets will tend not to coincide exactly, and a much smoother variation of activity with time will occur. Remember however, that if sufficient real storage is available, better performance can be obtained by using non-shared resources for VSAM KSDS files.

DFHTCT TYPE=LINE (for BTAM terminals)

The system programmer can impact the response of the system by the way he sets up the Terminal Control Table (TCT). The following recommendations are based on the way CICS/VS scans the terminal table and the way BTAM polls terminals for input. Some consideration of the terminal device is also necessary.

Generally speaking the closer the entry to the start of the TCT the better the response. For multipoint lines, note that BTAM will still poll the line for input after receiving an input, and this will continue until CICS/VS, through BTAM, issues a command to tell the line to stop sending data and to be prepared to receive data. This causes contention between the terminals for use of the line, with longer delays for heavier loads. The best system throughput will be achieved by setting up the TCT so that the heaviest loaded line is the first entry.

This principle should be followed only if all responses are acceptable. It is possible that some of the lightly loaded lines or terminals (usually point-to-point) may always get an unacceptably bad response. In this case some of the lightly loaded lines should be moved higher up the TCT. This will alleviate the immediate response problem but may cause a bottleneck on the heavier loaded lines, in which case the line capacity should be increased.

SEQUENCE OF ENTRIES IN CICS/VS TABLES

The order of entries in CICS/DOS/VS-ELS tables can affect performance because the tables are searched sequentially. PPT entries should be ordered by frequency of usage. Map entries should be coded with the entry of the first program that uses them; not only does this ensure a short scan for frequently used programs, but putting maps and programs together avoids fragmentation of storage and unnecessary paging. PCT entries can be coded in any order because the PCT is automatically merged with the PPT when the entry level system is executed. Entries in other tables should be ordered by frequency of usage.

STORAGE REQUIREMENTS FOR EXECUTION (COMMAND LEVEL) DIAGNOSTIC FACILITY (EDF)

The virtual storage required for EDF, the diagnostic facility described in Chapter 4.1, are as follows:

1. The total size of programs, tables, and maps is approximately 35K bytes.

2. The working storage required for each concurrent EDF session is approximately 3K bytes.

3. The temporary storage required for each concurrent EDF session is approximately 14K bytes (assuming that a maximum of 10 displays are being remembered).

Application Program Design

The entry level system is designed to be most efficient with non-conversational application programs. If conversational programs are essential, however, they should be designed to use the pseudo-conversational technique. (See "Pseudo-conversational Programming" in Chapter 2.1.)

Separate high-use code and low-use code into separate programs or pages to improve the working set and paging characteristics.

Use of CICS/VS Statistics

CICS/VS system statistics are automatically recorded by the entry level system management programs and are printed when CICS/DOS/VS-ELS is shut down. The statistics can be used by the system programmer to help monitor the performance of the entry level system. Analysis of CICS/VS system statistics will provide information about usage of the CICS/DOS/VS-ELS system components, and occurrence of limit conditions. Details of the statistics produced by the entry level system are contained in the CICS/VS Operator's Guide.

Part 4. Program Checkout

Introduction

Thorough testing of application programs is essential to a successful CICS/VS installation. If the sample programs described in this manual are used as programming guides, errors can be minimized, but testing remains a necessity. It is recommended that all testing be in a CICS/VS test partition to prevent program errors from abnormally terminating the online CICS/VS system. If facilities for a CICS/VS test partition are not available, programs should be tested during separately reserved test hours to prevent destruction of files or uncontrolled abnormal termination of an operating CICS/VS system.

This part contains two chapters. The first describes the execution (command-level) diagnostic facility (EDF), which enables the programmer to carry out comprehensive online tests of application programs. The second chapter describes the trace and dump facilities.

Chapter 4.1. Execution (Command Level) Diagnostic Facility

The Execution (Command Level) Diagnostic Facility (EDF) enables an application programmer to test an application program online without making any modifications to the source program or the program preparation procedure. The facility intercepts execution of the program at certain points and displays relevant information about the program at these points. Also displayed are any screen layouts sent by the user program, so that the programmer can converse with the application program during checkout just as a user would on the production system.

EDF runs as a CICS/VS transaction. It is started by a transaction identifier or PF key named in the PCT by the system programmer; also, the PPT needs to specify the programs and maps that are used by EDF. EDF uses temporary storage and BMS. It can be used only from a 3270 terminal with a screen width of 80 columns and a screen depth of 24 lines or more.

Functions of EDF

During execution of a transaction in debug mode, EDF intercepts the execution of the application program at the following points:

1. At transaction initialization:
   - After the EXEC interface block (EIB) has been initialized; but
   - Before the application program is given control.
2. At the start of the execution of every command:
   - After the initial trace entry has been made; but
   - Before the requested action has been performed.
3. At the end of the execution of every command (except ABEND, XCTL, and RETURN):
   - After the requested action has been performed; but
   - Before the HANDLE CONDITION mechanism is invoked; and
   - Before the response trace entry is made.
4. At program termination
5. At normal task termination
6. When an ABEND occurs
7. At abnormal task termination

At these points of interception, EDF displays the current status. In addition:

1. At point 1, EDF displays the values of the fields in the EIB.

2. At point 2, EDF displays the command, including keywords, options, and argument values. The command is identified by transaction identification, program name, the hexadecimal offset within the program, and, if the program was translated with DEBUG option, the line number of the command as given in the translator source listing.

3. At point 3, EDF displays the same as at point 2, plus the response from command execution.

4. At points 6 and 7, EDF displays the values of the fields in the EIB and the following items:

   - The abend code;
   - If the abend code is ASRA (that is, a program interrupt has occurred), the PSW at the time of interrupt, and the cause of the interrupt as indicated by the PSW;
   - If the PSW indicates that the instruction giving rise to the interrupt is within the application program, the offset of that instruction.

The user is also given the ability to display any of the following:

- The values of the fields in the EIB and DIB (DL/I interface block)
- The program's working storage in hexadecimal and character form.
- The last ten commands executed, including all argument values, responses, and so on.
- The hexadecimal contents of any address location within the CICS/VS partition.

At any of these points of interception, the user is allowed to interact with the application in the following ways:

- If the current command is being displayed before it is executed, the user can modify any argument value by overtyping the value that is displayed on the screen. Alternatively, the user can cancel execution of the command (that is, convert it to a null operation by overtyping the command verb with NOOP or NOP).
- If the current command is being displayed <u>after</u> it has been executed, the user can modify any argument value or the response code by overtyping the <u>displayed</u> value or response with the <u>required</u> value or response.
- The user can modify the program's working storage and most fields of the EIB and DIB.
- The user can switch off debug mode (except at point 2) and continue running the application normally. Alternatively, the user can force an abend.
- The user may request that commands are not displayed until one or more of a set of specific conditions is fulfilled. These conditions may be:
  - A specific named command is encountered
  - An exceptional condition occurs for which the system action is to raise ERROR.

- A specific exceptional condition occurs
- The command at a specific offset or on a specific line number (assuming the program had been translated with the DEBUG option) is encountered
- An abend occurs
- The task terminates normally
- The task terminates abnormally
- Any DL/I error status occurs
- A specific DL/I error status occurs

Security Rules

To invoke EDF, the user must have a security key that matches the security key defined for EDF in the PCT. In addition, to test a particular transaction, the EDF user must have a security key that matches the security key for that transaction. If this condition is not satisfied, the EDF session is terminated immediately.

Installing EDF

To ensure that EDF is available on the test system, the system programmer must make one group entry in the PPT and at least one entry in the PCT (see Chapter 3.4).

Invoking EDF

EDF can be run on the same terminal as the transaction requiring checkout, provided that the application does not use extended attributes, or on a different terminal.

For same-terminal checkout, EDF can be invoked either by:

1. Using transaction CEDF or
2. Using the appropriate PF key, if one has been defined for EDF.

The transaction requiring checkout can then be started.

For different-terminal checkout, EDF is invoked by using the transaction identifier CEDF with an argument that specifies the four-character identifier (as defined in the TRMIDNT operand of the DFHTCT TYPE=GPENTRY macro) of the terminal on which the transaction requiring checkout is being run. For example:

```
CEDF L77A
```

If a transaction is already running on that terminal, EDF will associate itself with that transaction; otherwise it will associate itself with the next transaction started at that terminal.

The transaction identifier CEDF may be entered from a formatted screen, in which case the effect is the same as pressing the PF key; that is, the terminal at which CEDF is entered is put into EDF mode. (No message is issued, so that the formatted screen remains intact.)

The full format of the command to initiate or terminate an EDF session is:

```
CEDF [terminal-id] [ , {ON|OFF} ]
```

If the teminal identifier is omitted, the terminal at which the CEDF transaction is initiated is assumed. If ON and OFF are omitted, ON is assumed.

Using EDF Displays

An example of a typical EDF display is given in Figure 4.1-1.

```
TRANSACTION: CMNU   PROGRAM: XDFHINST   TASK NUMBER: 0000115   DISPLAY:  00
STATUS: COMMAND EXECUTION COMPLETE
EXEC CICS SEND
MAP('XDFHCMA')
FROM(' N....F..j&K..Y&.....K.......m...H...DK.zX&.........*...'...)
TERMINAL
ERASE
OFFSET:X'0003EE'                        EIBFN=X'1804'
RESPONSE: NORMAL                        EIBRCODE=X'000000000000'
ENTER: CONTINUE
PF1 : UNDEFINED          PF2 : SWITCH HEX/CHAR     PF3 : END EDF SESSION
PF4 : SUPPRESS DISPLAYS  PF5 : WORKING STORAGE     PF6 : USER DISPLAY
PF7 : SCROLL BACK        PF8 : SCROLL FORWARD      PF9 : STOP CONDITIONS
PF10: PREVIOUS DISPLAY   PF11: UNDEFINED           PF12: ABEND USER TASK
```

Figure 4.1-1. Typical EDF Display

The five lines at the foot of the screen provide a menu indicating the effect of the ENTER and PF keys for that particular display. If the terminal does not have PF keys, the same effect can be obtained by positioning the cursor under the required instruction on the screen and pressing the ENTER key. The cursor can be correctly positioned by using the tab keys.

Although the menu may change from one display to another, no function will move from one key to another as a result of a menu change.

If the ENTER key is pressed while the cursor is not positioned within the menu, the effect is the same as if the cursor had been positioned under the ENTER key action.

EDF uses the line immediately above the menu to display messages to the user.

The number at the top right of the screen indicates the current display number; it is possible to step back through previous displays, which are numbered -1, -2, and so on. The user cannot overtype this number to address a display. Up to ten displays are remembered and can be redisplayed later.

Argument values can be displayed in character or hexadecimal format. If character format is requested, numeric arguments are shown in signed numeric character format. Each argument value is restricted to one line of the display; if the value is too long, only the first few bytes are displayed, followed by "..." to indicate that the value is incomplete. If the argument is displayed in hexadecimal format, the address of the argument is also displayed. This enables the user to display the argument value in full by requesting a display of that location and scrolling if necessary.

The user can overtype any screen area at which the cursor stops as the tabbing keys are pressed, such as the response field. Thus, for example, the response can be changed from "NORMAL" to "ERROR" or some other exceptional condition, so as to test the program's error handling at this point in the program. A list of areas that can be overtyped is given later under "Overtyping EDF Displays."

If the screen is accidentally altered, it can be restored by pressing the CLEAR key.

When a non-function EDF screen is being displayed (for example, one that shows the EIB or working storage), the meaning of the ENTER key is given as "CURRENT DISPLAY". This means that pressing the ENTER key will return the screen to where it was before examination of the other screens started. The same applies when a user screen is being displayed as a result of an EDF function, such as the USER DISPLAY key. However, if a user screen is displayed on instruction from the application program, use of the ENTER key causes EDF to continue execution.

Terminal Sharing

When both EDF and the user transaction are sharing the same terminal, EDF restores the user transaction's display at the following times:

- when the transaction requires input from the operator
- when the transaction's display is changed
- at the end of the transaction
- when EDF displays are suppressed
- when USER DISPLAY is requested.

Thus, when a SEND command is followed by a RECEIVE command, the display sent by the SEND command appears twice — once when the SEND command is executed, and again when the RECEIVE command is executed. It is not necessary to respond to the SEND command, but if a response is made, EDF will remember it and redisplay it when the screen is restored for the RECEIVE command. The response passed to the transaction is that which is made to the RECEIVE command.

When EDF restores the transaction display, it does not sound the alarm or affect the keyboard in the same way as the user transaction. The effect of the user transaction options will be seen when the SEND command is executed, but not when the screen is restored.

When EDF restores the transaction display, it locks the keyboard until the transaction issues a RECEIVE command, at which time EDF frees the keyboard.

If the EDF session is terminated part way through the transaction, EDF restores the screen with the keyboard locked if the last send/receive to the terminal was in fact a RECEIVE command; otherwise, the keyboard is unlocked. This will usually, but not always, match the normal behavior of the transaction.

Function Key Meanings

The following list explains the effect of each function in the menu:

ABEND USER TASK
: — terminates the task. The display will then contain the message "ENTER ABEND CODE AND REQUEST ABEND AGAIN." After entering the code at the position indicated by the cursor, the user must request this function again to actually abend the task with a transaction dump identified by the specified code. If the user enters "NO" the task will be abended without a dump.

  This function cannot be used if an abend is already in progress or the task is terminating.

CONTINUE
: causes the user transaction to continue unless the screen has been modified. If the screen has been modified, EDF redisplays the screen, with the modifications incorporated.

CURRENT DISPLAY
: — displays the screen that was being displayed before the user started examining other displays, such as remembered displays, unless the screen has been modifed. If this is the case, the modified screen is redisplayed.

DIB DISPLAY
: shows the contents of the DL/I Interface Block (DIB).

EIB DISPLAY
: — shows the contents of the EXEC Interface Block and COMMAREA (see Appendix B for a description of the fields in the EIB).

END EDF SESSION
: — ends the debugging session, and takes the terminal out of debug mode. The user transaction continues.

NEXT DISPLAY
: — used when examining displays, to step on to the next remembered display. Repeated use stops at the current display, when the "next display" key is no longer available.

OLDEST DISPLAY
: — displays the earliest remembered screen, that is, the one currently having the lowest display number.

PREVIOUS DISPLAY
— shows the latest remembered display. Repeated use stops at the earliest remembered display. Further use merely causes the earliest remembered display to be redisplayed.

RE-DISPLAY
— redisplays the current screen to enable confirmation that a change entered on the screen has in fact taken place as expected.

If an invalid change is made, it will be ignored and a message will be displayed indicating that an error has occurred.

REGISTERS AT ABEND
— displays storage containing the values of the registers in the event of an ASRA abend. The layout of the storage is as follows:

a. PSW at abend (8 bytes)

b. Register values (0 through 15)

In some (very rare) cases, when a second program check occurs in the system before EDF has captured the values of the registers, this function will not appear on the menu of the abend display. If this happens, a second test run will generally prove to be more informative.

REMEMBER DISPLAY
— places a display that would not normally be remembered, such as an EIB display, in the memory. (Normally, only the command displays are remembered.) The memory can hold up to ten displays. All pages associated with the display are remembered (and can be scrolled when recalled) except for storage displays where only the page currently displayed is remembered.

SCROLL BACK
scrolls a command or EIB display backwards. A plus sign (+) against the first option or field indicates there are more options or fields preceding.

SCROLL BACK FULL
scrolls a working storage display a full screen backwards, displaying lower addresses.

SCROLL BACK HALF
scrolls a working storage display half a screen backwards, displaying lower addresses.

SCROLL FORWARD
scrolls a command or EIB display forwards. A plus sign (+) against the last option or field indicates there are more options or fields following.

SCROLL FORWARD HALF
scrolls a working storage display half a screen forwards, displaying higher addresses.

SCROLL FORWARD FULL
scrolls a working storage display a full screen forwards, displaying higher addresses.

STOP CONDITIONS
— displays a skeleton menu with which the user can specify one or more conditions that will cause EDF to stop the user transaction, and start redisplaying commands, after displays have been suppressed by the SUPPRESS DISPLAYS function (see below). These functions are used to reduce the amount of operator intervention required to check out a program that is partly working.

The transaction can be stopped under the following conditions:

- when a specified type of command is reached
- when a specified exceptional or error condition occurs during execution of a command
- when a specified offset or line is reached
- at transaction abend
- at normal task termination
- at abnormal task termination.

Note that the line number, which will be available on the source listing if the program was translated using the DEBUG option, must be specified exactly as it appears on the listing, including leading zeros, and must be the line on which a command starts.

Note also that the offset specified must be the offset of the BALR instruction corresponding to the command.

The correct line can be determined easily from the translator output listing. The offset can be determined from the code listing produced by the assembler or compiler.

For transactions that contain DLI commands, the qualifier CICS on the command line can be overtyped with DLI to specify a DLI command. Also, the transaction can be stopped when a specified error status, or any error status, occurs.

SUPPRESS DISPLAYS
— suppresses all EDF displays until the next stop condition occurs.

SWITCH HEX/CHAR
— switches the display between hexadecimal and character representation. This is a mode switch; all subsequent displays will stay in the chosen mode until the next time this key is pressed. This switch has no effect on previously-remembered displays, stop condition displays, and working storage displays.

UNDEFINED
— means that this key is not available with this type of display.

USER DISPLAY
— shows what the user would see if the terminal was not in EDF mode. Hence, this function is usable only for same-terminal checkout.

WORKING STORAGE
— displays the program's working storage, in a form similar to that of a dump listing, that is, in both hexadecimal and character representation. When this key is used, two additional scrolling keys are provided, and other PF keys allow the EIB (and the DIB if a DL/I command has been processed by EDF) to be displayed.

The meaning of "working storage" depends on the programming language of the application program:

COBOL
— all data storage defined in the WORKING-STORAGE section of the program.

PL/I
— the dynamic storage area (DSA) of the main procedure.

Assembler Language
— the storage defined in the current DFHEISTG DSECT.

RPG II
the entire RPG II module, including both data and code.

Note that working storage starts with a standard format save area, that is, registers 14-12 are stored at offset 12 and register 13 at offset 4.

Working storage can be changed at the screen; either the hexadecimal section or the character section may be used. Also, the ADDRESS field at the head of the display can be overtyped with a hexadecimal address; storage starting at that address will then be displayed when ENTER is pressed. This allows any location in the partition to be examined. Further information on the use of overtyping is given later under "Overtyping EDF Displays."

If the storage examined is not part of the user's working storage (which is unique to the particular transaction under test), the corresponding field on the screen is inhibited to prevent the user from overwriting storage that can affect more than one invocation of the program.

If the initial part of a working storage display line is blank, the blank portion is not part of working storage. This can occur because the display is doubleword aligned.

At the beginning and end of a task, working storage is not available. In these circumstances, EDF generates a blank storage display so that the user can still examine any storage area in the partition by overtyping the address field.

Overtyping EDF Displays

As mentioned above, certain areas of a EDF display can be overtyped. These areas can be identified by use of the tab keys — the cursor stops only at fields that can be overtyped (excluding fields within the menu).

- The verb of a command, such as the "SEND" in "EXEC CICS SEND", can be overtyped with "NOOP" or "NOP" before execution; this supresses execution of the command. When the screen is redisplayed with NOOP, the original verb line can be restored by erasing the whole verb line using the ERASE EOF key.

- Any argument value can be overtyped, but not the keyword of the argument. Overtyping must be in the same representation, hexadecimal or character, as the original field, and must not extend beyond the argument value displayed. Any modification that is not overtyping of the displayed value is ignored (no diagnostic message being generated). When an argument is displayed in hexidecimal format, the address of the argument location is also displayed.

- Numeric values always have a sign field, which can be overtyped with a minus or a blank only.

- Every field in the EXEC Interface Block, except the function code and the task number, can be overtyped; this also applies to the COMMAREA field.

- The response field can be overtyped with the name of any exceptional condition, including ERROR, that can occur for the current function, or with the word "NORMAL". The effect when EDF continues will be that the program will take whatever action has been prescribed for the specified response.

- The EIBRCODE field, when displayed as part of the EXEC Interface Block, can be overtyped with any desired bit pattern. This does not apply when the EIBRCODE field is part of a command display.

Note: When a field representing a data area of a program is overtyped, the entered value is placed directly into the application program's storage. On the other hand, before execution of a command, when a field representing a data value (which may possibly be a constant) is overtyped, a copy of the field is used; thus, other parts of the program that might use the same constant for some unrelated purpose will not be affected by the change. If, for example, the map name is overtyped before executing a SEND MAP command, the map actually used temporarily is the map with the entered name; but the map name displayed on response will be the original map name. (The "previous display" key can be used to display the map name actually used.)

When an argument is to be displayed in character format, some of the characters may not be displayable (including lowercase characters). EDF replaces each non-displayable character by a period. When overtyping a period, the user must be aware that the storage may in fact contain a character other than a period, the user may not overtype any character with a period — if this is done, the change is ignored and no diagnostic message is issued. Similarly, when a value is displayed in hexadecimal format, overtyping with a blank character is ignored and no diagnostic message is issued.

When storage is displayed in both character and hexadecimal format and changes are made to both, the value of the hexadecimal field will take precedence should the changes conflict; no diagnostic message is issued.

If invalid data is entered, the result is as follows, regardless of the action requested by the user:

- the invalid data is ignored;
- a diagnostic message is displayed;
- the keyboard is locked;
- the alarm is sounded if the terminal has the alarm feature;

The user may then reset the keyboard and reenter the data correctly.

Note: EDF does not translate lowercase characters to uppercase. If uppercase translation is not specified for the terminal in use, the user must take care to enter only uppercase characters.

Checking Out Pseudo-Conversational Programs

On termination of the task, EDF displays a message saying that the task is terminated and prompting the user to specify whether or not debug mode is to continue into the next task. This is to allow realistic debugging of pseudo-conversational programs (see "Program Control" in Chapter 2.1). If the terminal came out of debug mode between the tasks involved, each task would start with fresh EDF settings, and the user would not be able, for example, to display screens remembered from previous tasks.

Program Labels

Some commands, such as HANDLE CONDITION, require the user to specify a program label. The form of the display program labels depends on the programming language in use:

- For COBOL, a null argument is displayed; for example, ERROR( )
- For PL/I, the address of the label constant is displayed; for example, ERROR (X'001D0016')
- For Assembler Language, the offset of the program label is displayed; for example, ERROR (X'00030C')
- For RPG II, the address of the label is displayed; for example, ERROR (X'00014038')

If no label value is specified on a HANDLE CONDITION command, EDF displays the condition name alone.

Chapter 4.2. Trace and Dump Control

In addition to the execution diagnostic facility, CICS/DOS/VS-ELS provides trace and dump facilities that can be used to help test and debug application programs.

ABEND Command

```
ABEND          ABCODE(name)
```

The ABEND command is used to abnormally terminate a task, and optionally provide a dump of the virtual storage areas related to the task. This function is useful in program testing, or in live programs where it might be used to handle any unmanageable condition.

The ABCODE(name) option can be used to specify a 1 to 4 character name to identify the dump. This name appears at the beginning of the dump listing.

ENTER Command

```
ENTER          TRACEID(data-value)
               [FROM(data-area)]
```

This command is useful in the debugging of programs because it helps to provide a trace of the processing path through a transaction. The trace function allows a programmer to insert trace commands at appropriate points in a program to determine the routines used, and the sequence of execution. Whenever a trace command is encountered during execution of a program, an entry is made in a trace table which identifies that trace command, and thus the processing path. The trace command can also name a data field to be stored in the table entry. For example, this field might contain intermediate results or switch settings from the transaction work area. After the table entry is made, control returns to the application program following the trace command.

In addition, whenever a CICS/VS command is executed, an entry is automatically made in the table, identifying the type of command, such as WRITE. The trace entries made by CICS/VS commands are often adequate to determine the processing path without inserting additional trace commands.

To print the trace table, dump commands should also be inserted as needed along with the trace commands. After inserting trace and dump commands, programs must be compiled. When the programs have been

debugged, the trace and dump commands should be removed, and the programs recompiled.

The trace identification (TRACEID operand) must be specified as a halfword binary value in the range 0 to 199. This value is stored in byte 0 of the entry in the trace table, to identify the trace command.

The FROM(data-area) option can be used to specify the name of an 8-byte field to be stored in bytes 8-15 of the trace table.

DUMP Command

```
DUMP          DUMPCODE(data-value)
```

The DUMP command causes all CICS/VS areas related to a transaction to be written to a sequential dump file (disk or tape) for later printing by the CICS/VS Dump Utility Program (see Chapter 3.6).

The DUMPCODE operand is used to specify a 4-character code to identify the dump. This code is printed at the beginning of the dump listing.

DUMP commands can be inserted at appropriate points in a program as a debugging aid. The program is then compiled. When a dump command is executed a transaction dump is written to the dump dataset. The transaction dump contains storage areas, the application program(s) being used, certain CICS/VS control blocks and tables (in particular, the trace table), and user data areas.

Certain error conditions, such as recoverable I/O errors occurring during normal operation, may also be recorded through a dump. These dump commands are a permanent part of the program.

The DUMP command provides a printout of the trace table and may therefore be useful in conjunction with the trace facility.

Trace Table Analysis

The trace table shows the flow of processing through a transaction by creating an entry every time a CICS/VS macro instruction is executed in a CICS/VS control program, or whenever an EXEC CICS command is encountered in an application program. The programmer can follow the path of the transaction through the program by using the trace table.

Further information about the trace table will be found in the CICS/VS Problem Determination Guide.

Dump Analysis

Three types of dump are available to the entry level system programmer: an individual transaction dump, a CICS/DOS/VS-ELS formatted dump, and a CICS/DOS/VS-ELS partition dump.

A transaction dump is produced when a CICS/VS transaction terminates abnormally, or else when an EXEC CICS DUMP command is encountered in a program (see "DUMP Command" earlier in this chapter); it contains only those areas directly related to the transaction itself.

A formatted dump of the CICS/VS system, with details of the contents of the major CICS/VS control blocks and data areas at the time of failure, is produced if the CICS/DOS/VS-ELS system terminates abnormally. A CICS/DOS/VS-ELS partition dump can be produced optionally when CICS/VS terminates abnormally (see "Startup Overrides" in Chapter 3.6 for details).

Transaction dumps are intended primarily for use by the application programmer for debugging purposes. Formatted dumps and CICS/DOS/VS-ELS partition dumps are intended for the system programmer; the entry level system application programmer should not normally need to refer to them.

Detailed information about analyzing CICS/VS dumps is contained in the CICS/VS Problem Determination Guide.

Part 5. Appendixes

Appendix A. Entry Level System Summary

This appendix is a summary of the components furnished with the CICS/DOS/VS entry level system.

Source Statement Libraries

The contents of the primary source statement library as supplied by IBM are as follows:

A.sublibrary
— CICS/DOS/VS programs and source needed to prepare tables, applications, and maps.

C.sublibrary
— source needed to prepare COBOL application programs.

E.sublibrary
— edited macros

P.sublibrary
— source needed to prepare PL/I application programs.

R.sublibrary
— source needed to prepare RPG II application programs.

Z.sublibrary
— contains the following source books:

DFHSTMP
— source used to generate the sample maps.

DFHSTLM
— source used to generate the VSE logic modules in the private relocatable library.

DFHSTAP
— the source used to generate the sample application programs (some of which are in the CICS/DOS/VS-ELS core image library, others of which are only on the full system library).

Relocatable Library

Certain items, such as access method logic modules, must be available on a relocatable library when CICS/DOS/VS programs or tables are being link edited. The private relocatable library used to generate the private core image library contains:

- The object modules with names of the form, DFHEXMxx, DFHEAMxx, DFHECMxx, DFHEPMxx, DFHERMxx, DFHEXMF4, and DFHEXMS4 for building the CICS/VS command-language translators.
- The following object modules:

- IJDVZZIW (PRMOD) (note 1)
- IJGVOEZZ (SDMODVO) (note 1)
- IJGFOEZZ (SDMODFO) (note 1)
- IJHZLGZZ (ISMOD) (note 1)
- BTMODH# (note 5)
- BTMODL# (notes 1 and 2)
- BTMODR# (notes 1 and 2)
- BTMODM# (notes 1 and 2)
- BTML32T (notes 1 and 2)
- BTMR32T (notes 1 and 2)
- BTMLR32T (notes 1 and 2)
- DFHISMNN
- DFHISMNC
- DFHISMRN
- DFHISMRC
- DFHEAI (note 3)
- DFHEAI0 (note 3)
- DFHECI (note 3)
- DFHEPI (note 3)
- DFHERI (note 3)
- DFHDLX (note 4)
- DFHPHN (note 5)
- DFHPL1I (note 5)

Notes:

1. These modules are needed when CICS/VS tables are link edited.
2. The user may wish to generate other BTAM logic modules, depending on his installation's requirements. The procedure is described in the CICS/VS System Programmer's Guide (DOS/VS).
3. DFHEAI (Assembler language), DFHECI (COBOL), DFHEPI (PL/I), and DFHERI (RPG II) are object modules link edited with application programs that use the command-level interface.
4. These modules are needed when DL/I is used.
5. These modules should be deleted. The entry level system does not need them; they are present because the full CICS/DOS/VS system, with which the library is shared, requires them.

Core Image Library

The following programs in the core image library have more than one version, each associated with a different startup override:

Program	Suffix	Startup Override
DFHDCP	1#	DUMPDEV=TAPE
	2#	DUMPDEV=2314
	3#	DUMPDEV=3330
	4#	DUMPDEV=3340
	5#	DUMPDEV=3350
	F#	DUMPDEV=FBA
DFHFCP	I#	FILE=ISAM
	V#	FILE=VSAM
	1#	FILE=(ISAM,VSAM)
DFHPCP	1#	Used if any of the entries in the PPT (processing program table) specify PGMLANG=COBOL
	2#	Used if no PPT entries specify PGMLANG=COBOL
DFHTCP	1#	BTAM=LOCAL BSCODE=EBCDIC
	2#	BTAM=LOCAL BSCODE=ASCII
	3#	BTAM=REMOTE BSCODE=EBCDIC
	4#	BTAM=REMOTE BSCODE=ASCII
	5#	BTAM=(LOCAL,REMOTE) BSCODE=EBCDIC
	6#	BTAM=(LOCAL,REMOTE) BSCODE=ASCII
DFHTDP	1#	INTRA=NO EXTRA=YES
	2#	INTRA=DAM EXTRA=YES or EXTRA=NO
	3#	INTRA=VSAM EXTRA=YES or EXTRA=NO
DFHTRP	1#	TRACE override includes TAPE
	2#	TRACE override includes 2314
	3#	TRACE override includes 3330
	4#	TRACE override includes 3340
	5#	TRACE override includes 3350
	F#	TRACE override includes FBA
DFHZCP	1#	VTAM=NO
	7#	VTAM=YES
DFHZCX	1#	VTAM=NO
	2#	VTAM=YES

The remaining entry level system programs have only one version in the library.

The Entry Level System Tables

Tables containing "$" or "#" are reserved for CICS/VS.

DFHDCT2#	2314-based Destination Control Table
DFHDCT3#	3330-based Destination Control Table
DFHDCT4#	3340-based Destination Control Table
DFHDCT5#	3350-based Destination Control Table
DFHDCTF#	FBA extrapartition, VSAM intrapartition

```
DFHFCT1#   VSAM File Control Table
DFHFCT2#   2314-based File Control Table
DFHFCT3#   3330-based File Control Table
DFHFCT4#   3340-based File Control Table

DFHPCT1#   Program Control Table for all sample programs
DFHPCT2#   Program Control Table for Assembler sample programs
DFHPCT3#   Program Control Table for COBOL sample programs
DFHPCT4#   Program Control Table for PL/I sample programs
DFHPCT5#   Program Control Table for RPG II sample programs

DFHPPT1#   Processing Program Table for all sample programs
DFHPPT2#   Processing Program Table for Assembler sample programs
DFHPPT3#   Processing Program Table for COBOL sample programs
DFHPPT4#   Processing Program Table for PL/I sample programs
DFHPPT5#   Processing Program Table for RPG II sample programs

DFHTCT1#   3270-local Terminal Control Table
DFHTCT2#   3270-remote Terminal Control Table
DFHTCT3#   3270-local and 3270-remote Terminal Control Table
```

Appendix B. EXEC Interface Block (EIB)

Each task in a command-level environment has an associated control block called the EXEC interface block (EIB). Each application program has automatic access by name to the fields within the task EIB. The EIB contains information that is useful during the execution of an application program, such as the transaction identifier, the time and date, and the cursor position on a display device. Furthermore, the EIB contains information that is helpful when a dump is being used to debug a program.

This appendix lists the fields of the EXEC interface block (EIB). Each application program can access all of the fields in the task's EIB by name but must not change the contents of any of them.

For each field, the contents and format (for each application programming language) are given. (All fields contain zeros in the absence of meaningful information.)

Note: For RPG II, the field names are as listed below except that the "IB" is removed; for example, EIBAID would be EAID for RPG II users.

EIB Fields

EIBAID

contains the attention identifier (AID) associated with the last terminal control or basic mapping support (BMS) input operation from a display device such as the 3270.

```
Assembler:  CL1
COBOL:      PICTURE X(1)
PL/I:       CHAR(1)
RPG II:     character, length 1
```

EIBCALEN

contains the length of the communication area that has been passed to the application program from the last program, using the COMMAREA and LENGTH options. If no communication area was passed, this field contains zeros.

```
Assembler:  H
COBOL:      PICTURE S9(4) USAGE COMPUTATIONAL
PL/I:       FIXED BIN(15)
RPG II:     halfword binary
```

EIBCPOSN

contains the cursor address (position) associated with the last terminal control or basic mapping support (BMS) input operation from a display device such as the 3270.

```
Assembler:  H
COBOL:      PICTURE S9(4) USAGE COMPUTATIONAL
PL/I:       FIXED BIN(15)
RPG II:     halfword binary
```

EIBDATE

contains the date when the task was started (and is updated by ASKTIME requests). The date is in packed decimal form (00YYDDD+).

```
Assembler:  PL4
COBOL:      PICTURE S9(7) USAGE COMPUTATIONAL-3
PL/I:       FIXED DEC(7,0)
RPG II:     4-byte packed decimal
```

EIBDS

contains the symbolic identifier of the last data set referred to in a file control request.

```
Assembler:  CL8
COBOL:      PICTURE X(8)
PL/I:       CHAR(8)
RPG II:     character, length 8
```

EIBFN

contains the encoded representation of the function requested by the last CICS/VS command to be issued by the task (updated when the requested function has been completed). See the section "EIBFN Codes" later in this appendix for a complete list of the codes used in this field.

```
Assembler:  CL2
COBOL:      PICTURE X(2)
PL/I:       CHAR(2)
RPG II:     character, length 2
```

EIBRCODE

contains the CICS/VS response code returned after the function requested by the last CICS/VS command to be issued by the task has been completed. Almost all of the information in this field can be used within application programs by issuing appropriate HANDLE CONDITION commands. See the section "EIBRCODE Codes" later in this appendix for a complete list of the codes used in this field.

```
Assembler:  CL6
COBOL:      PICTURE X(6)
PL/I:       CHAR(6)
RPG II:     character, length 6
```

EIBREQID

contains the request identifier assigned to an interval control request by CICS/VS; this field is not used when the application programmer supplies a request identifier.

```
Assembler:  CL8
COBOL:      PICTURE X(8)
PL/I:       CHAR(8)
RPG II:     character, length 8
```

EIBTASKN

contains the task number assigned to the task by CICS/VS. This number will appear in trace table entries generated while the task is in control.

```
Assembler:  PL4
COBOL:      PICTURE S9(7) USAGE COMPUTATIONAL-3
PL/I:       FIXED DEC(7,0)
RPG II:     4-byte packed decimal
```

EIBTIME

contains the time when the task was started (and is updated by ASKTIME requests). The time is in packed decimal form (0HHMMSS+).

```
Assembler:  PL4
COBOL:      PICTURE S9(7) USAGE COMPUTATIONAL-3
PL/I:       FIXED DEC(7,0)
RPG II:     4-byte packed decimal
```

EIBTRMID

contains the symbolic terminal identifier of the terminal or logical unit associated with the task.

```
Assembler:  CL4
COBOL:      PICTURE X(4)
PL/I:       CHAR(4)
RPG II:     character, length 4
```

EIBTRNID

contains the symbolic transaction identifier of the task.

```
Assembler:  CL4
COBOL:      PICTURE X(4)
PL/I:       CHAR(4)
RPG II:     character, length 4
```

EIBFN Codes

The hexadecimal codes and the corresponding CICS/VS command functions are shown in Figure B-1.

Code		Command Function
02	02	ADDRESS
02	04	HANDLE CONDITION
02	06	HANDLE AID
04	02	RECEIVE
04	18	ISSUE ERASEAUP
04	1C	ISSUE PRINT
06	02	READ
06	04	WRITE
06	06	REWRITE
06	08	DELETE
06	0A	UNLOCK
06	0C	STARTBR
06	0E	READNEXT
06	12	ENDBR
06	14	RESETBR
08	02	WRITEQ TD
08	04	READQ TD
08	06	DELETEQ TD
0A	02	WRITEQ TS
0A	04	READQ TS
0A	06	DELETEQ TS
0E	02	LINK
0E	04	XCTL
0E	08	RETURN
0E	0C	ABEND
10	02	ASKTIME
10	08	START
10	0A	RETRIEVE
10	0C	CANCEL
18	02	RECEIVE MAP
18	04	SEND MAP
1A	04	ENTER
1C	02	DUMP

Figure B–1. EIBFN Codes

EIBRCODE Codes

The EIBRCODE field is coded in conjunction with the first byte of the EIBFN field described in the last section. It is coded so that one or more bits in one of the six bytes of the field have a certain meaning equivalent to a CICS/VS command-level exceptional condition. Note that some of the exceptional conditions are not documented in this manual; they are, however, subsumed by the ERROR condition. For further information see the CICS/VS Application Programmer's Reference Manual (Command Level).

The EIBRCODE codes and their meanings are listed in Figure B–2. Each bit (or group of bits) is listed in hexadecimal format; for example, bit 7 is listed as 01. Some exceptional conditions can occur in combination, so some hexadecimal arithmetic must be done to determine the individual bit patterns in the field if a direct meaning for the contents of a byte is not listed. The EIBRCODE field can also contain information copied from other fields within CICS/VS; these occurrences are shown in the figure.

EIBFN	EIBRCODE		
Byte 0	Byte	Bit(s)	Meaning
02	0	E0	INVREQ
04	0	04	EOF
04	0	10	EODS
04	0	C1	EOF
04	0	C2	ENDINPT
04	0	E1	LENGERR
04	0	E3	WRBRK
04	0	E4	RDATT
04	0	E5	SIGNAL
04	0	E6	TERMIDERR
04	0	E7	NOPASSBKRD
04	0	E8	NOPASSBKWR
04	1	20	EOC
04	1	40	INBFMH
04	3	F6	NOSTART
04	3	F7	NONVAL
06	0	01	DSIDERR
06	0	02	ILLOGIC (Note 1)
06	0	08	INVREQ
06	0	0C	NOTOPEN
06	0	0F	ENDFILE
06	0	80	IOERR (Note 1)
06	0	81	NOTFND
06	0	82	DUPREC
06	0	83	NOSPACE
06	0	84	DUPKEY
06	0	E1	LENGERR
08	0	01	QZERO
08	0	02	QIDERR
08	0	04	IOERR
08	0	08	NOTOPEN
08	0	10	NOSPACE
08	0	C0	QBUSY
08	0	E1	LENGERR
0A	0	01	ITEMERR
0A	0	02	QIDERR
0A	0	04	IOERR
0A	0	08	NOSPACE
0A	0	20	INVREQ
0A	0	E1	LENGERR

EIBFN	EIBRCODE		
Byte 0	Byte	Bit(s)	Meaning
0C	0	E2	NOSTG
0E	0	01	PGMIDERR
0E	0	E0	INVREQ
10	0	01	ENDDATA
10	0	04	IOERR
10	0	11	TRANSIDERR
10	0	12	TERMIDERR
10	0	14	INVTSREQ
10	0	20	EXPIRED
10	0	81	NOTFND
10	0	E1	LENGERR
10	0	FF	INVREQ
14	0	01	JIDERR
14	0	02	INVREQ
14	0	05	NOTOPEN
14	0	06	LENGERR
14	0	07	IOERR
14	0	09	NOJBUFSP
18	0	01	INVREQ
18	0	02	RETPAGE
18	0	04	MAPFAIL
18	0	08	INVMPSZ (Note 2)
18	0	20	INVERRTERM
18	0	40	RTESOME
18	0	80	RTEFAIL
18	0	E3	WRBRK
18	0	E4	RDATT
18	1	10	INVLDC
18	1	80	TSIOERR
18	1	01	OVERFLOW
18	2	04	EODS
18	2	08	EOC
18	2	10	IGREQID
1E	0	04	DSSTAT
1E	0	08	FUNCERR
1E	0	0C	SELNERR
1E	0	10	UNEXPIN
1E	0	E1	LENGERR
1E	1	11	EODS

Notes:
1. When either ILLOGIC or IOERR condition exists during file control operations, further information is provided in the EIBRCODE field as follows:
   For ISAM bytes 1 and 2 = ISAM response
   For VSAM bytes 1 = VSAM return code; byte 2 = VSAM error code
2. When the INVMPSZ condition exists during BMS operations, EIBRCODE byte 3 contains the terminal code character.

Figure B-2. EIBRCODE Codes

Appendix C. CICS/DOS/VS-ELS under VTAM

In general, the contents of this manual apply to the entry level system under both BTAM and VTAM. The differences are:

1. The VTAM=YES startup override parameter is specified during system initialization (see "Startup Override Parameters," in Chapter 3.6). The APPLID startup override parameter is also needed at system initialization unless the APPLID operand is issued with the DFHTCT TYPE=INITIAL macro during table generation (see below).

2. During terminal control table generation, the DFHTCT TYPE=GPENTRY macro cannot be used. Instead, the DFHTCT TYPE=TERMINAL macro must be used (in addition to the DFHTCT TYPE=INITIAL and DFHTCT TYPE=FINAL macros). The DFHTCT macros are fully explained in the CICS/VS System Programmer's Reference Manual. The DFHTCT TYPE=SDSCI and DFHTCT TYPE=LINE macros are not needed. Only the following options apply to the entry level system:

   - DFHTCT TYPE=INITIAL

     ACCMETH=VTAM

     APPLID

     OPNDLIM

     RAMAX

     RAMIN

     RAPOOL

     RATIMES

     SUFFIX

```
-   DFHTCT TYPE=TERMINAL
          ACCMETH=VTAM
          ALTPAGE
          ALTPRT
          ALTSCRN
          CONNECT
          DEFSCRN
          FEATURE
          GMMSG
          OPERID
          OPERPRI
          OPERSET
          PGESTAT
          PRINTTO
          RELREQ
          RUSIZE
          TCTUAL
          TIOAL
          TRANSID
          TRMIDNT
          TRMPRTY
          TRMSTAT
          TRMTYPE
```

Appendix D. Printer Authorization Matrix

The Printer Authorization Matrix allows a 3270 installation employing 3274 controllers to define classes of print devices and to authorize display operators to select printers for local copy operations. Information on local copy operations and on the format of the printer authorization matrix is given in IBM 3270 Information Display System: Component Description.

If 3270 local copy operations are to be supported in a CICS/DOS/VS Entry Level system, the CICS/VS user must prepare a printer authorization matrix and provide a transaction to enable it to be transmitted to the 3274 control unit. This appendix gives details of how this may be done.

DEFINING THE PRINTER AUTHORIZATION MATRIX

Basic Mapping Support provides the most convenient way of defining a printer authorization matrix for the 3274 control unit. A typical map definition is shown in Figure D.1.

The first two rows of the map do not form part of the printer authorization matrix; they are used to present descriptive information to the display operator.

The third row of the map contains a sequential string of attribute characters that uniquely identifies the buffer data that follows as a printer authorization matrix. The required string of attribute characters is X'60', X'C1', X'D4', X'60'. The first character, X'60', is generated by specifying ATTRB=PROT in the DFHMDF macro; the remaining characters are specified in the XINIT operand. Note that each of these characters is preceded by the SF (start field) control character X'1D' to identify it as an attribute byte.

The fourth line of the map identifies a printer connected to address 03, operating in shared mode, and available for local copy operations initiated from the terminals with addresses 00, 01, and 02. Further printers may be defined as required on succeeding lines.

The final line of the map contains the string of attribute characters that identifies the end of the matrix.

```
*************************************************************************
PRAM     DFHMSD TYPE=MAP,                                                *
               MODE=INOUT,                                               *
               CTRL=(FREEKB,FRSET)
PRAM1    DFHMDI SIZE=(12,80)
         DFHMDF POS=(1,1),                                               *
               ATTRB=PROT,                                               *
               LENGTH=79,                                                *
               INITIAL='SCREEN IS FORMATTED TO LOAD PRINT AUTHORIZATION*
                MATRIX'
         DFHMDF POS=(2,1),                                               *
               ATTRB=PROT,                                               *
               LENGTH=79,                                                *
               INITIAL='HOLD DOWN ALT KEY AND PRESS EOF KEY'
         DFHMDF POS=(3,1),                                               *
               ATTRB=PROT,                                               *
               LENGTH=6,                                                 *
               XINIT='1DC11DD41D60'
*        NEXT MACRO DEFINES PRINTER ON PORT 03
*        OPERATING IN SHARED MODE
*        NO DEVICE CLASSES ARE SPECIFIED
*        VALID SOURCE DEVICES ARE ON ADDRESSES 00, 01, AND 02
         DFHMDF POS=(4,1),                                               *
               ATTRB=PROT,                                               *
               LENGTH=51,                                                *
               INITIAL='03JXXXXXXXXXXXXXXX111YYYYYYYYYYYYYYYYYYYYYYYYY*
               YYYY'
         DFHMDF POS=(5,1),                                               *
               ATTRB=PROT,                                               *
               LENGTH=6,                                                 *
               XINIT='1DC51DD51DC4'
         DFHMSD TYPE=FINAL
*************************************************************************
```

Figure D.1. Map Definition for Printer Authorization Matrix

LOADING THE PRINTER AUTHORIZATION MATRIX

The CICS/VS user must write a transaction to transmit the printer authorization matrix to the 3274 controller. If a map such as that shown in figure D.1 is employed, it may be transmitted by a command of the form:

```
EXEC CICS SEND MAPSET('PRAM') MAP('PRAM1') MAPONLY ERASE
```

The transaction must be invoked from a 3276 keyboard-display or from the terminal at port 0 of a 3274 controller. When the matrix is displayed, the terminal operator loads the matrix by holding down the ALT key and pressing the EOF key.

The display must be operating in 80-column mode to load the matrix. This corresponds to alternate screen-size mode for all models of the 3278 displays; the correct mode may thus be selected by coding SCRNSZE=ALTERNATE in the PCT entry for the matrix-load transaction. The ERASE option in the EXEC CICS SEND statements shown previously ensure that the display is set to the correct mode before the map is displayed.

Appendix E. Loading Programmed Symbols

The IBM 3278 and IBM 3279 displays are supplied with two 191-character sets of symbols. They can also contain additional program symbol store for up to six 191-character, user-defined symbol sets. This enables the user to define symbol sets which contain special characters, for example italic lettering or greek symbols.

Although it is possible to load a symbol set from any application program, the user will probably wish to prohibit this, and will allocate each program symbol store to installation standard symbol sets. His start-up procedure should thus include execution of a transaction which loads as many as six symbol sets.

The data to be loaded is stored in a form called a "structured field". The user-written application program which loads the structured fields will use a SEND command of the following form:

```
EXEC  CICS  SEND  FROM(data-area)
                    LENGTH(data value)
                    WAIT
                    STRFIELD
```

FROM
: Indicates the name of the data-area from which the symbol set is to be taken. The format of this data area is defined in the IBM 3270 component description.

LENGTH
: Indicates the length of the data string containing the set of programmed symbols.

WAIT
: Indicates that the application must wait until the set of symbols has been loaded.

STRFIELD
: Indicates that the data is stored in the form of a structured field.

Appendix F. The Facility Error Recognition System

The Facility Error Recognition System (FERS) records BTAM terminal errors, and displays selected error information on request.

FERS is an integral part of CICS/VS-ELS Version 1.5. It will be incorporated in the system at startup, unless the user Specifies FERS=NO in the startup overrides.

FERS is a diagnostic tool, and is used by IBM Customer Engineers to diagnose hardware faults. CICS/VS-ELS users are not expected to use it unaided. Consequently, details of its use are not given in this guide. For further information see the CICS/VS Operator's Guide.

Appendix G. Transferring to a Full CICS/DOS/VS System

The following information is provided as guidance for the entry level system user who wishes to implement a full-function CICS/DOS/VS system.

CICS/VS Generation

A full CICS/DOS/VS system is produced by a two-stage system generation process, which allows the CICS/DOS/VS modules to be tailored to suit the requirements of the individual installation. The distribution volume includes a dump of a private core-image library that contains pregenerated versions of all the full CICS/DOS/VS modules, and these can be used as a starting point for installing the full-function system. Detailed information about generating full CICS/DOS/VS is contained in the CICS/VS System Programmer's Guide (DOS/VS).

Table Generation

CICS/DOS/VS-ELS tables are source code compatible with full CICS/DOS/VS, but need to be reassembled before thay can be used on a full system. Note however, that some options of the PPT and PCT have different defaults in the full system. See the CICS/VS System Programmer's Reference Manual for information about the PPT and PCT.

SYSTEM INITIALIZATION TABLE

For full CICS/DOS/VS, a system initialization table (DFHSIT), containing "default" startup overrides, is required. The list of startup overrides for the full system is not identical to the list for the entry level system. See the CICS/VS System Programmer's Guide (DOS/VS).

The pregenerated system initialization table DFHSIT4$, provided in the private core-image library supplied for the full system, will load a full system that is similar in function to the entry level system. See the CICS/VS System Programmer's Reference Manual for details about the system initialization table.

BMS Maps

BMS maps assembled under the entry level system are object code compatible with full CICS/DOS/VS.

Application Programs

Application programs assembled under the entry level system are object code compatible with full CICS/DOS/VS.

Bibliography

Additional information about CICS/DOS/VS is available in the following IBM CICS/VS publications:

General Information, GC33-0066

System Programmer's Guide (DOS/VS), SC33-0070

System Programmer's Reference Manual, SC33-0069

Diagnosis Reference, LC33-0105

Messages and Codes, SC33-0081

Master Terminal Operator's Reference Summary, SX33-6011

Problem Determination Guide, SC33-0089

Operator's Guide, SC33-0080

Application Programmer's Reference Manual (Command Level), SC33-0077

Application Programmer's Reference Manual (RPG II), SC33-0085

Information about VSE job control and library maintenance is contained in the publication VSE/Advanced Functions System Control Statements, SC33-6095. Other VSE publications are listed in the IBM System/370 Bibliography, GC20-0001.

Information about IBM 3270 terminals is available in the following publications:

Operator's Guide for the IBM 3270 Information Display System, GA27-2742

IBM 3270 Information Display System Component Description, GA27-2749

IBM 3270 Information Display System: Color and Programmed Symbols, GA33-3056

Index

Each page number in this index refers to the start of the paragraph containing the indexed item.

SC33-0086-1

CICS/VS Entry Level System User s Guide Printed in U.S.A. SC33-0086-1

Customer Information Control System/Virtual Storage (CICS/VS)
Entry Level System User's Guide (DOS/VS)

SC33-0086-1

READER'S COMMENT FORM

This manual is part of a library that serves as a reference source for systems analysts, programmers, and operators of IBM systems. This form may be used to communicate your views about this publication. They will be sent to the author's department for whatever review and action, if any, is deemed appropriate. Comments may be written in your own language; use of English is not required.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation whatever. You may, of course, continue to use the information you supply.

Note: *Copies of IBM publications are not stocked at the location to which this form is addressed. Please direct any requests for copies of publications, or for assistance in using your IBM system, to your IBM representative or to the IBM branch office serving your locality.*

Number of your latest Technical Newsletter for this publication. . . . . .

Note: *Staples can cause problems with automated mail sorting equipment. Please use pressure-sensitive or other gummed tape to seal this form.*

— Cut Along Dotted Line —

If you want an acknowledgement, give your name and address below.

Name . . . . . .

Job Title . . . . . Company . . . . . .

Address . . . . . .

. . . . . Zip . . . . . .

Thank you for your cooperation. No postage stamp is necessary if mailed in the U.S.A. (Elsewhere, your IBM representative or IBM branch office will be happy to forward your comments.)

SC33-0086-1

Reader's Comment Form

Fold and tape Please do not staple Fold and tape

No postage necessary if mailed in the United States

BUSINESS REPLY MAIL

FIRST CLASS PERMIT 40 ARMONK, NEW YORK

Postage will be paid by addressee:

International Business Machines Corporation
Department 812HP
1133 Westchester Avenue
White Plains, New York 10604

Fold and tape Please do not staple Fold and tape

IBM®

CICS/VS Entry Level System User's Guide Printed in U.S.A. SC33-0086-1

Customer Information Control System/Virtual Storage (CICS/VS)
Entry Level System User's Guide (DOS/VS)

SC33-0086-1

READER'S COMMENT FORM

This manual is part of a library that serves as a reference source for systems analysts, programmers, and operators of IBM systems. This form may be used to communicate your views about this publication. They will be sent to the author's department for whatever review and action, if any, is deemed appropriate. Comments may be written in your own language; use of English is not required.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation whatever. You may, of course, continue to use the information you supply.

Note: *Copies of IBM publications are not stocked at the location to which this form is addressed. Please direct any requests for copies of publications, or for assistance in using your IBM system, to your IBM representative or to the IBM branch office serving your locality.*

Number of your latest Technical Newsletter for this publication. . . . . .

Note: *Staples can cause problems with automated mail sorting equipment. Please use pressure-sensitive or other gummed tape to seal this form.*

— Cut Along Dotted Line —

If you want an acknowledgement, give your name and address below.

Name . . . . . .

Job Title . . . . . Company . . . . . .

Address . . . . . .

. . . . . Zip . . . . . .

Thank you for your cooperation. No postage stamp is necessary if mailed in the U.S.A. (Elsewhere, your IBM representative or IBM branch office will be happy to forward your comments.)

SC33-0086-1

Reader's Comment Form

Fold and tape Please do not staple Fold and tape

No postage necessary if mailed in the United States

BUSINESS REPLY MAIL

FIRST CLASS PERMIT 40 ARMONK, NEW YORK

Postage will be paid by addressee:

International Business Machines Corporation
Department 812HP
1133 Westchester Avenue
White Plains, New York 10604

Fold and tape Please do not staple Fold and tape

IBM®

CICS/VS Entry Level System User's Guide Printed in U.S.A. SC33-0086-1