SC33-0077-3

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

Application Programmer's Reference Manual (Command Level)

Program Product

Program Numbers 5740-XX1 (CICS/OS/VS)
5746-XX3 (CICS/DOS/VS)

IBM

Fourth Edition (July 1981)

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).

This edition is based on the previous CICS/VS Version 1.5 edition, and changes from that edition are indicated by vertical lines to the left of the changes.

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 pub'ication; if the form has been removed, comments may be addressed eith

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, Englan

IBM may use or distribute any of the informat
believes appropriate without incurring any ot
of course, continue to use the information yo

© Copyright International Business Machines C
1980, 1981

Summary of Amendments — Number 10

Date of Publication: December 3, 1976
Form of Publication: TNL GN26-0887 to GC28-6394-4, -5, -6

IBM DOS COBOL

Maintenance: Documentation

- Minor technical changes and additions have been made to the text.

Summary of Amendments — Number 9

Date of Publication: March 15, 1974
Form of Publication: TNL GN28-1062 to GC28-6394-4

IBM DOS/VS COBOL

New: Programming Features

- SORT-OPTION clause for Sort and Merge Features
- 5425 MFCU Support

Maintenance: Documentation only

Minor technical changes and corrections to update the documentation to Release 2

IBM DOS Full American National Standard COBOL, Versions 2 and 3

Maintenance: Documentation only

- 5425 MFCU support deleted
- Minor technical changes and corrections

Editorial changes that have no technical significance are not noted here.

Specific changes to the text made as of this publishing date are indicated by a vertical bar to the left of the text. These bars will be deleted at any subsequent republication of the page affected.

Date of Publication: October 15, 1973
Form of Publication: TNL GN28-1047 to GC28-6394-4

IBM DOS/VS COBOL

New: Programming Features

- Merge Facility

New: Documentation only

- Miscellaneous File Processing Considerations

Maintenance: Documentation only

Minor technical changes to update the documentation to the initial release level.

IBM DOS Full American National Standard COBOL, Versions 2 and 3

Maintenance: Documentation only

Minor technical changes and corrections.

Editorial changes that have no technical significance are not noted here.

Specific changes to the text made as of this publishing date are indicated by a vertical bar to the left of the text. These bars will be deleted at any subsequent republication of the page affected.

Preface

This manual describes the IBM Customer Information Control System/Virtual Storage (CICS/VS) command-level application-programming interface; it contains introductory and reference information necessary to prepare assembler-language, COBOL, and PL/I application programs, using CICS/VS commands, to execute under either of two IBM program products: CICS/DOS/VS (5746-XX3) or CICS/OS/VS (5740-XX1). It is intended primarily for use by application programmers, but will be useful also for system programmers and system analysts. A knowledge of the concepts and terminology introduced in the Customer Information Control System/Virtual Storage (CICS/VS) Version 1 Release 5: General Information, GC33-0066 is required.

The manual contains the following parts:

- "Part 1. Command-Level Programming" introduces CICS/VS commands and describes the basic facilities that are available to the user. A chapter is included about the command language translator and the options that can be selected to modify the way in which the translator operates.
- "Part 2. Data Base Operations" deals with access to data sets in the user's CICS/VS system either through CICS/VS file control or through DL/I.
- "Part 3. Data Communication Operations" deals with communication with terminals and logical units of the subsystems in the telecommunications network that forms part of the CICS/VS system.
- "Part 4. Control Operations" groups together facilities for controlling the operation of application programs in the CICS/VS system.
- "Part 5. Recovery and Debugging" deals with facilities available for recovery from abnormal termination; monitoring; tracing program operation; and dumping areas of main storage.
- "Part 6. The CICS/VS Built-In Function (BIF DEEDIT) Command" describes the one built-in function available with the command-level interface.
- "Part 7. Appendixes"

  A. EXEC Interface Block.
  B. Translation Tables for the 2980.
  C. CICS/VS Macros and Equivalent Commands.
  D. Sample Programs (ASM).
  E. Sample Programs (COBOL).
  F. Sample Programs (PL/I).
  G. Sample Programs for Distributed Transaction Processing.

Experience in writing programs in assembler language, COBOL, or in PL/I is assumed. No previous experience of CICS/VS is assumed. (Note: in some places in the manual, ASM is used as the abbreviation for assembler language.)

Related publications are listed in the bibliography at the end of this publication.

In this publication, the term **VTAM** refers to ACF/VTAM, to ACF/VTAME (CICS/DOS/VS only), and to the Record Interface of ACF/TCAM (CICS/OS/VS only). The term **TCAM** refers both to TCAM and to the DCB Interface of ACF/TCAM. The term **BTAM** refers to BTAM (CICS/OS/VS only) and to BTAM-ES (CICS/DOS/VS only). For further details of system requirements, refer to the publication CICS/VS General Information.

Summary of Amendments

SUMMARY OF AMENDMENTS FOR VERSION 1 RELEASE 5

This fourth edition (SC33-0077-3) includes information about outboard formatting support for 8100 Information Systems using the DPPX/DPS Version 2.

In addition various editorial and formatting changes, together with minor corrections, have been made throughout.

The third edition (SC33-0077-2) provides information about the new or enhanced features introduced by CICS/VS Version 1 Release 5, as follows:

- Extensions to the intercommunication facilities, offering:
  - Multiregion operation (MRO) -- a new mechanism that allows communication between multiple connected CICS/VS regions within the same processing system without the use of SNA networking facilities.
  - Distributed transaction processing (DTP) -- direct transaction-to-transaction communication across systems. (This facility is not available on MRO.)
  - Intersystem Communication between CICS/VS and IMS/VS.
  - Improved throughput by support of SNA parallel sessions.
- Enhanced master terminal facilities for interactive control of CICS/VS.
- Command-level interface enhancements: an interactive command interpreter, and a new command-level interface with DL/I.
- Security enhancements, including support for an external security manager (for example, the Resource Access Control Facility (RACF) program product).
- Improved monitoring facilities.
- Further device support, including:
  - Additional 3270 support.
  - Use of the OS/VS console as a CICS/VS terminal.
  - Networking of TWX and WTTY terminals through the Network Terminal Option (NTO) program product.
- Usability and serviceability aids, including a new user exit mechanism and facilities in CICS/DOS/VS similar to those provided by the FERS service aid.

Some of the above features are not described in this manual because they do not directly affect the application programmer; for information on these, refer to the other CICS/VS manuals listed in the bibliography.

SUMMARY OF AMENDMENTS FOR VERSION 1 RELEASE 4.1

The technical newsletter (SN33-6242) provides information about the new or enhanced features introduced by CICS/VS Version 1 Release 4.1, as follows:

- LUTYPE4 support
- FBA device support (CICS/DOS/VS only)
- Intersystem communication message performance option.

SUMMARY OF AMENDMENTS FOR VERSION 1 RELEASE 4

The second edition (SC33-0077-1) provides information about the new or enhanced features introduced by CICS/VS Version 1 Release 4, as follows:

- Intersystem Communication
- Data Base Support (Transaction Restart)
- Extensions to Support of the 3270 Information Display System
- Enhancements to the Command Level Interface (Assembler Language and DL/I)
- Execution (Command Level) Diagnostic Facility (EDF)

The appendixes have been extended to include assembler-language sample application programs and a separate appendix has been allocated to each language.

Contents

Figures

Part 1. Command-Level Programming

Chapter 1.1. Introduction to Command-Level Programming

The Customer Information Control System/Virtual Storage (CICS/VS) command-level application-programming interface allows application programmers to request CICS/VS services by means of CICS/VS **commands**. These commands are statements that can be included at appropriate points in an application program. They have a format similar to the statements of the programming language in use.

CICS/VS commands can be used in application programs written in assembler language, COBOL, PL/I, and in RPG II. The commands are essentially the same in each language, differing only in the delimiter used, and, in the case of RPG II only, in the syntax.

Because of its fixed format, RPG II is not included in this manual. Instead, a separate manual is available entitled CICS/VS Application Programmer's Reference Manual (RPG II).

Application programs that include CICS/VS commands are processed by the **command language translator**, which translates the commands into statements, in the language being used, which can then be assembled (or compiled) and link-edited in the usual way. When these application programs are executed, the statements inserted by the translator invoke the **EXEC interface program** (DFHEIP), which provides the service requested by each command by invoking one or more CICS/VS control programs.

In addition to invoking CICS/VS control programs, the EXEC interface program obtains, and provides addressability to, any required areas of storage, such as terminal input/output areas and various work areas which, when no longer required, are released automatically. As a general rule, the application programmer need only select the required function and code the appropriate command. There is normally no need to know about CICS/VS storage areas and control blocks; in those cases when access to such areas is needed, the command-level interface provides a command for this purpose, the ADDRESS command, described in "Chapter 1.6. Access to System Information" on page 29.

STRUCTURE OF THIS MANUAL

This manual consists of several parts, each generally having an introductory chapter and one or more other chapters. The remaining chapters in Part 1 deal with the following topics:

- Command format and argument values used throughout this manual ("Chapter 1.2. Command Format and Argument Values" on page 5)
- Command language translator ("Chapter 1.3. Command Language Translator" on page 9)
- Programming techniques, and restrictions placed on the use of the programming language when CICS/VS commands are used ("Chapter 1.4. Programming Techniques and Restrictions" on page 17)
- Exceptional conditions that can occur during the execution of CICS/VS commands ("Chapter 1.5. Exceptional Conditions" on page 25)
- Access to system information ("Chapter 1.6. Access to System Information" on page 29)
- Execution (command-level) diagnostic facility (EDF) ("Chapter 1.7. Execution (Command-Level) Diagnostic Facility" on page 35)
- Command-level interpreter ("Chapter 1.8. Command-Level Interpreter" on page 45)

Part 2 through 6 of the manual are each concerned with CICS/VS commands that can be discussed as a group:

- Part 2. Data base operations - describes the CICS/VS commands provided for storage and retrieval of data in a data base using CICS/VS file control facilities or using DL/I services.
- Part 3. Data communication operations - describes the CICS/VS commands provided for communication between CICS/VS and the terminals and logical units of the subsystems in the telecommunications network of the CICS/VS system.
- Part 4. Control operations - describes the CICS/VS commands that control the execution of tasks within the CICS/VS system.
- Part 5. Recovery and debugging - describes the CICS/VS commands provided for recovery from abnormal termination, and for error-handling, tracing, and monitoring. Commands are also provided to cause dumping of selected areas of storage for offline analysis.

- Part 6. The CICS/VS Built-In Function (BIF DEEDIT) Command - describes the one built-in function available with the command level interface.

Each of the chapters (other than the introductory chapter) of these parts of the manual has a standard format. The first section of a chapter describes, in general terms, functions of the commands included in the chapter. For each command the following information is presented: the syntax of the command and its associated options; exceptional conditions that can occur; a detailed description of what the command does; and possibly one or more examples showing typical coding of the command. Finally, two lists are given: a list of the options, with their functions, that can be used in any of the commands in the chapter; and a list of the exceptional conditions, and their causes, that can occur during the execution of the commands.

Part 7 contains several appendixes. References to most of these appendixes are included in the text. The last four appendixes provide sample programs that illustrate the use of many of the commands described in the manual. The BMS maps and file record descriptions used by the sample programs are also included.

SYNTAX NOTATION USED IN THIS MANUAL

Throughout this manual, wherever a CICS/VS command is presented, the symbols { }, [ ], |, and ... are used in defining the command format. These symbols are not part of the command and are not coded by the programmer. Their purpose is to indicate how the command may be written, and they should be interpreted as follows:

- Uppercase identifiers and punctuation symbols must be coded exactly as shown.
- Lowercase identifiers indicate that user text should be coded as required. The lower case character "b" is used in some places to indicate a blank.
- Square brackets [ ] are used to indicate that the enclosed identifiers are optional. The less than, and greater than symbols < > are used to replace square brackets in the syntax displays produced by the command-level interpreter. (See "Chapter 1.8. Command-Level Interpreter" on page 45).
- The "or" symbol | is used to separate alternatives.
- Underlining is used to denote that the identifier is the default; that is, the one that will be assumed if no explicit choice is made.
- Braces { } are used to enclose a set of alternatives, one of which must be coded.
- The ellipsis ... denotes that the immediately preceding identifier(s) can be coded repetitively.

To denote, for example, that either GTEQ, or EQUAL, or neither, can be coded (and that GTEQ is the default), the syntax notation would be:

[GTEQ|EQUAL]

Chapter 1.2. Command Format and Argument Values

The purpose of this chapter is to explain the general rules governing the use of the CICS/VS commands that are described in the following chapters.

COMMAND FORMAT

The general format of a CICS/VS command is EXECUTE CICS (or EXEC CICS) followed by the name of the required **function**, and possibly by one or more **options**, as follows:

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

where:

"function" describes the operation required (for example READ),

"option" describes any of the many optional facilities available with each function. Some options are followed by an argument in parentheses, others are not. Options (including those that require arguments) can be written in any order,

"argument" is a value such as "data-value" or "name", as defined later in this chapter.

An example of a CICS/VS command (from "Chapter 2.2. File Control" on page 55) is as follows:

```
EXEC CICS READ INTO(FILEA)
DATASET('FILEA') RIDFLD(KEYNUM) UPDATE
```

The appropriate end-of-command delimiter, described in the next section, must be added.

CODING CONVENTIONS

CICS/VS commands can be included in an assembler-language, COBOL, or PL/I program anywhere that an executable statement can be included.

In assembler language:

- The keyword EXEC must appear in an operator position. The command can be labeled.
- The delimiter between options must be either a blank or a comma, but not both. The appearance of ",b" or ".b" immediately following an option indicates that the rest of the line is a comment.
- The usual continuation conventions apply (non-blank character in column 72, the continuation line to start in column 16).

In COBOL, a command must be delimited with "END-EXEC" as shown in the following example:

```
EXEC CICS ISSUE RESET END-EXEC
```

This delimiter allows a command to be written within a THEN clause.

In PL/I, a command must be delimited with a semicolon as shown in the following example:

```
EXEC CICS ISSUE RESET;
```

In the following chapters, for simplicity, the syntax of each of the commands that can be specified in an application program is presented without the phrase EXEC CICS, without the continuation conventions, and without the end-of-command delimiter (END-EXEC or semicolon).

In the programming examples in the text, the phrase EXEC CICS is added but not the continuation conventions or end-of-command delimiter. When coding commands these must be added as appropriate for the programming language in use.

ARGUMENT VALUES

In the following chapters, the parenthesized argument values that follow options in a CICS/VS command are specified as follows:

- data-value
- data-area
- pointer-value (or ptr-value)
- pointer-ref (or ptr-ref)
- name
- label
- hhmmss

Halfword binary values are generally used for lengths. This restricts the size of data areas to 32,768 bytes. There will usually be no other restriction so, in particular, a communication area (COMMAREA) can be 32,768 bytes long and GETMAIN can be used to obtain such an amount of storage.

The argument values are defined in the following sections.

ARGUMENT VALUES IN ASSEMBLER LANGUAGE

In general, an argument may be either the address of the data or the data itself (in assembler-language terms, either a relocatable expression or an absolute expression).

A relocatable expression must not contain unmatched brackets (outside quotes) or unmatched quotes (apart from length attribute references). Provided this rule is obeyed, any expression may be used, including literal constants, such as =AL2(100), forms such as 20(0,R11), and forms which use the macro replacement facilities.

An absolute expression must be a single term which may be either a length attribute reference, or a self-defining constant.

Care must be taken with equated symbols which should be used only when referring to registers (pointer references). If an equated symbol is used for a length, say, it will be treated as the address of the length and an unpredictable error will occur.

- "data-value" can be replaced by an assembler-language reference to data of the correct type for the argument or by a constant of the correct type for the argument.
- "data-area" can be replaced by an assembler-language reference to data of the correct type for the argument.
- "pointer-value" can be replaced by an assembler-language reference to a register.
- "pointer-ref" can be replaced by an assembler-language reference to a register.
- "name" can be replaced either

  by a character string in quotes

  by an assembler-language reference to a character string with a length equal to the maximum length allowed for the name. The value of the character string is the name to be used by the argument.
- "label" can be replaced by any program label or address constant.
- "hhmmss" can be replaced by a self-defining decimal constant or an assembler-language reference to a field defined as PL4. The value must be of the form 0HHMMSS+ where HH represents hours from 00 through 99, MM represents minutes from 00 through 59, and SS represents seconds from 00 through 59.

Many commands involve the transfer of data between the application program and CICS/VS. In most cases, the length of the data to be transferred must be provided by the application program. However, if a data area is specified as the source or target, it is not necessary to provide the length explicitly, because the command language translator will generate a default length value of L'data-area.

Although the DESTIDLENG, FROMLENGTH, KEYLENGTH, LENGTH, PFXLENG, TOLENGTH, or VOLUMELENG options are shown as required options in the syntax for a command, these options are always optional in an assembler-language program which specifies a data area (except in the case of the ENQ and DEQ commands). Length values cannot be defaulted if the SET option is specified in a command.

ARGUMENT VALUES IN COBOL

- "data-value" can be replaced by any COBOL data name of the correct data type for the argument or by a constant that can be converted to the correct type for the argument. The data type can be specified as being one of the following:

  halfword binary - PIC S9(4) COMP

  fullword binary - PIC S9(8) COMP

  character string - PIC X(n) where n is number of bytes
- "data-area" can be replaced by any COBOL data name of the correct data type for the argument. The data type can be specified as being one of the following:

  halfword binary - PIC S9(4) COMP

  fullword binary - PIC S9(8) COMP

  character string - PIC X(n) where n is number of bytes

  In cases where the data type is unspecified, the data area can refer to an elementary or group item.
- "pointer-value" can be replaced by the name of any BLL (base locator for linkage) cell, or by any COBOL data name which contains a copy of such a pointer in a BLL cell.
- "pointer-ref" can be replaced by the name of any BLL (base locator for linkage) cell.
- "name" can be replaced either

  by a character string in quotes (that is, a nonnumeric literal)

by a COBOL data-area with a length equal to the maximum length allowed for the name. The value in the data area is the name to be used by the argument.

- "label" can be replaced by any COBOL paragraph name or a section name.
- "hhmmss" can be replaced by a decimal constant or by a data name of the form PIC S9(7) COMP-3. The value must be of the form 0HHMMSS+ where HH represents hours from 00 through 99, MM represents minutes from 00 through 59, and SS represents seconds from 00 through 59.

ARGUMENT VALUES IN PL/I

- "data-value" can be replaced by any PL/I expression that can be converted to the correct data type for the argument. The data type can be specified as being one of the following:

  halfword binary - FIXED BIN(15)

  fullword binary - FIXED BIN(31)

  character string - CHAR(n) where n is number of bytes

  If the data value is specified as halfword binary, the data value is converted, if necessary, to FIXED BIN(15). "Data-value" includes "data-area" as a subset.

- "data-area" can be replaced by any PL/I data reference which is ALIGNED and has the correct data type for the argument. The data type can be specified as being one of the following:

  halfword binary - FIXED BIN(15)

  fullword binary - FIXED BIN(31)

  character string - CHAR(n) where n is number of bytes

  If the data type is unspecified, the data area can refer to an element, array, or structure; the reference must be to connected storage, for example, FROM(P->STRUCTURE) LENGTH(LNG).

  If data, that is not in varying-length string format, is read into a varying-length string, the length bytes at the beginning of the varying-length string will be corrupted.

- "pointer-value" (which includes "pointer-ref" as a subset) can be replaced by any PL/I expression that can be converted to POINTER.
- "pointer-ref" can be replaced by any PL/I reference of type POINTER ALIGNED.
- "name" can be replaced either

  by a character string in quotes (that is, a literal constant); or

  by a PL/I expression or reference whose value can be converted to a character string with a length equal to the maximum length allowed for the name. The value of the character string is the name to be used by the argument.

- "label" can be replaced by any PL/I expression whose value is a label. Program labels are always passed by value, not by reference.
- "hhmmss" can be replaced by a decimal constant or an expression that can be converted to a FIXED DECIMAL(7,0) value. The value must be of the form 0HHMMSS+ where HH represents hours from 00 through 99, MM represents minutes from 00 through 59, and SS represents seconds from 00 through 59.

If the UNALIGNED attribute is added to the ENTRY declarations generated by the CICS/VS translator by a DEFAULT DESCRIPTORS statement, data-area or pointer-reference arguments to CICS/VS commands must also be UNALIGNED.

Many commands involve the transfer of data between the application program and CICS/VS. In most cases, the length of the data to be transferred must be provided by the application program. However, if a data area is specified as the source or target, it is not necessary to provide the length explicitly, because the command language translator will generate a default length value of either STG(data-area) or CSTG(data-area) as appropriate.

Although the DESTIDLENG, FROMLENGTH, KEYLENGTH, LENGTH, PFXLENG, TOLENGTH, or VOLUMELENG options may be shown as required options in the syntax for a command, these options are always optional in a PL/I program which specifies a data area (except in the case of the ENQ and DEQ commands). Length values cannot be defaulted if the SET option is specified in a command.

Chapter 1.3. Command Language Translator

The command language translator accepts as input a source program, written in assembler language, COBOL, or PL/I, in which CICS/VS commands have been coded, and produces as output an equivalent source program in which the commands have been translated into statements in the language of the source program. At execution time, these statements invoke the EXEC interface program, which accepts the arguments passed by the call from the application program, sets up the parameters in the CICS/VS control blocks, and passes control to the appropriate CICS/VS facility.

The translator is executed in a separate job step. The job step sequence for preparing an application program is translate - assemble (or compile) - link-edit. Cataloged procedures are supplied to assist the user; refer to the appropriate CICS/VS System Programmer's Guide for details. The translator requires a region or partition of 96K bytes.

There are three separate translators, one for assembler language, one for COBOL, and one for PL/I. The translators are each provided in two versions, one for VSE and one for OS/VS. The VSE version reads its input from SYSIPT, produces its output (the translated source program) on SYSPCH, and writes the source listing, error messages and so on, on SYSLST. The OS/VS version reads its input from SYSIN, produces its output on SYSPUNCH, and writes the source listing, error messages and so on, on SYSPRINT. (SYSLST and SYSPRINT do not contain the source listing or error messages for the assembler-language translator).

The VSE translators for COBOL and PL/I accept also the commands that can be used to access DL/I data bases. These commands, of the form EXEC DLI, are translated in a similar way to EXEC CICS commands, and are described in "Chapter 2.4. DL/I Services (EXEC DLI Command)" on page 77.

If the Entry Level System (ELS) is used (VSE only), a translator is generated with function limited to that supported by the host entry level CICS/VS system. This translator will flag functions that are not supported by the entry level system (as described in the CICS/VS Entry Level System User's Guide (DOS/VS)).

TRANSLATOR DATA SETS

INPUT DATA SET

The input data set must be a sequential data set. It may be on punched cards, on a direct-access device, or on magnetic tape.

For VSE, the input data set must contain 80-byte fixed-length unblocked records.

For OS/VS, the input data set for COBOL must contain fixed-length records (blocked or unblocked); for assembler language and PL/I it may contain either fixed-length or variable-length records. The maximum record size (LRECL) must not exceed 104 bytes.

OUTPUT DATA SET

The output data set must be a sequential data set. It may be on punched cards, on a direct access device, or on magnetic tape.

For VSE, the output data set must contain 80-byte fixed-length unblocked records.

For OS/VS, the output data set must contain 80-byte fixed-length records (blocked or unblocked).

LISTING DATA SET

The listing data set must be a sequential data set. Although the listing is usually printed, it can be stored on any magnetic tape or direct access device.

For VSE, the listing data set must contain 121-byte fixed-length unblocked records.

For OS/VS COBOL users, the listing data set must contain 121-byte fixed-length blocked records (RECFM=FBA).

For OS/VS assembler language and PL/I users, the listing data set must contain variable length blocked records with a maximum length of 121 bytes (RECFM=VBA).

TRANSLATED CODE

ASSEMBLER LANGUAGE

For an assembler-language application program, each command is replaced by an invocation of the DFHEICAL macro which 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 example in Figure 1 shows a simple assembler-language application program that uses the BMS command SEND MAP to send a map to a terminal.

The dynamic storage that is obtained for building the parameter list may be extended by the user to provide reentrant storage for assembler-language variables. The example in Figure 2 on page 11 shows a simple assembler-language application program that uses variables in dynamic storage.

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 a command also obeys system standards. Details are given below.

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

R1 contains address of parameter list.
R15 contains address of entry point.
R14 contains address of return point.
R13 contains address of save area.

Other registers are undefined.

The parameter list held in register 1 consists of two entries, as follows:

- Address of the EXEC interface block (EIB).
- Address of the COMMAREA. If there is no COMMAREA, the entry should contain the value X'80000000'.

A copy book, DFHEIBLK, containing a DSECT which describes the EIB is included automatically.

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

R15 contains entry point of EXEC interface program.
R14 contains return address in application program.
R0 is undefined.
R1 contains address of parameter list.

The entry point held in register 15 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 two macros DFHEISTG and DFHEIEND. These macros define the storage required for the parameter list and a save area.

```
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
```

Figure 1. Translated Code for BMS Command

```
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
```

Figure 2. Translated Code for Variables

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 command-level interface, and space for user variables.

Assembler-language programs larger than 4095 bytes that do not use the CODEREG parameter of the DFHEIENT macro to establish multiple base registers, must include an LTORG statement for use by DFHEIENT.

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

```
CODEREG - base register or registers
DATAREG - dynamic storage register or
          registers
EIBREG - register to address the EIB.
```

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=(13,5),EIBREG=6
```

The symbolic register DFHEIPLR is equated to the first DATAREG either explicitly specified or obtained by default. It is recommended that register 13 be used as the first dynamic storage register since register 13 points to the save area defined in dynamic storage by DFHEISTG. 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 to ensure that it is restored before each command.

An assembler-language application program that uses both the command-level interface and the macro-level interface (that is, a mixture of commands and macros) must define the macro global bit &DFHEIMX and set it to 1. This will ensure that register 13 points to the CSA, and register 12 to the TCA. In this case, DFHEIPLR will not be assumed by the expansion of a command.

COBOL

For COBOL, each command is replaced by one or more COBOL MOVE statements followed by a COBOL CALL statement. The purpose of the MOVE statements is to assign constants to COBOL data variables; this enables constants and names to be specified as arguments to options in the commands. For example, a command such as:

```
EXEC CICS RECEIVE MAP('A') END-EXEC
```

may be translated to:

```
MOVE '      ' TO DFHEIV0
MOVE 'A' TO DFHEIV1
CALL 'DFHEI1' USING DFHEIV0 DFHEIV1 AI
```

Declarations for the generated variables DFHEIV0 and DFHEIV1 are included automatically in working storage; their names are reserved. The string moved to DFHEIV0 is a hexadecimal string, not blanks. The use of EXEC, CICS, DLI, and END-EXEC as names for user variables should be avoided.

The translator modifies the linkage section by inserting the EIB structure as the first parameter, and inserts declarations of the temporary variables that it requires into the working-storage section.

It is possible to translate program segments for later inclusion into the procedure division.

PL/I

For PL/I, each command is always replaced by a single PL/I CALL statement. Warning messages from the PL/I compiler to the effect that the number of arguments to the call is incorrect should be ignored.

If OPTIONS(MAIN) is specified, the translator modifies the parameter list by inserting the EIB structure pointer as the first parameter, and a %INCLUDE statement to copy the structure into the program. If OPTIONS(MAIN) is not specified (that is, if the program is to be link-edited to the main module), the parameter list is not modified, and it is the application programmer's responsibility to pass the EIB structure (or addressability to it) to the link-edited program if access to it is required.

It is possible to translate program segments for later inclusion into a main program.

TRANSLATOR OPTIONS

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. The translator options and their defaults (indicated by underlines) are listed below. There are different sets of options for assembler language, COBOL, and for PL/I users.

Translator options are specified in the *ASM statement for assembler language, the CBL statement for COBOL, or in the *PROCESS statement for PL/I. These statements must precede the source program; there is no batching facility. The *ASM statement must obey the same syntax and continuation rules as the assembler-language comment statement. For OS/VS, options may also be specified in the EXEC job control statement that invokes the translator; if both methods are used, the options specified in the *ASM, CBL, or *PROCESS statements override those in the EXEC job control statement, and the last setting for each option takes precedence.

Translator options are written as a list within the CICS keyword option, for example:

```
*ASM CICS(NOPROLOG NOEPILOG)

        or

CBL CICS(QUOTE SPACE2)

        or

*PROCESS CICS(FLAG(W) SOURCE);
```

No characters, other than blanks, can appear before the CBL statement on the COBOL options card.

The options may appear in any order. They may be separated by one or more blanks or by a comma. If coded in the EXEC job control statement, the CICS keyword (and its associated parentheses) is unnecessary; only options for the translator are permitted.

For COBOL and PL/I under VSE, the CBL and *PROCESS statements can use the XOPTS keyword as an alternative to the CICS keyword, for example:

```
CBL XOPTS(QUOTE SPACE2)

        or

*PROCESS XOPTS(FLAG(W) SOURCE);
```

If the application program contains EXEC DLI commands, the options DLI and CICS must be specified in a CBL or *PROCESS statement, as follows:

```
CBL XOPTS(DLI,CICS)

        or

*PROCESS XOPTS(DLI,CICS);
```

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

```
*PROCESS CICS(SOURCE),ATTRIBUTES;
```

will be passed to the PL/I compiler preceded by:

```
*PROCESS ATTRIBUTES;
```

ASSEMBLER-LANGUAGE TRANSLATOR OPTIONS

NOSPIE
: prevents the translator trapping unrecoverable errors; instead, a dump is produced.

NOPROLOG
: prevents the translator inserting the macros DFHEISTG, DFHEIEND, and DFHEIENT, described earlier in this chapter.

NOEPILOG
: prevents the translator inserting the macro DFHEIRET, described earlier in this chapter.

COBOL TRANSLATOR OPTIONS

DEBUG|NODEBUG
: specifies whether or not the translator is to produce code that passes the translator line number through to CICS/VS to be displayed by the Execution (Command Level) Diagnostic Facility (EDF).

CICS
: specifies that the translator is to process EXEC CICS commands. This option may be specified either as an alternative to, or as a suboption of, the XOPTS option. If neither XOPTS nor CICS is specified, CICS is assumed by default. This option must not be specified for batch DL/I application programs containing EXEC DLI commands; XOPTS(DLI) must be specified instead.

DLI
: specifies that the translator is to process EXEC DLI commands.

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.

LANGLVL(1)|LANGLVL(2)
: specifies whether the translator is to analyse the source program and generate code according to the ANS X3.23-1968 (LANGLVL(1)) or ANS X3.23-1974 (LANGLVL(2)) interpretation. The same value for this option must be specified for the translator and following compiler.

LIST|NOLIST (VSE only)
: specifies whether or not the translator is to produce a listing of the source program.

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.

OPT|NOOPT
: specifies whether or not the translator is to generate SERVICE RELOAD statements to address the EIB and DFHCOMMAREA. The same value for this option must be specified for the translator and following compiler. The default is OPT for OS, NOOPT for VSE.

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 same value must be specified for the translator and following compiler.

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.

SOURCE|NOSOURCE (OS/VS only)
specifies whether or not the translator is to produce a listing of the source program.

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.

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

PL/I TRANSLATOR OPTIONS

DEBUG|NODEBUG
specifies whether or not the translator is to produce code that passes the translator line number through to CICS/VS to be displayed by the Execution (Command Level) Diagnostic Facility (EDF).

CICS
specifies that the translator is to process EXEC CICS commands. This option may be specified either as an alternative to, or as a suboption of, the XOPTS option. If neither XOPTS nor CICS is specified, CICS is assumed by default. This option must not be specified for batch DL/I application programs containing EXEC DLI commands; XOPTS(DLI) must be specified instead.

DLI
specifies that the translator is to process EXEC DLI commands.

FE
specifies that the translator is to produce 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)]
Abbreviation: F
specifies the minimum severity of error that requires a message to be listed.

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

LINECOUNT(n)
Abbreviation: LC
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. The default is 55.

MARGINS(m,n[,c])
Abbreviation: MAR
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"
Column number of left-hand margin.

"n"
Column number of right-hand margin. It must be greater than "m".

"c"
Column number of the ANS printer control character. It must be outside the values specified for "m" and "n". A zero value for "c" means no printer control character. Only the following printer control characters can be used:

(blank)
Skip one line before printing.

0
Skip two lines before printing.

-
Skip three lines before printing.

+
No skip before printing.

1
Start new page.

The default is MARGINS(2,72,0) for fixed-length records; and MARGINS(10,100,0) for variable-length records (OS/VS only).

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

OPMARGINS(m,n[,c])
Abbreviation: OM
specifies the translator output margins, that is, the margins of the input to the following compiler. Normally these will be the same as the input margins. For the meaning of "m", "n", and "c" see MARGINS. The default is OPMARGINS (2,72,0)

OPSEQUENCE(m,n)|NOOPSEQUENCE
Abbreviations: OS and NOS
specifies the position of the sequence field in the output records. For the meaning of "m" and "n" see SEQUENCE. The default is OPSEQUENCE(73,80).

OPTIONS|NOOPTIONS
Abbreviations: OP and NOP
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
Abbreviations: SEQ and NSEQ
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 no sequence field 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"
Column number of left-hand margin.

"n"
Column number of right-hand margin.

The extent must not exceed eight characters and must not overlap the source program (as specified in the MARGINS option).

The default for fixed-length records is SEQUENCE(73,80); for varying-length records it is SEQUENCE(1,8) (OS/VS only).

SOURCE|NOSOURCE
Abbreviations: S and NS
specifies whether or not the translator is to produce a listing of the source program.

XREF|NOXREF
Abbreviations: X and NX
specifies whether the translator is to include in the listing a list of all the commands used in the program together with the sequence numbers of the lines in which they are used.

Chapter 1.4. Programming Techniques and Restrictions

This chapter contains information that will help to improve performance and efficiency of an application program in the CICS/VS system.

The first section deals with general programming techniques; this section gives advice about the virtual-storage environment in which CICS/VS application programs operate. The rest of the chapter contains information that is applicable only to programs written in assembler language, COBOL, and PL/I respectively, and includes the restrictions that apply to each language when CICS/VS commands are used.

This manual does not contain any guidance on the use of programming-language statements or programming techniques that are unrelated to CICS/VS; such information is given in the appropriate language publications.

Files and queues are not defined within application programs; these definitions are established with the help of the system programmer. Refer to the CICS/VS System Programmer's Reference Manual.

GENERAL PROGRAMMING TECHNIQUES

To see how programming techniques can affect the performance and efficiency of the CICS/VS system, it is necessary to understand a little of the virtual-storage environment in which CICS/VS operates. Two concepts are important: multithreading and virtual-storage paging.

Multithreading is a technique, used by CICS/VS, that allows a single copy of an application program to process several transactions concurrently. For example, the first section of an application program may be processing one transaction. When that section is completed (in general, signaled by the execution of a CICS/VS command that causes a wait), processing of another transaction using a different section of the application program may take place. (Compare this with single threading, which is the execution of a program to completion. Processing of one transaction is completed before another transaction is started.)

Multithreading requires that all CICS/VS application programs be quasi-reentrant; that is, they must be serially reusable between entry and exit points, and any instructions or data altered in them must be restored. CICS/VS application programs using the command-level interface obey this rule automatically (provided that, in PL/I programs, static storage is used for read-only data). For these program to stay reentrant, variable data should not appear as static storage in PL/I, nor as a DC in the program CSECT in assembler language.

Care must be taken if a program involves lengthy calculations; since an application program retains control from one CICS/VS command to the next, processing of other transactions is completely excluded. However, the SUSPEND command can be used to allow other transaction processing to proceed; refer to "Chapter 4.3. Task Control" on page 187 for details.

Virtual-storage paging is a technique used by CICS/VS in a virtual-storage environment. The key objective of programming in this environment is the reduction of page faults. A page fault occurs when a program refers to instructions or data that do not reside in real storage, in which case, the page in virtual storage that contains the referenced instructions or data must be paged into real storage. The more paging required, the lower the overall system performance.

An understanding of the following terms is necessary for writing application programs to be run in a virtual-storage environment:

- **locality of reference** - the consistent reference, during the execution of the application program, to instructions and data within a relatively small number of pages (compared to the total number of pages in a program) for relatively long periods
- **working set** - the number and combination of pages of a program needed during a given period
- **validity of reference** - direct reference to the required pages, without intermediate storage references that retrieve useless data

In general, the following techniques should be used:

1. To improve locality of reference, processing should be sequential for both code and data, where possible.
   a. The ideal application program executes sequentially with no branch logic reference beyond a

small range of address space. However, error-handling or unusual-situation routines should be separated from the main section of a program; they should be subprograms.

b. Subroutines should be placed near to the caller.

c. Subprograms that are short and used only once or twice (other than error-handling or unusual-situation routines) should be coded inline in the calling program.

d. Try to keep the execution path in a straight line by using XCTL commands to transfer control to other programs when necessary, rather than LINK commands.

e. Initialize data as close as possible to its first use.

f. Define arrays or other data structures in the order in which they will be referred to. Refer to elements within arrays in the order in which they are stored; for example, in PL/I programs, in rows rather than in columns.

g. Issue as few as possible GETMAIN commands.

h. In COBOL programs, avoid using EXAMINE or VARIABLE MOVE operations, because these expand into subroutine executions.

2. To minimize the size of the working set, the amount of storage that a program refers to in a given period should be as small as possible.

   a. Write modular programs and structure the modules according to frequency and anticipated time of reference. Do not modularize merely for the sake of size; consider duplicate code inline as opposed to subroutines or separate modules.

   b. Use separate subprograms whenever the flow of the program suggests that execution will not be sequential.

   c. Do not tie up main storage awaiting a reply from a terminal user.

   d. Use command-level file control locate-mode input/output rather than move-mode.

   e. In COBOL programs, specify constants directly, rather than as data variables in the Working-Storage Section.

   f. In PL/I programs, use static storage for constant data.

   g. Avoid using LINK commands where possible, because they generate requests for main storage.

3. To improve validity of reference, the correct page should be determined directly.

   a. Avoid long searches for data.

   b. Use data structures that can be addressed directly, such as arrays, rather than structures that must be searched, such as chains.

   c. Avoid indirect addressing and any methods that simulate indirect addressing.

No attempt should be made to use overlays (paging techniques) in an application program. System paging is provided automatically and has superior performance. The design of an application program for a virtual-storage environment is similar to that for a real environment. The system should have all modules resident so that code on unreferenced pages need not be paged in.

If the program is dynamic, the entire program must be loaded across adjacent pages before execution begins. Dynamic programs can be purged from storage if not in use and an unsatisfied storage request exists. Allowing sufficient dynamic area to prevent purging is more expensive than making them resident, because a dynamic program will not share unused space on a page with another program.

CICS/VS MACROS USED WITH CICS/VS COMMANDS

Care should be exercised when writing application programs that contain a mixture of CICS/VS commands and CICS/VS macros, or in a macro-level program that invokes a command-level program and vice-versa.

When a RECEIVE MAP command is used with the SET option, the EXEC interface program always reuses the terminal input/output area (TIOA) obtained. Do not use a DFHSC TYPE=FREEMAIN, RELEASE=ALL macro in the same or an invoked program because the TIOA is freed unknown to the EXEC interface program, which will attempt to reuse it, giving unpredictable results.

OBJECT PROGRAM SIZE

The object module resulting from any application program must not occupy more than 262,136 bytes of main storage.

ASSEMBLER-LANGUAGE CONSIDERATIONS

RESTRICTIONS

The following restrictions apply to an assembler-language program that is to be used as a CICS/VS application program.

1. The assembler instructions COM (identify blank common control section), ICTL (input format control), and OPSYN (equate operation code) cannot be used.
2. Private code containing commands cannot be used.

COMMANDS CONTAINED WITHIN MACROS AND COPY CODE

Macro instructions that generate commands, and COPY code that contains commands, must be translated and stored in the source library in translated form for later inclusion by the assembler.

INVOKING ASSEMBLER-LANGUAGE APPLICATION PROGRAMS BY A CALL STATEMENT

Assembler-language application programs containing commands can be treated as separate CICS/VS programs that have their own PPT entries and that can be invoked by assembler-language, COBOL, PL/I, or RPG II application programs using LINK or XCTL commands (see "Chapter 4.4. Program Control" on page 189).

However, since assembler-language application programs containing commands are invoked by a system standard call, they can be invoked also by a COBOL, PL/I, or RPG II CALL statement or by an assembler-language CALL macro. A single CICS/VS application 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 commands can be linked with other assembler-language programs, or with programs in one of the high-level languages COBOL, PL/I, or RPG II, but with only one. When such an application program is linked with an assembler-language application program, the main program must be the one coded in the high-level language, and the PPT must specify that high-level language.

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

COBOL CONSIDERATIONS

RESTRICTIONS

The following restrictions apply to a COBOL program that is to be used as a CICS/VS application program. (Refer to the appropriate COBOL programmer's guide for more information about these features.)

1. Environment Division and Data Division entries normally associated with data management cannot be used.
2. File Section of the Data Division cannot be used.
3. Special features: ACCEPT, DISPLAY, EXHIBIT, INSPECT, REPORT WRITER, SEGMENTATION, SORT, TRACE, and UNSTRING cannot be used. For CICS/OS/VS, any feature that requires an OS/VS GETMAIN cannot be used, (for example, CURRENT-DATE).
4. Options that require the use of operating system services: COUNT, FLOW, STATE, STOP RUN, STXIT, or SYMDMP for CICS/DOS/VS; COUNT, ENDJOB, FLOW, DYNAM, STATE, STOP RUN, SYMDUMP, SYST, or TEST for CICS/OS/VS cannot be used. Note that since STOP RUN can be generated by the COBOL compiler, the application programmer must always code either a COBOL GOBACK statement or an EXEC CICS RETURN command at the end of the program.
5. COBOL statements: READ, WRITE, OPEN, and CLOSE cannot be used. (Commands are provided for the storage and retrieval of data, and for communication with terminals.)
6. Optimization option of the DOS Full COBOL V3 compiler cannot be used.
7. When separate COBOL routines are link-edited together, only the first can invoke CICS/VS.
8. The length of working storage plus the length of the TGT (task global table) must not exceed 64K bytes.

COMPILERS SUPPORTED

Only the following compilers are supported by CICS/VS:

- DOS Full COBOL Version 3 Compiler (5736-CB2)
- DOS/VS COBOL Compiler (5746-CB1)
- OS Full COBOL Version 4 Compiler (5734-CB2)
- OS/VS COBOL Compiler (5740-CB1)

BASE LOCATOR FOR LINKAGE (BLL)

The base locator for linkage (BLL) mechanism is used to address storage outside the working-storage section of an application program. It operates by addressing the storage as if it were a parameter to the program. The storage must be defined by means of an 01-level data definition in the linkage section of the program. The COBOL compiler generates code to address the storage via the parameter list. When the program is invoked, CICS/VS sets up the parameter list in such a way that the parameter list is itself addressable by the application program.

The parameter list must be defined as the first parameter to the program, unless a communication area is being passed to the program, in which case the DFHCOMMAREA definition must precede it. (See "Chapter 4.4. Program Control" on page 189).

In the following example, the first 02-level data name (that is, FILLER) is set up by CICS/VS to provide addressability to the other fields in the parameter list. The other data names are known as BLL cells, and address the remaining parameters of the program. There is a one-to-one correspondence between the 02-level data names of the parameter list definition and the 01-level data definitions in the linkage section.

```
LINKAGE SECTION.
01  PARMLIST.
    02  FILLER PIC S9(8) COMP.
    02  A-POINTER PIC S9(8) COMP.
    02  B-POINTER PIC S9(8) COMP.
    02  C-POINTER PIC S9(8) COMP.
01  A-DATA.
    02  PARTNO PIC  9(4).
    02  QUANTITY PIC 9(4) .
    02  DESCRIPTION PIC X(100).
01  B-DATA PIC X.
01  C-DATA PIC X.
```

In this example, A-POINTER addresses A-DATA, B-POINTER addresses B-DATA, and C-POINTER addresses C-DATA. The actual data names chosen for the BLL cells and for the data areas that they address are not significant, but the names must be defined in the correct order, so that the necessary correspondence is established.

If a BLL cell is named in the SET option of a CICS/VS command, subsequent reference to the corresponding data definition name will address the storage supplied by CICS/VS as a result of executing the command. For example, suppose that a program is required to read a variable-length record from a file, examine part of it, and update it; all of this is to be done without providing storage for the record within the program. Using the data definitions shown in the example above, the program could be written as follows:

```
EXEC CICS READ UPDATE DATASET('FILEA')
     RIDFLD(PART-REQD) SET(A-POINTER)
     LENGTH(A-LRECL) END-EXEC
IF A-LRECL LESS THAN 8 GO TO ERRORS.
IF QUANTITY GREATER ZERO
     SUBTRACT 1 FROM QUANTITY
     EXEC CICS REWRITE DATASET('FILEA')
          FROM(A-DATA) LENGTH(A-LRECL)
          END-EXEC.
```

CICS/VS reads the record into an internal buffer and supplies the address of the record in the buffer to the application program. The application program updates the record in the buffer and rewrites the record to the data set.

BLL and Chained Storage Areas

If access is needed to a series of chained storage areas (that is, areas each of which contain a pointer to the next area in the chain), a paragraph name must be inserted immediately following any statement that establishes addressability to one of the storage areas. For example:

```
LINKAGE SECTION.
01  PARMLIST.
    .
    .
    02  USERPTR PIC S9(8) COMP.
    .
    .
01  USERAREA.
    02  FIELD PIC X(4).
    02  NEXTAREA PIC S9(8) COMP.
    .
    .
PROCEDURE DIVISION.
    .
    .
    MOVE NEXTAREA TO USERPTR.
ANYNAME.
    MOVE FIELD TO TESTVAL.
    .
    .
```

In this example, storage areas mapped or defined by USERAREA are chained. The first MOVE statement establishes addressability to the next area in the chain. The second MOVE statement moves data from the newly addressed area, but only because a paragraph name follows the first MOVE statement. If no paragraph name is inserted, the reference to FIELD is taken as being to the storage area

that is addressed when the first MOVE statement refers to NEXTAREA. Insertion of a paragraph name causes the compiler to generate code to reestablish addressability through USERPTR, so that the reference to FIELD (and the next reference to NEXTAREA) is to the newly addressed storage area.

BLL and OCCURS DEPENDING ON Clauses

If the object of an OCCURS DEPENDING ON clause is defined in the linkage section, a special technique is required to ensure that the correct value is used at all times. In the following example, FIELD-COUNTER is defined in the linkage section. The MOVE FIELD-COUNTER TO FIELD-COUNTER statement is needed to ensure that unpredictable results do not occur when referring to DATA.

```
LINKAGE SECTION.
     .
     .
01   FILE-REC.
     .
     .
     02   FIELD-COUNTER PIC 9(4) COMP.
     02   FIELDS PIC X(5) OCCURS 1 TO 5
          TIMES DEPENDING ON
          FIELD-COUNTER.
     02   DATA PIC X(20).
     .
     .
PROCEDURE DIVISION.
     .
     .
     EXEC CICS READ DATASET('FILEA')
     RIDFLD(KEYVAL)
     SET(RECPTR) END-EXEC.
     MOVE FIELD-COUNTER TO
     FIELD-COUNTER.
     MOVE DATA TO DATA-VAL.
     .
     .
```

The MOVE statement referring to FIELD-COUNTER causes the compiler to reestablish the value it uses to compute the current number of occurrences of FIELDS and ensures that it can determine the displacement of DATA correctly.

BLL and Large Storage Areas

If an area greater than 4096 bytes is defined in the linkage section, additional statements are required to establish addressability to the extra area. An additional BLL cell is required for each extra 4096 bytes (or part) added to the area. (No additional corresponding 01-level data name definition is added, so the usual one-to-one correspondence of BLL cells to the data areas they address is not maintained.) An ADD statement is required also for each extra 4096 bytes (or part); it is placed after the statement that establishes addressability to the data area.

The extra statements are shown in the following example:

```
LINKAGE SECTION.
01   PARMLIST.
     .
     .
     02   FRPTR PIC S9(8) COMP.
     02   FRPTR1 PIC S9(8) COMP.
     .
     .
01   FILE-REC.
     02   FIELD1 PIC X(4000).
     02   FIELD2 PIC X(1000).
     02   FIELD3 PIC X(400).
     .
     .
PROCEDURE DIVISION.
     .
     .
     EXEC CICS READ DATASET('FILEA')
     RIDFLD(KEYVAL) SET(FRPTR)
     END-EXEC.
     ADD 4096 TO FRPTR GIVING FRPTR1.
```

BLL and the Optimization Feature

If an application program is to be compiled using the OS full COBOL V4 Compiler, the OS/VS COBOL compiler, or the DOS/VS COBOL compiler with the optimization (OPT) feature, a special compiler control statement must be inserted at appropriate places within the program to ensure addressability to a particular area defined in the linkage section. This control statement has the form:

```
SERVICE RELOAD fieldname
```

where "fieldname" is the symbolic name of a specific storage area which is also defined in an 01-level statement in the linkage section. The SERVICE RELOAD statement must be used following each statement which modifies addressability to an area defined in the linkage section, that is, whenever the contents of a BLL cell is changed in any way.

If a HANDLE CONDITION or a HANDLE AID command is invoked as a result of a command that changes the contents of a BLL cell, a SERVICE RELOAD statement should follow the label branched to as the exit for that condition.

If the BLL mechanism is used (described earlier in this chapter), addressability to the parameter list must be established at the start of the procedure division. This is done by adding a SERVICE RELOAD PARMLIST statement at the start of the procedure division in the earlier examples.

For example, after a locate-mode input operation the SERVICE RELOAD statement

must be used to establish addressability to the data, as follows:

```
EXEC CICS HANDLE CONDITION
          ERROR(GIVEUP)
          LENGERR(BADLENGTH) END-EXEC
EXEC CICS READ DATASET('FILEA')
          RIDFLD(PART-REQD)
          SET(A-POINTER)
          LENGTH(A-LRECL)
          END-EXEC
SERVICE RELOAD A-DATA.
BADLENG.
SERVICE RELOAD A-DATA.
```

If an address is moved into a BLL cell, addressability must be established in the same way, for example:

```
MOVE B-POINTER TO A-POINTER
SERVICE RELOAD A-DATA.
```

If areas larger than 4096 bytes are being addressed, the secondary BLL cells must be reset after the SERVICE RELOAD statement has been executed. (Resetting a BLL cell is described in the previous section.)

BLL and Large Communication Area

If a communication area greater than 4096 bytes is defined in the linkage section, an additional statement is required for each extra 4096 bytes (or part) to establish addressability to the extra area. For example, the coding for a communication area of 10000 bytes might be as follows:

```
LINKAGE SECTION
01 DFHCOMMAREA PIC X(10000).
01 PARMLIST.
   02 FILLER PIC S9(8) COMP.
   02 FILLER PIC S9(8) COMP.
   02 FILLER PIC S9(8) COMP.
   02 ...
```

The first FILLER statement establishes BLL addressability to the first 4096 bytes, the second FILLER statement establishes addressability to the next 4096 bytes, and so on.

NOTRUNC COMPILER OPTION

If an argument to a command is greater than 9999 in value, the NOTRUNC compiler option must be specified to ensure successful execution.

PROGRAM SEGMENTS

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

PL/I CONSIDERATIONS

RESTRICTIONS

The following restrictions apply to a PL/I program that is to be used as a CICS/VS application program. (Refer to the PL/I Optimizing Compiler Programmer's Guide for more information about these features.)

1. The multitasking built-in functions: COMPLETION, PRIORITY, and STATUS cannot be used.

2. The multitasking options: EVENT, PRIORITY, and TASK cannot be used.

3. The PL/I statements: READ, WRITE, GET, PUT, OPEN, CLOSE, DISPLAY, DELAY, REWRITE, LOCATE, DELETE, UNLOCK, STOP, HALT, EXIT, FETCH, and RELEASE should not be used. Commands are provided for the storage and retrieval of data, and for communication with terminals.

   Refer to the PL/I Optimizing Compiler Programmer's Guide for information on when the use of these PL/I statements is necessary and the consequences of using them.

4. PL/I Sort/Merge cannot be used.

5. Static storage (except for read-only data) cannot be used. A consequence of this restriction for CICS/DOS/VS PL/I users is that CONTROLLED variables cannot be used.

6. A declaration for a variable with the attributes STATIC EXTERNAL should have also the INITIAL attribute. Failing this, such declarations will generate a common CSECT that cannot be handled by CICS/VS.

PL/I STAE EXECUTION-TIME OPTION

If this option is specified, an abend occurring in the transaction will be handled by PL/I error handling routines, and the transaction may terminate normally, in which case, CICS/VS facilities, such as dynamic transaction backout (DTB), will not be invoked.

COMPILERS SUPPORTED

Only the following compilers are supported:

- DOS PL/I Optimizing Compiler, Version 1, Release 5.0
- OS PL/I Optimizing Compiler, Version 1, Release 3.0

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, it cannot be the first program 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 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.

Chapter 1.5. Exceptional Conditions

Exceptional conditions may occur during the execution of a CICS/VS command and, unless specified otherwise in the application program by an IGNORE CONDITION or HANDLE CONDITION command or by the NOHANDLE option, a default action for each condition will be taken by CICS/VS. Usually, this default action is to terminate the task abnormally. (Exceptional conditions are described, together with the CICS/VS default action, at the end of a chapter, and a list of conditions that apply to a command is included within the syntax box for the command.)

However, to prevent abnormal termination, an exceptional condition can be dealt with in the application program by 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 must be executed before the command which may give rise to the associated condition.

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 from it are reactivated, and those in the lower-level program are deactivated. (Refer to "Chapter 4.4. Program Control" on page 189 for information about logical levels.)

Some exceptional conditions can occur during the execution of any one of a number of unrelated commands. For example, IOERR can occur during file-control operations, interval-control operations, and others. If the same action is required for all occurrences, a single HANDLE CONDITION IOERR 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 fields EIBFN and EIBRCODE (in the EIB) can be tested to find out which exceptional condition has occurred and in which command. The EIB is described in "Appendix A. EXEC Interface Block" on page 239.

The IGNORE CONDITION command specifies that no action is to be taken if an exceptional condition occurs. Execution of a command could result in several conditions being raised. CICS/VS checks these in a predetermined order and only the first one that is not ignored (by an IGNORE CONDITION command) will be passed to the application program.

The NOHANDLE option may be used with any command to specify that no action is to be taken for any condition resulting from the execution of that command. In this way the scope of the IGNORE CONDITION command covers specified conditions for all commands (until a HANDLE CONDITION for the condition is executed) and the scope of the NOHANDLE option covers all conditions for specified commands.

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.

Commands should not be included in an error routine that may give rise to the same condition that caused the branch to the routine; special care should be taken not to cause a loop on the ERROR condition. A loop can be avoided by including a HANDLE CONDITION ERROR command as the first command in the error routine. The original error action should be reinstated at the end of the error routine by including a second HANDLE CONDITION ERROR command.

HANDLE EXCEPTIONAL CONDITIONS (HANDLE CONDITION)

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

This command is used to specify the label to which control is to be passed if an

exceptional condition occurs. It remains in effect until a subsequent IGNORE CONDITION command for the condition is encountered. No more than twelve conditions are allowed in the same command; additional conditions must be specified in further HANDLE CONDITION commands. 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. DUPREC is to be handled as a special case; system default action (that is, to terminate the task abnormally) is to be taken for LENGERR; and all other conditions are to be handled by the generalized error routine ERRHANDL.

```
EXEC CICS HANDLE CONDITION
          ERROR(ERRHANDL)
          DUPREC(DUPRTN)
          LENGERR
```

If the generalized error routine can handle all exceptional conditions except IOERR, for which the default action (that is, to terminate the task abnormally) is required, IOERR (without a label) would be added to the above command.

In an assembler-language application program, a branch to a label caused by an exceptional condition will restore the registers in the application program to their values at the point where the EXEC interface program is invoked.

In a PL/I application program, a branch to a label in an inactive procedure or in an inactive begin block, caused by an exceptional condition, will produce unpredictable results.

HANDLE CONDITION COMMAND OPTION

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 taken if the condition occurs.

IGNORE EXCEPTIONAL CONDITIONS (IGNORE CONDITION)

```
IGNORE CONDITION condition
                 [condition]...
```

This command is used to specify that no action is to be taken if an exceptional condition occurs. It remains in effect until a subsequent HANDLE CONDITION command for the condition is encountered. No more than twelve conditions are allowed in the same command; additional conditions must be specified in further IGNORE CONDITION commands. The option "condition" specifies the name of the exceptional condition that is to be ignored.

LIST OF EXCEPTIONAL CONDITIONS

The following list shows all the exceptional conditions that can occur during the execution of CICS/VS commands. Each condition is followed by one or more keywords and by numbers (in parentheses). The keywords are the commands during the execution of which the condition may occur, and the numbers are the chapters that describe those commands. For the meaning of a condition, and the default action associated with that condition, refer to the list of exceptional conditions at the end of the indicated chapter.

CBIDERR	ALLOCATE(3.2), CONVERSE(3.2), EXTRACT ATTACH(3.2), SEND(3.2)
DSIDERR	DELETE(2.2), READ(2.2), READNEXT(2.2), READPREV(2.2), REWRITE(2.2), STARTBR(2.2), UNLOCK(2.2), WRITE(2.2)
DSSTAT	ISSUE RECEIVE(3.4)
DUPKEY	READ(2.2), READNEXT(2.2), READPREV(2.2)
DUPREC	WRITE(2.2), REWRITE(2.2)
ENDDATA	RETRIEVE(4.2)
ENDFILE	READNEXT(2.2), READPREV(2.2)
ENDINPT	RECEIVE(3.2)
ENQBUSY	ENQ(4.3)
ENVDEFERR	RETRIEVE(4.2)
EOC	CONVERSE(3.2), RECEIVE MAP(3.3), RECEIVE(3.2)

EODS	CONVERSE(3.2), ISSUE RECEIVE(3.4), RECEIVE MAP(3.3), RECEIVE(3.2)
EOF	CONVERSE(3.2), RECEIVE(3.2)
ERROR	General exceptional condition (1.5). Not included in the list of conditions in the syntax of individual commands.
EXPIRED	DELAY(4.2), POST(4.2)
FUNCERR	ISSUE ABORT(3.4), ISSUE ADD(3.4), ISSUE END(3.4), ISSUE ERASE(3.4), ISSUE NOTE(3.4), ISSUE QUERY(3.4), ISSUE REPLACE(3.4), ISSUE SEND(3.4), ISSUE WAIT(3.4)
IGREQCD	CONVERSE(3.2), ISSUE SEND(3.4), SEND(3.2), SEND MAP(3.3), SEND PAGE(3.3), SEND TEXT(3.3)
IGREQID	SEND MAP(3.3), SEND PAGE(3.3), SEND TEXT(3.3)
ILLOGIC	DELETE(2.2), ENDBR(2.2), READ(2.2), READNEXT(2.2), READPREV(2.2), RESETBR(2.2), REWRITE(2.2), STARTBR(2.2), UNLOCK(2.2), WRITE(2.2)
INBFMH	CONVERSE(3.2), RECEIVE(3.2)
INVERRTERM	ROUTE(3.3)
INVLDC	ROUTE(3.3), SEND MAP(3.3), SEND TEXT(3.3)
INVMPSZ	RECEIVE MAP(3.3), SEND MAP(3.3)
INVREQ	ALLOCATE(3.2), ASSIGN(1.6), CANCEL(4.2), CONVERSE(3.2), DELAY(4.2), DELETE(2.2), ENDBR(2.2), EXTRACT ATTACH(3.2), EXTRACT TCT(3.2), FREE(3.2), POST(4.2), READ(2.2), READNEXT(2.2), READPREV(2.2), RECEIVE(3.2), RESETBR(2.2), RETRIEVE(4.2), RETURN(4.4), REWRITE(2.2), SEND(3.2), SEND MAP(3.3), SEND PAGE(3.3), SEND TEXT(3.3), START(4.2), STARTBR(2.2), WAIT JOURNAL(5.5), WRITE(2.2), WRITEQ TS(4.7)
INVTSREQ	RETRIEVE(4.2)
IOERR	DELETE(2.2), JOURNAL(5.5), READ(2.2), READNEXT(2.2), READPREV(2.2), READQ TD(4.6), READQ TS(4.7), RESETBR(2.2), RETRIEVE(4.2), REWRITE(2.2), START(4.2), STARTBR(2.2), UNLOCK(2.2), WAIT JOURNAL(5.5), WRITE(2.2), WRITEQ TD(4.6), WRITEQ TS(4.7)
ISCINVREQ	CANCEL(4.2), DELETE(2.2), DELETEQ TD(4.6), DELETEQ TS(4.7), ENDBR(2.2), READ(2.2), READNEXT(2.2), READPREV(2.2), READQ TD(4.6), READQ TS(4.7), RESETBR(2.2), RETRIEVE(4.2), REWRITE(2.2), START(4.2), STARTBR(2.2), UNLOCK(2.2), WRITE(2.2), WRITEQ TD(4.6), WRITEQ TS(4.7)
ITEMERR	READQ TS(4.7), WRITEQ TS(4.7)
JIDERR	JOURNAL(5.5), WAIT JOURNAL(5.5)
LENGERR	CONVERSE(3.2), ISSUE RECEIVE(3.4), JOURNAL(5.5), READ(2.2), READNEXT(2.2), READPREV(2.2), READQ TD(4.6), READQ TS(4.7), RECEIVE(3.2), RETRIEVE(4.2), REWRITE(2.2), WRITE(2.2), WRITEQ TD(4.6)
MAPFAIL	RECEIVE MAP(3.3)
NODATARECD	ISSUE RECEIVE(3.4)
NOJBUFSP	JOURNAL(5.5)
NONVAL	ISSUE LOAD(3.2)
NOPASSBKRD	RECEIVE(3.2)
NOPASSBKWR	SEND(3.2)
NOSPACE	REWRITE(2.2), WRITE(2.2), WRITEQ TD(4.6), WRITEQ TS(4.7)
NOSTART	ISSUE LOAD(3.2)
NOSTG	GETMAIN(4.5)
NOTALLOC	CONVERSE(3.2), EXTRACT ATTACH(3.2), FREE(3.2), ISSUE DISCONNECT(3.2), ISSUE SIGNAL(3.2),

	POINT(3.2), RECEIVE(3.2), SEND(3.2), WAIT TERMINAL(3.2)
NOTFND	CANCEL(4.2), DELETE(2.2), READ(2.2), READNEXT(2.2), READPREV(2.2), RESETBR(2.2), RETRIEVE(4.2), STARTBR(2.2)
NOTOPEN	DELETE(2.2), JOURNAL(5.5), READ(2.2), READNEXT(2.2), READPREV(2.2), READQ TD(4.6), RESETBR(2.2), REWRITE(2.2), STARTBR(2.2), UNLOCK(2.2), WAIT JOURNAL(5.5), WRITE(2.2), WRITEQ TD(4.6)
OVERFLOW	SEND MAP(3.3)
PGMIDERR	HANDLE ABEND(5.2), LINK(4.4), LOAD(4.4), RELEASE(4.4), XCTL(4.4)
QBUSY	READQ TD(4.6)
QIDERR	DELETEQ TD(4.6), DELETEQ TS(4.7), READQ TD(4.6), READQ TS(4.7), WRITEQ TD(4.6), WRITEQ TS(4.7)
QZERO	READQ TD(4.6)
RDATT	CONVERSE(3.2), RECEIVE MAP(3.3), RECEIVE(3.2)
RETPAGE	SEND MAP(3.3), SEND PAGE(3.3), SEND TEXT(3.3)
RTEFAIL	ROUTE(3.3)
RTESOME	ROUTE(3.3)
SEGIDERR	READ(2.2), READNEXT(2.2), READPREV(2.2)
SELNERR	ISSUE ABORT(3.4), ISSUE ADD(3.4), ISSUE END(3.4), ISSUE ERASE(3.4), ISSUE NOTE(3.4), ISSUE QUERY(3.4), ISSUE REPLACE(3.4),
	ISSUE SEND(3.4), ISSUE WAIT(3.4)
SESSBUSY	ALLOCATE(3.2)
SESSIONERR	ALLOCATE(3.2), CONVERSE(3.2), EXTRACT ATTACH(3.2), FREE(3.2), ISSUE DISCONNECT(3.2), ISSUE SIGNAL(3.2), POINT(3,2), RECEIVE(3,2), SEND(3.2), WAIT TERMINAL(3.2)
SIGNAL	CONVERSE(3.2), ISSUE DISCONNECT(3.2), RECEIVE(3.2), WAIT TERMINAL(3.2), SEND(3.2), WAIT SIGNAL(3.2)
SYSBUSY	ALLOCATE(3.2)
SYSIDERR	ALLOCATE(3.2), CANCEL(4.2), DELETE(2.2), DELETQ TD(4.6), DELETEQ TS(4.7), ENDBR(2.2), READ(2.2), READNEXT(2.2), READPREV(2.2), READ TD(4.6), READQ TS(4.7), RESETBR(2.2), RETRIEVE(4.2), REWRITE(2.2), START(4.2), STARTBR(2.2), UNLOCK(2.2), WRITE(2.2), WRITEQ TD(4.6), WRITEQ TS(4.7)
TERMIDERR	ISSUE COPY(3.2), START(4.2)
TRANSIDERR	START(4.2)
TSIOERR	PURGE MESSAGE(3.3), SEND MAP(3.3), SEND PAGE(3.3), SEND TEXT(3.3)
UNEXPIN	ISSUE ABORT(3.4), ISSUE ADD(3.4), ISSUE END(3.4), ISSUE ERASE(3.4), ISSUE NOTE(3.4), ISSUE QUERY(3.4), ISSUE RECEIVE(3.4), ISSUE REPLACE(3.4), ISSUE SEND(3.4), ISSUE WAIT(3.4)
WRBRK	CONVERSE(3.2), SEND MAP(3.3), SEND PAGE(3.3), SEND(3.2), SEND TEXT(3.3)

Chapter 1.6. 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 and ASSIGN commands are provided to make access to such information possible and these commands are described in the following sections. Not all fields are intended to be accessed by the application program; refer to the CICS/VS Application Programmer's Reference Manual (Macro Level) for a list of the fields that are part of the application programming interface (the API) and that will remain valid from release to release. Details of each control block and its fields are contained in the appropriate CICS/VS Data Areas publication.

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. The offsets, fieldnames, and lengths of the fields in this control block are as follows:

Offset (Hex)	Field Name	Length (Bytes)
0	EIBTIME	4
4	EIBDATE	4
8	EIBTRNID	4
C	EIBTASKN	4
10	EIBTRMID	4
14	EIBRSVD1	2
16	EIBCPOSN	2
18	EIBCALEN	2
1A	EIBAID	1
1B	EIBFN	2
1D	EIBRCODE	6
23	EIBDS	8
2B	EIBREQID	8
33	EIBRSRCE	8
3B	EIBSYNC	1
3C	EIBFREE	1
3D	EIBRECV	1
3E	EIBSEND	1
3F	EIBATT	1
40	EIBEOC	1
41	EIBFMH	1

An application program can access all of the fields in the EIB by name. The EIB contains information, additional to that provided by execution of a terminal control command, that is useful during the execution of an application program, such as the transaction identifier, the time and date (initially when the task is started, 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.

ACCESS TO CICS/VS STORAGE AREAS (ADDRESS)

```
ADDRESS [CSA(ptr-ref)]
        [CWA(ptr-ref)]
        [TCTUA(ptr-ref)]
        [TWA(ptr-ref)]
```

This command is used to obtain access to any of the following areas: the common storage area (CSA), the common work area (CWA), the terminal control table user area (TCTUA), and the transaction work area (TWA).

ADDRESS COMMAND OPTIONS

CSA
allows access to control blocks addressed by the CSA. The pointer reference is set to the address of the CSA. The CSA gives access to all fields in CICS/VS control blocks and storage areas.

CWA
is used to pass information between application programs. The pointer reference is set to the address of the CWA. If a CWA does not exist, the pointer reference is set to X'FF000000'.

TCTUA
is used also to pass information between application programs, but only if the same terminal is associated with the application programs involved (which can be in different tasks). The pointer reference is set to the address of the TCTUA. If a TCTUA does not exist, the pointer reference is set to X'FF000000'. The data area contains the address of the TCTUA of the principal facility, not that for any alternate facility that may have been allocated.

TWA
is used also to pass information between application programs but only if they are in the same task. The pointer reference is set to the

address of the TWA. If a TWA does not exist, the pointer reference is set to X'FF000000'.

An example of the use of the ADDRESS command is given in the next section. (Information can also be passed between programs using the COMMAREA option of the program control commands, described in "Chapter 4.4. Program Control" on page 189.)

If an ADDRESS command is included in a COBOL program that is to be compiled using the optimization feature, it must be followed by SERVICE RELOAD statements to reload the BLL cell being used. (The SERVICE RELOAD statement is described earlier in the manual in "BLL and the Optimization Feature" in "Chapter 1.4. Programming Techniques and Restrictions" on page 17.)

VALUES OUTSIDE THE APPLICATION PROGRAM (ASSIGN)

```
ASSIGN option(data-area)
       [option(data-area)]...

Condition:  INVREQ
```

This command is used to obtain values outside the local environment of the application program. The value obtained is assigned to the data area specified in the option.

The following values can be obtained:

- lengths of storage areas
- values needed when communicating with the 2980 General Banking Terminal System (copied from the TCTTE)
- values needed during BMS operations (copied from the TCA)
- values needed during batch data interchange
- screen size in use on the 3270
- other information that may be useful to the application programmer (copied from various CICS/VS control blocks)

A complete list of ASSIGN command options is given at the end of this chapter.

The following example shows, in the different application programming languages, how the ADDRESS command is used to obtain access to the TWA, and how the ASSIGN command is used to obtain the length of the TWA. Included is a test for validity based on the fact that, if there is no TWA, the ASSIGN command will obtain a length of zero.

Assembler Language

```
DSWORKA   DSECT
WAPTR     EQU   08
          USING DSWORKA,WAPTR
.
.
.
COUNT     DS    H
.
.
.
DFHEISTG  DSECT
TWALENG   DS    H
CODE      CSECT
          EXEC CICS ASSIGN
               TWALENG(TWALENG)
          CLC   TWALENG,=H'0'
          BNH   CONTINUE
          EXEC CICS ADDRESS TWA(WAPTR)
          LH    6,COUNT
          LA    6,1(6)
          STH   6,COUNT
CONTINUE  DS    0H
```

COBOL

```
WORKING-STORAGE SECTION.
77  TWALENG PIC S9(4) COMP.

LINKAGE SECTION.
01  BLLCELLS.
    02  FILLER PIC S9(8) COMP.
    02  WAPTR PIC S9(8) COMP.
01  WORKAREA.
    02  COUNT PIC S9(4) COMP.

PROCEDURE DIVISION.
    EXEC CICS ASSIGN TWALENG(TWALENG)
    END-EXEC
    IF TWALENG GREATER THAN 0 THEN
    EXEC CICS ADDRESS TWA(WAPTR)
    END-EXEC
    ADD 1 TO COUNT.
```

PL/I

```
DCL TWALENG FIXED BIN(15);
DCL 1 WORKAREA BASED(WAPTR),
      2 COUNT FIXED BIN(15);
      .
      .
EXEC CICS ASSIGN TWALENG(TWALENG);
IF TWALENG>0 THEN DO;
   EXEC CICS ADDRESS TWA(WAPTR);
   COUNT=COUNT+1;
END;
```

ASSIGN COMMAND OPTIONS

Where any of the following options apply to terminals or terminal-related data, the reference is always to the principal facility.

ABCODE
specifies a variable that is set to the current value of the abend code (abend codes are documented in CICS/VS Messages and Codes). If an abend has not occurred, the variable is set to blanks. The format of the value is a four-byte character string.

APPLID
specifies that the value required is the application name used in transaction routing or to identify the local CICS/VS system to VTAM. The format of the value is an eight-byte character string.

COLOR
specifies that the value required is an indicator showing that the terminal is defined as having the extended color capability (X'FF'); or no extended color capability (X'00'). If this option is specified and there is no TCTTE for the task, the INVREQ condition occurs. The format of the value is a one-byte character.

CWALENG
specifies that the length of the CWA is required. If no CWA exists, a zero length is returned. No exceptional condition occurs. The format of the value is halfword binary.

DELIMITER
specifies that the value required is the data-link control character for a 3600, copied from TCTTEDLM. If this option is specified and there is no TCTTE for the task, the INVREQ condition occurs. The format of the value is a one-byte character.

DESTCOUNT
specifies that the value required is the relative overflow control number of the destination that has encountered overflow. If this option is specified when overflow processing is not in effect, the value obtained will be meaningless. If no BMS commands have been issued, the INVREQ condition occurs. The format of the value is halfword binary.

DESTID
specifies that the value required is the identifier of the outboard destination, padded with blanks on the right to eight characters. If this option is specified before a batch data interchange command has been issued in the task, the INVREQ condition occurs. The format of the value is an eight-byte character string.

DESTIDLENG
specifies that the value required is the length of the destination identifier obtained by DESTID. If this option is specified before a batch data interchange command has been issued in the task, the INVREQ condition occurs. The format of the value is halfword binary.

EXTDS
specifies that the value required is an indicator showing that the terminal is defined as having the extended data stream capability (X'FF'); or no extended data stream capability (X'00'). If this option is specified and there is no TCTTE for the task, the INVREQ condition occurs. The format of the value is a one-byte character.

FACILITY
specifies that the value required is the identification of the facility that initiated the transaction. The value is copied from the first four bytes pointed at by TCAFCAAA. If this option is specified, and there is no allocated facility, the INVREQ condition occurs. For example, this option gives the name of the transient data destination whose trigger level caused the transaction to be started. The format of the value is a four-byte character string.

FCI
specifies that the value required is the facility control indicator, copied from TCAFCI, that indicates the type of facility associated with the transaction, for example, X'01' indicates a terminal or logical unit. The obtained value is always returned. No exceptional condition occurs. The format of the value is a one-byte character.

HILIGHT
specifies that the value required is an indicator showing that the terminal is defined as having the extended highlight capability (X'FF'); or no extended highlight capability (X'00'). If this option is specified and there is no TCTTE for the task, the INVREQ condition occurs. The format of the value is a one-byte character.

LDCMNEM
specifies that the value required is the LDC mnemonic of the destination that has encountered overflow. If this option is specified when overflow processing is not in effect, the value obtained will be meaningless. No exceptional condition occurs. The format of the value is a two-byte character string.

LDCNUM
specifies that the value required is the LDC numeric value of the destination that has encountered overflow. If this option is specified when overflow processing is not in effect, the value obtained will be meaningless. No exceptional condition occurs. The format of the value is a one-byte character.

NUMTAB
specifies that the value required is the number of the tabs required to position the print element in the correct passbook area of the 2980. If this option is specified and there is no TCTTE for the task, the INVREQ condition occurs. The format of the value is a one-byte character.

OPCLASS
specifies that the value required is the operator class, copied from TCTTEOCL. If this option is specified and there is no TCTTE for the task, the INVREQ condition occurs. The format of the value is a three-byte character string.

OPID
specifies that the value required is the operator identification, copied from TCTTEOI. If this option is specified and there is no TCTTE for the task, the INVREQ condition occurs. The format of the value is a three-byte character string.

OPSECURITY
specifies that the value required is the operator security key, copied from TCTTESK. If this option is specified and there is no TCTTE for the task, the INVREQ condition occurs. The format of the value is a three-byte character string.

PAGENUM
specifies that the value required is the current page number for the destination that has encountered an overflow. If this option is specified when overflow processing is not in effect, the value obtained will be meaningless. If no BMS commands have been issued, the INVREQ condition occurs. The format of the value is halfword binary.

PRINSYSID
specifies that the value required is the name of the TCTSE (terminal control table system entry) associated with the principal facility. If there is no TCTTE for the task or if the principal facility is not an LU6 or MRO session, the INVREQ condition occurs. The format of the value is a four-byte character string.

PS
specifies that the value required is an indicator showing that the terminal is defined as having the programmed symbols capability (X'FF'); or no programmed symbols capability (X'00'). If this option is specified and there is no TCTTE for the task, the INVREQ condition occurs. The format of the value is a one-byte character.

RESTART
specifies that the value required is an indicator showing whether a restart (X'FF'), as opposed to a normal start (X'00'), has occurred.

SCRNHT
specifies that the value required is the height of the current 3270 screen. If this option is specified and there is no TCTTE for the task, the INVREQ condition occurs. The format of the value is halfword binary.

SCRNWD
specifies that the value required is the width of the current 3270 screen. If this option is specified and there is no TCTTE for the task, the INVREQ condition occurs. The format of the value is halfword binary.

SIGDATA
specifies that the value required is the signal data received from a logical unit, copied from TCTESIDI. If this option is specified and there is no TCTTE for the task, the INVREQ condition occurs. The format of the value is a four-byte character string.

STARTCODE
specifies that the value required is a code indicating how a transaction has been started. The format of the value is a two-byte character string which can have the following values:

Code	Transaction started by
QD	Transient data trigger level
S	START command (no data)
SD	START command (with data)
TD	Terminal input
U	User-attached task

STATIONID
specifies that the value required is the station identifier of a 2980. If this option is specified and there is no TCTTE for the task, the INVREQ condition occurs. The format of the value is a one-byte character.

SYSID
specifies that the value required is the name given to the local CICS/VS system. This value may be specified

in the SYSID option of a file control, interval control, temporary storage, or transient data command, in which case the resource to be accessed is assumed to be on the local system. The format of the value is a four-byte character string.

TCTUALENG
specifies that the value required is the length of the terminal control table user area (TCTUA). If no TCTUA exists, a zero length is returned. No exceptional condition occurs. The format of the value is halfword binary.

TELLERID
specifies that the value required is the teller identifier of a 2980. If this option is specified and there is no TCTTE for the task, the INVREQ condition occurs. The format of the value is a one-byte character.

TERMCODE
specifies that the value required is a code giving the type and model number of the terminal associated with the task, copied from TCTTETT and TCTTETM. If this option is specified and there is no TCTTE for the task, the INVREQ condition occurs. The format of the value is a two-byte character string.

TWALENG
specifies that the value required is the length of the transaction work area (TWA). If no TWA exists, a zero length is returned. No exceptional condition occurs.

UNATTEND
specifies that the value required is a code indicating that the mode of operation of the terminal is unattended (X'FF') or attended (X'00'), copied from TCTEMOP. If this option is specified and there is no TCTTE for the task, the INVREQ condition occurs.

VALIDATION
specifies that the value required is an indicator showing that the terminal is defined as having the validation capability (X'FF') consisting of the mandatory fill, mandatory enter, and trigger attributes. No validation capability is indicated by (X'00'). If this option is specified and there is no TCTTE for the task, the INVREQ condition occurs. The format of the value is a one-byte character.

Chapter 1.7. Execution (Command-Level) Diagnostic Facility

The Execution (Command-Level) Diagnostic Facility (EDF) enables an application programmer to test a command-level application program online without making any modifications to the source program or the program preparation procedure. The facility intercepts execution of the program at various points and displays information about the program at these points. Also displayed are any screens sent by the user program, so that the programmer can converse with the application program during testing 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.

EDF is a command-level diagnostic aid only, and unpredictable results may occur if macro instructions are coded in application programs using this facility.

For OS/VS only, this facility is not supported if TCTUA=VICOMPAT is specified in the DFHSG TYPE=INITIAL system macro.

TCAM (a data stream access method) is not supported by EDF, which supports only terminals and logical units.

Terminal input received by EDF should be in read modified format as it is mapped using BMS.

When using EDF, the user task should specify DTIMOUT=NO or a large value in the DFHPCT TYPE=ENTRY system macro. When running EDF from the same terminal as the user task, the user task must not specify the ONEWTE parameter in the DFHPCT TYPE=OPTGRP system macro.

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 EXEC CICS and EXEC DLI 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, by identifying the cause of interception. 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 has been translated with the 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 source 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 the 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 suppress execution of the command (that is, convert it to a null operation).
- If the current command is being displayed after it has been executed, the user can modify certain argument values and the response code by overtyping the displayed value or response with the required 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 command displays are suppressed until one or more of a set of specific conditions is fulfilled. These conditions may be:

  A specific named command is encountered.

  Any 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.

Resource level security checks will be made during execution of the transaction under test unless EDF has been defined as not requiring these checks. If such checks indicate that the EDF user is not allowed access to the resource, the user transaction will be abended.

INSTALLING EDF

To ensure that EDF is available on the test system, the system programmer must make one group entry in the PPT and one group entry in the PCT (see the CICS/VS System Programmer's Reference Manual for details of constructing a PPT and PCT).

EDF can send messages greater than 4K bytes in length. VTAM users should ensure that their NCP (network control program) can handle data of this length. The same applies if temporary storage is defined as auxiliary, in which case the VSAM control interval length must be large enough to handle the message.

INVOKING EDF

EDF can be run on the same terminal as the transaction requiring checkout provided that the application under test does not make use of extended attributes, or on a different terminal.

For same-terminal checkout, EDF can be invoked either by:

1. Using the 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 on the current terminal, which must be in TRANSCEIVE status, 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=TERMINAL system macro) of the terminal on which the transaction requiring checkout is being run. For example:

```
CEDF L77A
```

If a command-level transaction is already running on that terminal, EDF will associate itself with that transaction; otherwise it will associate itself with the next command-level transaction started at that terminal.

The above applies to a single system. If the transaction running on the terminal has been transaction routed, EDF will not associate itself with it, nor with any other transaction that has been routed. EDF will associate itself with the next command-level transaction that runs on the system to which the terminal is connected.

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 terminal identifier is omitted, the terminal at which the CEDF transaction is initiated is assumed.

CEDF cannot be defined to be a remote transaction. The only way to test a transaction running in a connected system is by means of the routing transaction CRTE. This transaction is used to set up a routing session with the connected system; CEDF can then be used for same-terminal checkout.

To invoke EDF within the routing session, the user must type CEDF because the routing session does not allow the use of PA or PF keys. It is impossible to use EDF for two-terminal checkout if the transaction under test, or the terminal that invokes it, is owned by a different system.

USING EDF DISPLAYS

An example of a typical EDF display is given in Figure 3 on page 38.

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 function specified for the ENTER key is performed.

EDF uses the line immediately above the menu to display messages to the user.

Up to ten displays are remembered and can be redisplayed later.

The number at the top right of the screen indicates the current display number; it is possible to recall any of the last ten displays, which are numbered -01, -02, and so on, by overtyping this number. Alternatively, PF7 and PF8 can be used to scroll back and forward one display at a time.

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 when 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."

The response of EDF to a user request is in accordance with the following order of priority:

```
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 3. Typical EDF Display

1. If the CLEAR key is used, EDF redisplays the screen with any changes ignored.
2. If invalid changes are made, EDF accepts any valid changes and redisplays the screen with a diagnostic message.
3. If the display number is changed, EDF accepts any other changes and displays the requested display.
4. If a PF key is used, EDF accepts any changes and performs the action requested by the PF key.
5. If the ENTER key is pressed, and the screen has been modified (other than the REPLY field), EDF redisplays the screen with changes included.
6. If the ENTER key is pressed, and the screen has not been modified (other than the REPLY field), then if the ENTER key means CONTINUE, execution of the user transaction continues, otherwise if the ENTER key means CURRENT DISPLAY, EDF redisplays the current status display.

TERMINAL SHARING BETWEEN TRANSACTION AND EDF

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.

For same-terminal use, when EDF restores the transaction display on a device that uses color, programmed symbols, or extended highlighting, the attributes will no longer be present and the display will be in monochrome with no programmed symbols, or extended highlighting.

If the inbound reply mode in the application program is set to character (to enable the attribute setting keys)

EDF will reset this mode causing these keys to be disabled.

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.

ENTER AND PF KEYS

The following list explains the meanings of the ENTER key and the program function (PF) keys:

ABEND USER TASK
terminates the task. EDF asks the user to confirm this action by displaying 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. In the latter case, EDF redisplays the screen with changes 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 modified. In the latter case, EDF redisplays the screen with changes incorporated.

DIB DISPLAY
shows the contents of the DIB.

EIB DISPLAY
shows the contents of the EIB and COMMAREA (if any) (see "Appendix A. EXEC Interface Block" on page 239 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.

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.

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:

- PSW at abend (8 bytes)
- 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, as shown in Figure 4 on page 42, 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. 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:

- 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.

The line number, which will be available on the source listing if the program has been 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.

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; 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, as follows:

ASM
the storage defined in the current DFHEISTG DSECT.

COBOL
all data storage defined in the WORKING-STORAGE section of the program.

PL/I
the dynamic storage area (DSA) of the current procedure.

Except for COBOL programs, 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 task in 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 an 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 suppresses execution of the command. When the screen is redisplayed with NOOP, the original verb line can be restored by erasing the whole verb line with 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 hexadecimal 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.

- 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.

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 alarm is sounded if the terminal has the alarm feature;

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

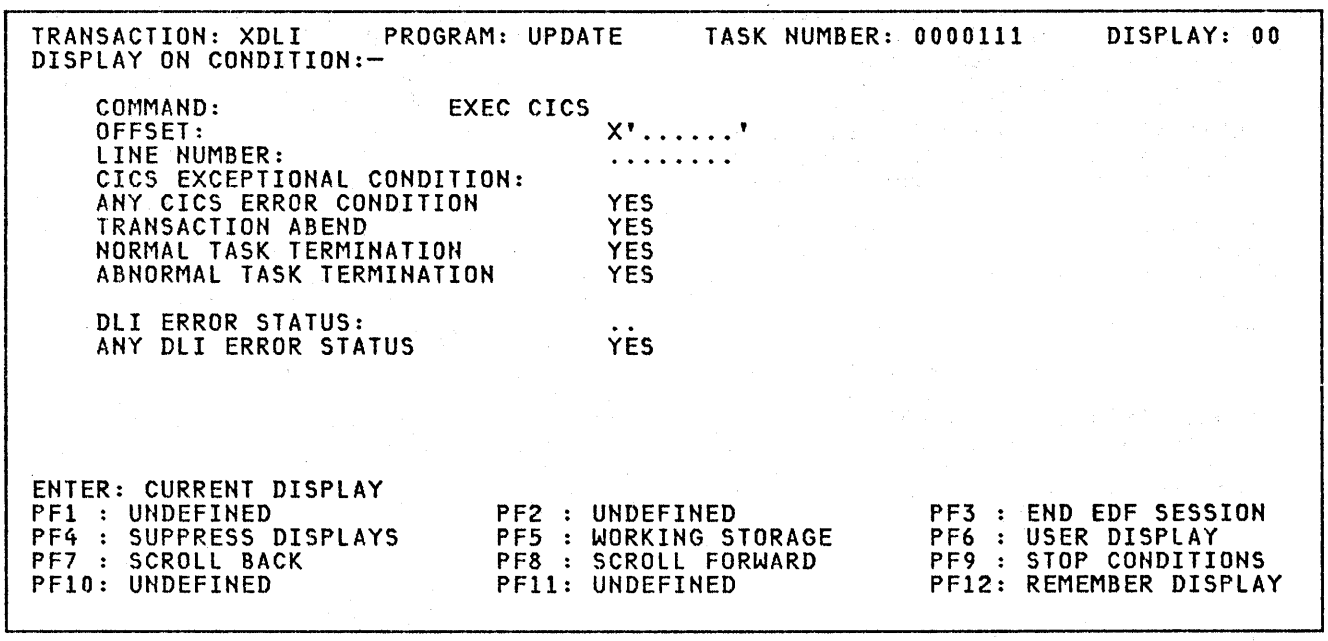


Figure 4. "Stop-Conditions" Display

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. 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 assembler language, the offset of the program label is displayed; for example, ERROR (X'00030C')
- 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')

If no label value is specified on a HANDLE CONDITION command, EDF displays the condition name alone.

USING EDF WITH EXEC DLI COMMANDS

EDF supports EXEC DLI commands in the same way as it supports EXEC CICS commands. However, the following minor differences should be noted:

- The two-character DL/I status code appears in the RESPONSE field and the EIBRCODE field is not displayed. The status code can be displayed in character or hexadecimal format. If the status code is changed to an invalid value, or to a value that would have caused DL/I to abend the user task, a warning message is issued before continuing the user task.
- For commands that generate more than one CALL statement, the offset is that of the last CALL.
- For the WHERE option, only the keyfield value (the third component) can be converted to hexadecimal. The address shown for this option is that of the keyfield value.
- The line number of the command is always displayed.
- For transactions that contain EXEC DLI commands, the DL/I interface block can be displayed, and additional stop conditions can be specified.

Examples of typical displays for an EXEC DLI command are given in Figure 5 on page 43 and Figure 6 on page 43.

```
TRANSACTION: XDLI     PROGRAM: UPDATE     TASK NUMBER: 0000111     DISPLAY: 00
STATUS:  COMMAND EXECUTION COMPLETE
EXEC DLI GET NEXT
 USING PCB (+00003)

 FIRST
 SEGMENT ('A     ')
 INTO ('              ')
 SEGLENGTH (+00012)

 FIRST
 VARIABLE
+SEGMENT ('B      ')



 OFFSET:X'000246'   LINE:  00000510          EIBFN:X'000C'
 RESPONSE:  'AD'

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 5. First Page of Typical EXEC DLI Display

```
TRANSACTION: XDLI     PROGRAM: UPDATE     TASK NUMBER: 0000111     DISPLAY: 00
STATUS: COMMAND EXECUTION COMPLETE
EXEC DLI GET NEXT
+
 FIRST
 SEGMENT ('C        ')
 SEGLENGTH (+00010)
 LOCKED
 INTO ('SMITH    ')
 WHERE (ACCOUNT = '12345')
 FIELDLENGTH (+00005)




 OFFSET:X'000246'   LINE: 00000510          EIBFN:X'000C'
 RESPONSE: 'AD'

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 6. Second Page of Typical EXEC DLI Display

Chapter 1.8. Command-Level Interpreter

The command-level interpreter enables CICS/VS commands to be entered, syntax-checked, and executed interactively at a 3270 screen. The interpreter performs a dual role in the operation of a CICS/VS system.

- For the application programmer, it provides a reference to the syntax of the whole of the CICS/VS command-level application programming interface (excluding DL/I). Most of the commands can be carried through to execution, and the results of execution can be displayed.
- For the system programmer, it provides a means of interaction with the system. For example, a corrupted data-base record can be "repaired", a temporary storage queue can be created or deleted, and so on. It thus provides a useful extension to the facilities provided by the master terminal transaction CEMT.

INVOKING THE COMMAND-LEVEL INTERPRETER

The command-level interpreter is a CICS/VS application program and runs as a CICS/VS transaction. It is started by the transaction identification of "CECI", or "CECS", followed optionally by the command.

The general format is:

```
CECI|CECS [command]
```

where "command" can be any of the CICS/VS commands (except EXEC DLI) described throughout this manual.

The use of CECI will give the full facilities of the interpreter right through to execution of the command.

For example, entering:

```
CECI READ DATASET('FILEA')
```

will give the screen display shown in Figure 7 on page 46.

Modifying the command input to:

```
READ DATASET('FILEA') RIDFLD('000001')
```

will give the screen display shown in Figure 8 on page 47. The error message has disappeared because the requested record identification field has been supplied.

The command is now ready to be executed, and this is achieved simply by pressing the ENTER key. The display shown in Figure 9 on page 48 will appear showing the result of execution.

It is possible to prevent unauthorized access by the interpreter to resources such as data sets. Refer to the security rules later in the chapter.

A question mark (?) before the command always gives the command syntax check display and prevents command execution.

The use of CECS forces a question mark before the command. This always gives the command syntax check display and prevents command execution. In a system where security is important, CECS can be made more widely available than CECI.

SCREEN LAYOUT

The command interpreter uses a basic screen layout of four areas, as shown in Figure 7 on page 46. These areas are:

- Command Input Area (the first line of the screen)
- Status Area (the second line of the screen)
- Information Area (21 lines on a 24 x 80 display)
- PF Key Values Area (the last line of the screen)

COMMAND INPUT AREA

This is the first line of the screen. The command, whose syntax is to be checked, or which is to be executed, is entered on this line, either in the normal format described in "Chapter 1.2. Command Format and Argument Values" on page 5 and as illustrated throughout this manual, or in an abbreviated or condensed form that reduces the number of keystrokes involved. The condensed form of the command is obtained as follows:

- The keywords EXEC CICS are optional.
- The options of a command can be abbreviated to any number of characters sufficient to make them unique. Valid abbreviations are shown in capital letters in syntax displays.

- The quotes around character strings are optional, and all strings of characters will be treated as character-string constants unless they are preceded by an ampersand (&) in which case they are treated as variables, as described later in the chapter.

- Options of a command that receive a value from CICS/VS when the command is executed are called "receivers", and need not be specified. The value received from CICS/VS will be included in the syntax display after the command has been executed.

The following example shows the condensed form of a command. The file control command:

```
EXEC CICS READ DATASET('FILEA')
RIDFLD('000001') INTO(data-area)
```

can be entered on the command input line, as:

```
READ DAT(FILEA) RID(000001)
```

or at a minimum, as:

```
READ D(FILEA) R(000001)
```

here, the INTO option is a receiver (as defined above), and can be omitted.

STATUS AREA

This is the second line of the screen. It will contain one of the following:

- COMMAND SYNTAX CHECK
- ABOUT TO EXECUTE COMMAND
- COMMAND EXECUTION COMPLETE (or COMMAND NOT EXECUTED)
- EIB DISPLAY
- VARIABLES
- ERROR MESSAGES
- EXPANDED AREA

This status line describes the type of information in the immediately following information area of the display.

INFORMATION AREA

This area consists of the remainder of the screen between the "command input" and "status" areas at the top, and "PF key values" at the bottom of the screen. This area is used to display the syntax of the entered command, error message information, the response to execution, and any other information that can be obtained by using the PF keys or the cursor.

A line at the bottom of this area is reserved for messages that describe errors in the conversation with the user (for example, "INVALID PACKED DECIMAL"). These messages are intensified to attract attention.

Command Syntax Check

When this status message appears (as shown in Figure 7), it indicates that the

```
READ DATASET('FILEA')
STATUS:  COMMAND SYNTAX CHECK                          NAME=
 EXEC CICS READ
  Dataset( 'FILEA    ' )
  SET() | Into()
  < Length() >
  Ridfld()
  < Keylength() < GEneric > >
  < SYsid() >
  < SEGset() | Segsetall >
  < RBa | RRn | DEBRec | DEBKey >
  < GTeq | Equal >
  < Update >




  DFH7052I  S  RIDFLD OPTION MUST BE SPECIFIED

PF: 1 HELP 2 HEX 3 END 4 EIB 5 VAR 6 USER 7 SBH 8 SFH 9 MSG 10 SB 11 SF
```

Figure 7. "Command Syntax Check" Display

command which has been entered on the command input line has been syntax checked but is not about to be executed. This will always be the status for CECS or for CECI with a question mark before the command. It is also the status when the syntax check of the command gives severe error messages and for those commands which are not executable (for example, HANDLE CONDITION and HANDLE AID).

The INFORMATION AREA of the display for "Command Syntax Check", "About to Execute Command", and "Command Execution Complete" contains information common to all three displays.

The full syntax of the command is displayed together with error information at the foot of the display. Options in the syntax panel are intensified to show those specified on the command input line, those assumed by default, and any "receivers".

The command on the command input line can be modified at any time by overtyping and pressing ENTER.

If the command has more options than can be held in one display, a plus sign (+) will appear at the left-hand side of the last option of the current display to indicate that there are more. These can be displayed by using one of the scrolling PF keys.

The syntax display differs slightly from the syntax shown throughout the manual in the following ways:

- Square brackets [ ] are replaced by the less-than and greater-than symbols < >.
- Braces { } are not used. If a mandatory option is omitted, an error message will be displayed and execution will not proceed until the option has been specified.
- Parentheses ( ) are used to indicate that an option requires a value or data field but none has been specified.

The error information consists either of a single error message or an indication of the number and severity of the messages generated.

The NAME= field on the syntax display can be used to create a variable containing the current command. (See the description of a variable later in the chapter.)

About to Execute Command

This display (as shown in Figure 8) appears when none of the reasons for stopping at Command Syntax Check apply. Option values can be modified by overtyping them in the syntax panel. This is a temporary modification for the duration of the command and does not affect the command input line. It is similar to the modification of option values that is possible with EDF when debugging an application program.

```
READ DATASET('FILEA') RIDFLD('000001')
STATUS:  ABOUT TO EXECUTE COMMAND                    NAME=
 EXEC CICS READ
  Dataset( 'FILEA   ' )
  SET() | Into()
  < Length() >
  Ridfld( '000001' )
  < Keylength() < GEneric > >
  < SYsid() >
  < SEGset() | Segsetall >
  < RBa | RRn | DEBRec | DEBKey >
  < GTeq | Equal >
  < Update >




PF: 1 HELP 2 HEX 3 END 4 EIB 5 VAR 6 USER 7 SBH 8 SFH 9 MSG 10 SB 11 SF
```

Figure 8. "About to Execute Command" Display

Command Execution Complete

This display (as shown in Figure 9) appears in response to the ENTER key after an "about to execute command" display. The command has been executed and the results are displayed on the screen. Any "receivers", whether specified or not, together with their CICS/VS-supplied values, are displayed intensified. If the value of an option is too long for the line, only the first part will be displayed followed by "..." to indicate there is more. Positioning the cursor, using the tab key, at the start of the option value and pressing ENTER will produce an expanded display of the whole option value.

Also displayed at the foot of the information area, is the appropriate response code (for example, NORMAL) together with the contents of the EIBRCODE field of the EIB.

Variables

This display will show, in response to pressing key PF5, all the variables associated with the current interpreter session, showing for each, its name, length, and value.

Normally, the value supplied for an option in the command input area is taken as a character-string constant. However, there is sometimes a requirement for this value to be represented by a variable. The command interpreter will recognize a value as a variable only if it is preceded by an ampersand (&).

A variable is required when two associated commands are to be connected through the values supplied in their options, for example, READ INTO(data-area) UPDATE and REWRITE FROM(data-area). A variable can be used to make the data area in the FROM option the same as that in the INTO option.

A variable is also useful when the values of options cause the command to exceed the line length of the command input area. Creating variables with the required values and specifying the variable names in the command will enable a command to be accommodated.

Variables can also be used to contain commands, and variable names can be entered in a command input line that contains complete or partial commands.

Variables are deleted at the end of an interpreter session unless action has been taken to save them, for example, in temporary storage, as described below.

Variables, which can be of data type character, fullword, halfword, or packed decimal, can be created, as follows:

1. By naming the variable in a receiver. The variable will be created when the command is executed. The data type is implied by the type of receiver.

2. By adding one or more new entries to the list of variables already

```
READ D(FILEA) R(000001)
STATUS:  COMMAND EXECUTION COMPLETE                    NAME=
 EXEC CICS READ
  Dataset( 'FILEA    ' )
  SET() | Into( 'U000001                                    ...' ... )
  < Length( +00080 ) >
  Ridfld( '000001' )
  < Keylength() < GEneric > >
  < SYsid() >
  < SEGset() | Segsetall >
  < RBa | RRn | DEBRec | DEBKey >
  < GTeq | Equal >
  < Update >




  RESPONSE: NORMAL                    EIBRCODE=X'00000000000'

PF: 1 HELP 2 HEX 3 END 4 EIB 5 VAR 6 USER 7 SBH 8 SFH 9 MSG 10 SB 11 SF
```

Figure 9. "Command Execution Complete" Display

defined. This list is displayed by pressing key PF5. The display shows all defined variables giving, for each, its name, length in bytes, and its value. The value is displayed in character form but PF2 can be used to switch from character to hexadecimal. An expanded display of each variable can be obtained by positioning the cursor under the & of the name and pressing ENTER. To create a new character variable, enter its name and its length and press ENTER. The variable will be initialized to blanks, which can then be overtyped. For a fullword, halfword, or packed variable, enter F, H, or P in the length field. These fields are initialized to zero.

Variable names, lengths, and their values, can be modified by overtyping. Variables can be deleted by positioning the cursor under the & of the name and pressing erase EOF. Variables can be copied by obtaining the expanded display of the variable and overtyping the name field.

3. By associating a variable name with the value of an option. Positioning the cursor, using the tab key, at the start of the line of the syntax display and pressing ENTER will produce an expanded display of the whole option value. A variable name can now be assigned to the data so displayed.

4. By entering a name in the NAME= field of the syntax panel. This will create a variable containing the current command.

Three variables are provided initially. The first, &DFHC, is a sample. The second, &DFHW, contains a temporary storage WRITEQ command, and the third, &DFHR, contains a READQ command. It is possible to write a command to temporary storage by entering &DFHC in the NAME= field of the syntax panel, entering &DFHW in the command input line, and executing the WRITEQ command. In this way, a list of commands can be written. The command list can be read and executed by alternately entering &DFHR and &DFHC in the command input line.

Expanded Area

This display will use the whole of the information area of the screen to display areas selected by means of the cursor. The cursor can be positioned at the start of the value of an option on a syntax display, or under the ampersand of a variable in a variables display. Pressing ENTER will then give the expanded area display. The scrolling keys can be used to display all the information if it exceeds a full screen.

ENTER KEY AND PF KEY VALUES

The single line at the foot of the screen provides a menu indicating the effect of the ENTER and PF keys for the display. Continuation of interpretation depends entirely upon use of the ENTER key; unless this key is pressed no further action will occur.

The PF keys are self-explanatory; if the terminal has no PF keys, the same effect can be obtained by positioning the cursor under the required item in the menu by means of the tab keys and pressing ENTER. The following PF keys are available:

PF1: HELP
displays a HELP panel giving more information on how to use the command interpreter and on the meanings of the PF keys.

PF2: 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.

PF3: END SESSION
ends the current session of the interpreter.

PF4: EIB DISPLAY
shows the contents of the EXEC interface block (EIB). (See "Appendix A. EXEC Interface Block" on page 239 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 BACK HALF
scrolls half a screenful backwards.

PF8: SCROLL FORWARD HALF
scrolls half a screenful forwards.

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

PF10: SCROLL BACK
scrolls backwards.

PF11: SCROLL FORWARD
scrolls forwards.

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

TERMINAL SHARING

When the command being interpreted is one that uses the screen which the interpreter is using, the command interpreter will manage the sharing of the screen between the interpreter display and the user display.

The user display will be restored:

- when the command being executed requires input from the operator.
- when the command being executed is about to modify the user display.
- 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, the interpreter will remember it and redisplay it when the screen is restored for the RECEIVE command.

When the interpreter restores the user display, it does not sound the alarm or affect the keyboard in the same way as when a SEND command is executed.

PROGRAM CONTROL

The interpreter is itself a CICS/VS application program and the execution of certain program control commands may cause different results from an application program containing those commands. For example, an EXEC CICS ABEND command will be intercepted by the interpreter rather than abending the interpreter (unless the CANCEL option is specified).

If the interpreter is used to LINK to a program, the interpreter will not be aware of modifications to the USER DISPLAY made by that program. If the interpreter executes an XCTL command, control will be transferred to that program and that will be the end of the interpreter session.

SECURITY RULES

To invoke the command interpreter, the user must have a security key that matches the security key defined in the PCT.

The command-level interpreter transaction identifier, CECI, specifies, by default, that resource level security checking is required for any resources referenced with the interpreter. This checking applies to data sets, transient data queues, temporary storage queues, programs, transaction identifiers of the START command, and journal file identifiers.

If the resource security level specified in the appropriate CICS/VS table (for example, the FCT for a dataset) is not matched by the authorization obtained from a sign-on, the resource security check fails, and the response to the command will be ABEND AEY7. This response is given on the "command execution complete" display.

INSTALLING THE COMMAND-LEVEL INTERPRETER

To ensure that the command interpreter is available on the system, the system programmer must make one group entry in the PPT and in the PCT. (See the CICS/VS System Programmer's Reference Manual for details on constructing a PPT and a PCT.)

Part 2. Data Base Operations

Chapter 2.1. Introduction to Data Base Operations

CICS/VS transactions can access two kinds of data bases, which can be on either a local or remote system, as follows:

- Standard operating system data sets holding a data base.
- DL/I (Data Language/I) data bases.

Standard operating system data sets are processed by the CICS/VS file control program, which permits the retrieval, addition, updating, deletion, and browsing of records in ISAM, VSAM, and DAM data sets. File control relieves the application programmer of buffer management, blocking and deblocking, and access-method dependencies. File control is described in "Chapter 2.2. File Control" on page 55.

A **DL/I data base** gives the application programmer a 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. DL/I offers powerful facilities for the manipulation of these segments without requiring the programmer to be aware of how they are organized.

Processing of a DL/I data base is performed by one of the following program products with which CICS/VS interfaces:

- For VSE users, Data Language/I DOS/VS (Program Number 5746-XX1).
- For OS/VS users, Information Management System/Virtual Storage (IMS/VS) (Program Number 5740-XX2).

The CICS/VS-DL/I interface for both VSE and OS, which is invoked by means of the DL/I CALL statement, is described in "Chapter 2.3. DL/I Services (DL/I CALL Statement)" on page 69.

The CICS/VS-DL/I interface for VSE only, which is invoked by means of the EXEC DLI command, is described in "Chapter 2.4. DL/I Services (EXEC DLI Command)" on page 77.

Chapter 2.2. File Control

The CICS/VS file control program processes fixed-length or variable-length, blocked or unblocked, undefined, or segmented records of a direct-access data set. (Sequential data sets are processed by the transient data control program, as described in "Chapter 4.6. Transient Data Control" on page 201).

File control uses the standard access methods of the host operating system (OS/VS or VSE), namely:

- Direct Access Method (DAM)
- Indexed Sequential Access Method (ISAM)
- Virtual Storage Access Method (VSAM).

Application programs can access DAM data sets on a logical-record level, deblocking services being provided by CICS/VS. If an ISAM data set is converted to a VSAM data set organization, using VSAM data set conversion utilities, no alteration to application programs that access the data set is necessary, but the file control table (FCT) must be changed. Data sets on fixed block architecture (FBA) devices can be accessed only by VSAM.

File control commands can be used to:

- Read a record from a data set (READ).
- Write a record to a data set (WRITE).
- Update a record in a data set (REWRITE).
- Delete a single record or a group of records from a key-sequenced or relative-record data set (DELETE) (VSAM only).
- Release exclusive control over a data set (UNLOCK).
- Specify the starting point for a browse (that is, sequentially access a data set) (STARTBR).
- Read the next record in a data set during a browse (READNEXT).
- Read the previous record in a data set during a browse (READPREV) (VSAM only).
- Reset the starting point for a browse (RESETBR).
- End a browse (ENDBR).

An option can be included in these commands to specify that the record to be accessed is in a data set on a remote system.

Exceptional conditions that occur during execution of a file control command are handled as described in Chapter 1.5.

The following sections discuss the identification of data sets to be used in file control operations; direct access to records in data sets; sequential access to records (browsing); and information particular to the access methods available (ISAM, VSAM, and DAM).

DATA SET IDENTIFICATION

Data sets are identified in file control commands by the DATASET option; they must have been defined previously in the file control table (FCT) unless, for a local system only, the SYSID option has been specified also, in which case a FCT definition is unnecessary. These definitions may be set up with the help of the system programmer, although 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 reading records 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 or SET has been specified.

The INTO option specifies 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 to this value and the LENGERR condition will occur. For fixed-length records, the LENGTH option must specify the length of the record, otherwise the LENGERR condition will occur. 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).

The SET option specifies a pointer reference that is set to the address of an area large enough to hold the record. After the record has been retrieved, the data area specified in the LENGTH option is set to the actual 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 or UNLOCK command should be issued as soon as possible to avoid obstructing file storage, and possibly preventing other transactions from accessing the record.

MULTIPLE FILE OPERATIONS

When accessing more than one file at a time, a lockout may occur, for example, if two tasks attempt to read the same record for update at the same time or when accessing files on a remote system. Assume the following:

```
Prog 1: READ UPDATE (File A)
        READ UPDATE (File B)

Prog 2: READ UPDATE (File B)
        READ UPDATE (File A)
```

Suppose that the two tasks become intermixed in multitasking, as follows:

```
Prog 1: READ UPDATE (File A, rec 338)
Prog 2: READ UPDATE (File B, rec 753)
Prog 1: READ UPDATE (File B, rec 753)
Prog 2: READ UPDATE (File A, rec 338)
```

The two tasks will both be suspended indefinitely, because each would have exclusive control of the first record requested by the other. The second request of each task cannot be completed. To avoid this problem, all programs should access the files in the same sequence, such as A first, followed by B.

SEQUENTIAL ACCESS TO RECORDS (BROWSING)

When reading 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, which would normally be, but need not be, the record specified in the STARTBR command. (For VSAM data sets, 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, described in a previous section.

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

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.

If records are unblocked, or have a very low blocking factor (which means that many file reads are done before displaying a page), it may be more efficient to display fewer records.

With a high blocking factor, fewer read operations are done, records merely being moved from a buffer area, so lengthy browses are not so inefficient.

A browse should always be terminated by an ENDBR command, but will, in any case, be terminated by a normal or abnormal end of task.

SEGMENTED RECORDS

An optional feature of CICS/VS file management allows the user to create and define a data set containing segmented records. A **segment** is one or more adjacent fields within a record. Some segments appear in all records while others appear in only certain records. Each record contains one segment (the root segment) which contains information about which other segments are present. Groups of segments can be defined and identified symbolically as **segment sets**. A record can be read with a specified segment set and only those segments of the record defined in that segment set are returned. The user cannot access segmented records in a data set on a remote system.

If it is planned to use segmented records the structure of individual segments and of segment sets must have been defined in the file control table by the system programmer, and the user must create and maintain the control information in the root segment of each record.

For further information on segmented records see the CICS/VS System/Application Design Guide.

ISAM DATA SETS

RECORD IDENTIFICATION

Records in ISAM data sets are identified by **key**. This key must be provided in a record identification field specified by the RIDFLD option.

For CICS/OS/VS systems, the contents of the record identification field may have been changed following the addition of a record; this point should be considered in CICS/DOS/VS systems also, to avoid future VSE to OS/VS conversion problems.

Records that are flagged for deletion are presented to the application program, which must be able to recognize them.

ADDING RECORDS TO ISAM DATA SETS

Adding records to an ISAM data set may degrade performance due to overflow accesses; also data sets may be destroyed undetected, if for example, a power failure occurs, or CICS/VS terminates abnormally. If such a failure occurs when adding records, records may be lost and overflow chains destroyed. To prevent these problems, consider one of the following:

- Memo posting. This is a technique that uses special memo fields created in each record of a file. All fields that are normally updated by changing quantities, such as the number of items, amounts, and so on, are recorded in these memo fields, so that system failures affect only these memo fields. All changes must be posted to a log file, so that the data file can be updated later on a batch basis. This ensures the integrity of the data file while retaining the advantages of online posting.
- Using a file copy. A copy of the data file is provided for use with CICS/VS. This allows the addition and deletion of records and modification of any data in the file without affecting the file integrity. All changes must be posted to a log file, so that the data file can be updated later on a batch basis. This ensures the integrity of the data file and allows complete online file maintenance.

ISAM EXCLUSIVE CONTROL

When an ISAM record is read for update, CICS/VS maintains exclusive control of the record. An attempt to re-read the record before it is updated (by a REWRITE command), or before exclusive control is released (by an UNLOCK command), will cause a lockout.

ISAM BROWSING OPERATIONS

A browse can be started at any record in an ISAM data set. A complete key of hexadecimal zeros, or the options KEYLENGTH(0) and GENERIC, will start the browse at the first record. Any other starting point must be specified in the RIDFLD option of the STARTBR or RESETBR command. The key provided can be a complete (specific) key or a generic (partial) key.

If a complete key is provided, the browse starts with the record having that key. If this record cannot be found, then by default, the browse starts with the first record having a key greater than the specified key.

If a generic key is provided, its length must be specified in the KEYLENGTH option, and the GENERIC option also must be specified. The search for the starting record uses only the number of characters in this key. The first record having a matching generic key is the starting point. If this record cannot be found, then by default, the browse starts with the first record having a generic key greater than the specified generic key.

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

Records flagged for deletion are presented to the application program, which must be able to recognize them.

VSAM DATA SETS

INITIALIZATION OF VSAM DATA SETS

When creating a VSAM key-sequenced data set for use with CICS/VS, at least one dummy record must be loaded into the data set before it can be processed by CICS/VS.

RECORD IDENTIFICATION

Records in VSAM data sets 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 to a VSAM data set, a complete key must be provided.

When reading records in inquiry mode, the search key can be a complete key or a generic key, and, for either type, the search can be for an equal key (EQUAL option) or a greater-or-equal key (GTEQ option).

When reading records for update, the search key should be a complete key, and the search should be for an equal key (EQUAL option).

If a complete key is specified, the record having that key is retrieved; if it cannot be found and the GTEQ option is specified, the first record having a key greater than the specified key is retrieved, otherwise the NOTFND exceptional condition occurs. The complete key is returned in the record identification field after the record has been retrieved.

If a generic key is specified, its length must be specified in the KEYLENGTH option, and the GENERIC option also must be specified. The search for the required record uses only the number of characters in the generic key. The first record having a matching generic key is retrieved; if no matching record is found, and the GTEQ option is specified, the first record having a generic key greater than the specified generic key is retrieved, otherwise 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 to read a second record for update or add a new record to the same control interval before exclusive control is released, would cause a lockout.

When local resources are shared, a lockout will also occur if an attempt is made to read two records (one of them for update) from the same control interval.

CICS/VS prevents such a lockout by raising the INVREQ condition if, following the first READ UPDATE command, a second READ UPDATE command, or a WRITE command is issued for the same data set and within the same transaction before exclusive control is released (by a REWRITE, UNLOCK, or DELETE command).

DELETION OF VSAM RECORDS

Records in a VSAM key-sequenced or relative-record data set can be deleted, either singly or in groups, using the DELETE command. Single records are identified by key, relative byte address, or relative record number. Groups of records can be deleted only if the data set is unprotected, and 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 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 INVREQ condition will occur.

VSAM MASS SEQUENTIAL INSERTION

The MASSINSERT option is used to specify that a VSAM mass sequential insertion operation is in progress; it must be specified in every WRITE command that is part of the operation.

A mass insert operation must be terminated (by an UNLOCK command) to ensure that all records are written to the data set; a READ command will not necessarily retrieve a record that has been added by an incomplete mass insert operation. Incomplete operations will be terminated when the task terminates.

A lockout will occur if more that one transaction is attempting simultaneously to perform a mass insert operation to the same control interval of a protected data set. A lockout will occur also if a transaction uses keys that are not in ascending sequence.

VSAM BROWSING OPERATIONS

A VSAM data set can be browsed in either direction.

A record identification field of hexadecimal zeros, or the options KEYLENGTH(0) and GENERIC in a STARTBR or RESETBR command, will start a forward browse at the first record.

A record identification field of hexadecimal 'FF's will start a backward browse at the last 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. There is one exception; a backward browse

cannot be specified if the previous STARTBR command has the GENERIC option.

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.

When browsing a protected data set (LOG=YES specified in the DFHFCT TYPE=DATASET macro by the system programmer), an end browse (ENDBR) command must be issued before issuing a READ UPDATE command.

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 data set 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 are identified in the DFHFCT TYPE=SHRCTL macro instruction, as explained in the CICS/VS System Programmer's Reference Manual. When a task requires resources in several VSAM data sets at the same time and these data sets are sharing resources, the probability of a lockout increases.

VSAM ALTERNATE INDEXES

The VSAM Alternate Index feature allows access to a data set using several indexes, which contain alternate keys to the records in the data set. 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 data set, unless records having non-unique alternate keys are involved. If the (alternate) key provided in a READ, READNEXT, or READPREV command is not unique, the first record in the data set having that key is read, and the DUPKEY condition occurs. To retrieve other records having the same key, a browse should be started, the subsequent READNEXT commands reading the records in the order in which they were added to the data set. (READPREV commands could be used, but the records will be returned in the same order as for READNEXT commands.)

When switching from direct retrieval (READ) to a browse (READNEXT), the first record having a non-unique key is read twice: once for the READ command, and again for the first READNEXT command.

The DUPKEY condition occurs for each retrieval operation except the last. For example, if there are three records with the same alternate key, DUPKEY occurs for the first two records, but not for the third. The application program can be designed to revert to direct retrieval operations when DUPKEY no longer occurs.

DAM DATA SETS

RECORD IDENTIFICATION

Records in DAM data sets are identified by a **block reference**, a **physical key** (keyed data set), and a **deblocking argument** (blocked data set). The record identification (specified in the RIDFLD option) contains a subfield for each, which, when used must be in the above order. The subfields are as follows:

Block reference - one of the following:

- Relative block address (CICS/OS/VS only): three-byte binary (RELTYPE=BLK).
- Relative track and record (hexadecimal format): two-byte TT, one-byte R (RELTYPE=HEX).

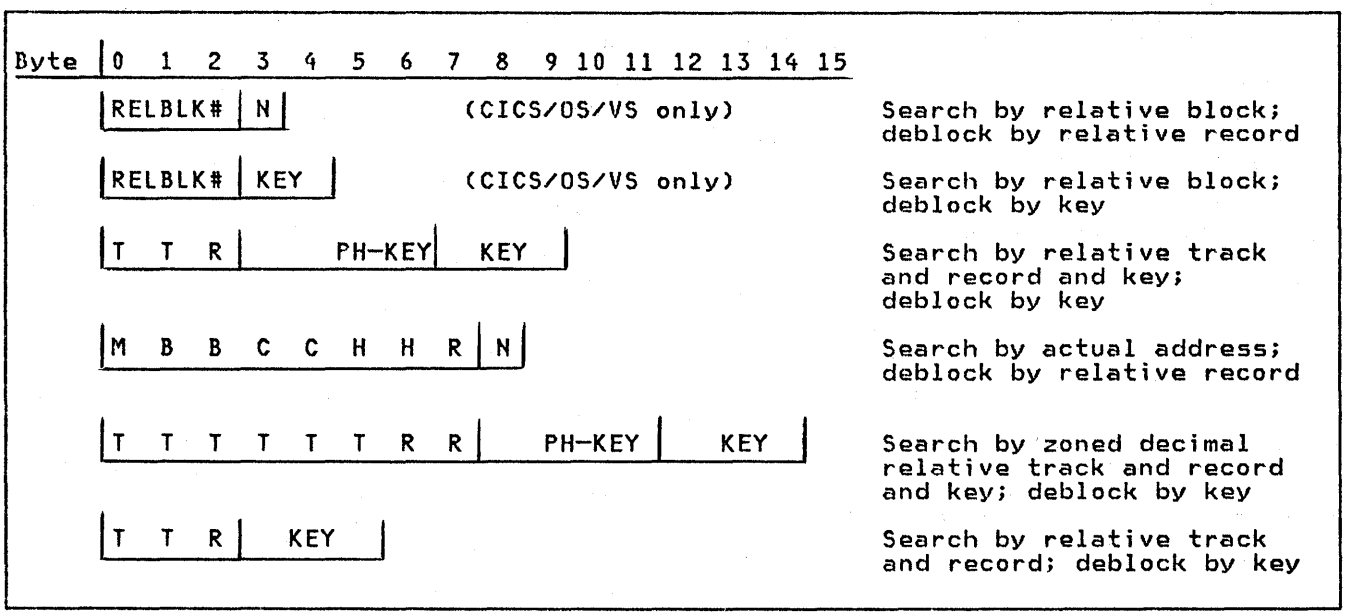


Figure 10. Examples of Record Indentification

- Relative track and record (zoned decimal format): six-byte TTTTTT, two-byte RR (RELTYPE=DEC).
- Actual (absolute) address: eight-byte MBBCCHHR (RELTYPE operand omitted).

The type of block reference being used must be specified in the RELTYPE operand of the DFHFCT TYPE=DATASET system macro which defines the data set.

Physical key - required only if the data set has recorded keys. If used, it must immediately follow the block reference. Its length must be the same as the length specified in the BLKKEYL operand of the DFHFCT TYPE=DATASET system macro that defines the data set.

Deblocking argument - required only if specific records are to be retrieved from a block. If used, it must follow immediately the physical key (if present) or the block reference. If omitted, an entire block will be retrieved.

The deblocking argument may be either a key (specify DEBKEY), in which case its length must be the same as that specified in the KEYLEN operand of the DFHFCT TYPE=DATASET system macro, or it may be a relative record number (specify DEBREC), in which case it is a one-byte binary number (first record=0).

The examples in Figure 10 assume a physical key of four bytes and a deblocking argument of three bytes.

ADDING RECORDS TO DAM DATA SETS

When adding records to a DAM data set, the following considerations and restrictions apply:

1. When adding undefined or variable-length records (keyed or non-keyed), the track on which each new record is to be added must be specified. If space is available on the track, the record is written following the last previously written record, and the record number is placed in the "R" portion of the record identification field of the record. The track specification may be in any of the acceptable formats except relative block. If zoned decimal relative format is used, the record number is returned as a two-byte zoned decimal number in the seventh and eighth positions of the record identification field.

   In a CICS/DOS/VS system, an attempt to add undefined or variable-length records is limited to the single track specified. If insufficient space is available on that track, the NOSPACE condition occurs. However, an attempt may be made to add the record on another track simply by altering the track identifier and using another WRITE command.

   In a CICS/OS/VS system, the extended search option allows the record to be added to another track if no space is available on the specified track. The location at which the record is added is returned to the application

program in the record identification field being used.

When adding records of undefined length, the length of the record must be specified in the LENGTH option. When an undefined record is retrieved, the application program must determine its length.

2. When adding keyed fixed-length records the data set must first be formatted with dummy records or "slots" into which the records may be added. (A dummy record is signified by a key of hexadecimal 'FF's; in a CICS/OS/VS system, the first byte of data contains the record number.)

3. When adding non-keyed fixed-length records the block reference must be given in the record identification field. The new records are written in the location specified, destroying the previous contents of that location.

4. When adding keyed fixed-length records track information only is used to search for a dummy key and record, which, when found, is replaced by the new key and record. The location of the new record is returned to the application program in the block reference subfield of the record identification field.

   For example, for a record whose identification field is as follows:

   ```
   0 3 0  ALPHA
   T T R  KEY
   ```

   the search will start at relative track 3. When control is returned to the application program, the record identification field will be as follows:

   ```
   0 4 6  ALPHA
   ```

   showing that the record is now record 6 on relative track 4.

5. When adding variable-length blocked records a four-byte record description field (RDF) must be included in each record. The first two bytes specify the length of the record (including the 4-byte RDF); the other two bytes consist of zeros.

DAM EXCLUSIVE CONTROL

When a blocked record is read for update, CICS/VS maintains exclusive control of the containing block. An attempt to read a second record from the block before the first is updated (by a REWRITE command), or before exclusive control is released (by an UNLOCK command), will cause a lockout.

DAM BROWSING OPERATIONS

The record identification field must contain a block reference (for example, TTR or MBBCCHHR) that conforms to the addressing method defined for the data set. Processing begins with the specified block and continues with each subsequent block until the browse is terminated. If the data set contains blocked records, processing begins at the first record of the first block and continues with each subsequent record, regardless of the contents of the record identification field.

The record identification field is updated by CICS/VS with the complete identification of each record retrieved by a READNEXT command. For example, assume a browse is to start with the first record of a blocked, keyed data set. Before issuing the STARTBR command, the TTR (assuming that is the addressing method) of the first block should be placed in the record identification field. After the first READNEXT command, the record identification field might contain

```
X'0000010504'
```

where 000001 represents the TTR value, 05 represents the block key, and 04 represents the record key.

As another example, assume that a blocked, non-keyed data set is being browsed, and the second record from the second physical block on the third relative track is read by a READNEXT command. Upon return to the application program, the record identification field contains

```
X'0000020201'
```

where 000002 represents the track, 02 represents the block, and 01 represents the record within the block.

KEYLENGTHS FOR REMOTE DATA SETS

In general, execution of file control commands requires the RIDFLD and KEYLENGTH options to be specified. KEYLENGTH may be specified explicitly in the command, or it may be determined implicitly from the file control table (FCT).

For remote data sets however, KEYLENGTH should be specified whenever SYSID and RIDFLD are specified, unless either RBA or RRN is specified, (when it is invalid), or if the command is a READNEXT or READPREV, (when it is not required).

For a remote DAM data set, where the DEBKEY or DEBREC options have been

specified, KEYLENGTH (when specified explicitly) should be the total length of the key (that is, all specified subfields). If the value of KEYLENGTH is taken from the FCT, the system programmer must ensure that the default for the KEYLENGTH value is equal to the DEBKEY value; again this value must be the total length of the key.

For relative-record data sets, the system programmer should specify KEYLEN=4 in the DFHFCT TYPE=REMOTE system macro. This will allow an application program translated on Version 1.3 to be executed on succeeding versions without retranslation.

READ A RECORD (READ)

```
READ DATASET(name)
{SET(ptr-ref)|INTO(data-area)}
[LENGTH(data-area)]
RIDFLD(data-area)
[KEYLENGTH(data-value)[GENERIC]]
[SYSID(name)]
[SEGSET(name)|SEGSETALL]
[RBA|RRN] (VSAM only)
[DEBKEY|DEBREC] (blocked DAM only)
[GTEQ|EQUAL] (VSAM only)
[UPDATE]

Conditions:  DSIDERR, DUPKEY,
ILLOGIC (VSAM only), OMVREQ, IOERR
ISCINVREQ, LENGERR, NOTFND, NOTOPEN
SEGIDERR, SYSIDERR
```

This command is used to read a record from a direct-access data set on a local or remote system.

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

```
EXEC CICS READ
     INTO(RECORD)
     DATASET('MASTER')
     RIDFLD(ACCTNO)
```

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

```
EXEC CICS READ
     INTO(RECORD)
     LENGTH(RECLEN)
     DATASET('MASTVSAM')
     RIDFLD(ACCTNO)
     KEYLENGTH(4)
     GENERIC
     GTEQ
     UPDATE
```

If more than one READ command with the UPDATE option is executed without corresponding REWRITE commands, a unique record identification field must exist for each to preserve the correct key for subsequent execution of the REWRITE commands.

Note that the last example above would fail if the data set is protected (LOG=YES specified in the DFHFCT TYPE=DATASET system macro), because a generic key cannot be used with READ UPDATE on a protected data set.

WRITE A RECORD (WRITE)

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

Conditions:  DSIDERR, DUPREC,
ILLOGIC (VSAM only), INVREQ, IOERR,
ISCINVREQ, LENGERR, NOSPACE, NOTOPEN,
SYSIDERR
```

This command is used to write a record to a direct-access data set on a local or remote system. For example:

```
EXEC CICS WRITE
     FROM(RECORD)
     LENGTH(DATLEN)
     DATASET('MASTER')
     RIDFLD(KEYFLD)
```

For a VSAM entry-sequenced data set (ESDS) the record is always added at the end of the data set, its relative byte address (RBA) being placed in the record identification field specified in the RIDFLD option.

For a VSAM key-sequenced data set (KSDS), the record is added in the location specified by the associated key; this location may be anywhere in the data set.

Records for entry-sequenced and key-sequenced data sets can be either fixed length or variable length. Those for a relative record data set must be fixed length.

UPDATE A RECORD (REWRITE)

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

Conditions:  DSIDERR, DUPREC,
ILLOGIC (VSAM only),INVREQ, IOERR,
ISCINVREQ, LENGERR, NOSPACE, NOTFND,
NOTOPEN, SYSIDERR
```

This command is used to update a record in a direct-access data set on a local or remote system. The record to be updated must first be read by a READ command with the UPDATE option. For example:

```
EXEC CICS REWRITE
     FROM(RECORD)
     DATASET('MASTER')
```

DELETE A VSAM RECORD (DELETE)

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

Conditions:  DSIDERR, ILLOGIC,
INVREQ, IOERR, ISCINVREQ, NOTFND,
NOTOPEN, SYSIDERR
```

This command is used to delete a record from a key-sequenced or relative-record data set on a local or remote system. The record to be deleted must be identified by means of the RIDFLD option.

A record that has been retrieved for update (by a READ UPDATE command) can also be deleted, instead of being rewritten, by the DELETE command. Since the record has already been identified, there is no need to specify RIDFLD.

Groups of records can be deleted in a similar way. A group is identified by the GENERIC option.

A generic key must not be used for data sets for which LOG=YES has been specified in the DFHFCT TYPE=DATASET macro by the system programmer.

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

```
EXEC CICS DELETE
     DATASET('MASTVSAM')
     RIDFLD(ACCTNO)
     KEYLENGTH(4)
     GENERIC
     NUMREC(NUMDEL)
```

RELEASE EXCLUSIVE CONTROL (UNLOCK)

```
UNLOCK DATASET(name)
       [SYSID(name)]

Conditions:  DSIDERR,
ILLOGIC (VSAM only), IOERR,
ISCINVREQ, NOTOPEN, SYSIDERR
```

This command is used to release exclusive control arrangements made in response to a READ command with the UPDATE option. It is used when a record has been retrieved for update and it is subsequently determined that the update should not occur. The effect is to allow other application programs to access the record that was to be updated. However, for a data set for which auto logging has been specified by the system programmer, the resource remains under the task control enqueue until either a sync point command is executed, or the task is terminated. The record can be in a data set on a local or remote system.

This command is also used to terminate a VSAM mass insert operation.

START BROWSE (STARTBR)

```
STARTBR DATASET(name)
[RIDFLD(data-area)
[KEYLENGTH(data-value)[GENERIC]]
REQID(data-value)
[SYSID(name)]
[RBA|RRN] (VSAM only)
[DEBKEY|DEBREC] (blocked DAM only)
[GTEQ|EQUAL] (VSAM only)

Conditions:  DSIDERR,
ILLOGIC (VSAM only), INVREQ, IOERR,
ISCINVREQ, NOTFND, NOTOPEN, SYSIDERR
```

This command is used to specify the record in a data set, on a local or remote system, at which the browse is to start. No records will be read until a READNEXT command (or, for VSAM only, a READPREV command) is executed.

READ NEXT RECORD DURING A BROWSE (READNEXT)

```
READNEXT DATASET(name)
{SET(ptr-ref)|INTO(data-area)}
[LENGTH(data-area)]
RIDFLD(data-area)
[KEYLENGTH(data-value)]
REQID (data-value)
[SYSID(name)]
[SEGSET(name)|SEGSETALL]
[RBA|RRN]                    (VSAM only)

Conditions:  DSIDERR, DUPKEY,
ENDFILE, ILLOGIC (VSAM only), INVREQ,
IOERR, ISCOMVREQ, LENGERR, NOTFND,
NOTOPEN, SEGIDERR, SYSIDERR
```

This command is used to read records in sequential order from a data set on a local or remote system. It can also be used during VSAM skip-sequential processing.

The RIDFLD option must specify the same data area as that specified in the RIDFLD option in the corresponding STARTBR command, but the contents of the data area can be different. If the NOTFND condition occurs during a browse, a RESETBR command must be issued to reset the browse, or an ENDBR command must be issued to terminate the browse.

READ PREVIOUS RECORD DURING A BROWSE (READPREV) (VSAM ONLY)

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

Conditions:  DSIDERR, DUPKEY,
ENDFILE, ILLOGIC, INVREQ, IOERR,
ISCINVREQ, LENGERR, NOTFND, NOTOPEN,
SEGIDERR, SYSIDERR
```

This command is used only to read records in reverse sequential order from a VSAM data set on a local or remote system.

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

If a READPREV command follows immediately a STARTBR command, the latter must specify the key of a record that exists on the data set, otherwise the NOTFND condition will be raised for the READPREV command.

A READPREV command following a READNEXT command will read the same record as that read by the READNEXT command.

RESET START OF BROWSE (RESETBR)

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

Conditions:  ILLOGIC (VSAM only),
INVREQ, IOERR, ISCINVREQ, NOTFND,
NOTOPEN, SYSIDERR
```

This command is used to specify the record in a data set, on a local or remote system, at which the browse is to be restarted.

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

The RESETBR command can be issued at any time prior to issuing a command. It is similar to an ENDBR STARTBR sequence (but with less function), and gives the ISAM and DAM user the sort of skip-sequential capability that is available to VSAM users through use of the READNEXT command.

END BROWSE (ENDBR)

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

Conditions:  ILLOGIC (VSAM only),
INVREQ, ISCINVREQ, SYSIDERR
```

This command is used to end a browse on a data set on a local or remote system.

FILE CONTROL OPTIONS

DATASET(name)
specifies the symbolic name of the data set to be accessed. The name must be alphameric, up to seven characters in length for DOS, up to eight characters in length for OS, and must have been defined in the file control table (FCT).

If SYSID is specified, the data set is assumed to be on a remote system irrespective of whether or not the name is defined in the FCT. Otherwise, the FCT entry will be used to determine if the data set is on a local or remote system.

DEBKEY (blocked DAM only)
specifies that deblocking is to occur by key. If neither DEBREC nor DEBKEY is specified, deblocking does not occur.

If KEYLENGTH is specified, its value must be the sum of the lengths of all three subfields comprising the key.

DEBREC (blocked DAM only)
specifies that deblocking is to occur by relative record (relative to zero). If neither DEBREC nor DEBKEY is specified, deblocking does not occur.

If KEYLENGTH is specified, its value must be the sum of the lengths of all three subfields comprising the key.

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

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

GENERIC (ISAM, VSAM only)
specifies that the search key is a generic key whose length is specified in the KEYLENGTH option. The search for a record is satisfied when a record is found that has the same starting characters (generic key) as that specified. For VSAM, this search will only take place if the EQUAL option also has been specified.

A generic key cannot be used with a READ UPDATE command or a DELETE command if the data set is protected (LOG=YES specified in the DFHFCT TYPE=DATASET system macro).

GTEQ (VSAM only)
specifies that if the search for a record having the same key (complete or generic) 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 data set is to be written.

KEYLENGTH(data-value)
specifies the length (halfword binary) of the key that has been specified in the RIDFLD option, except when RBA or RRN is specified when it is invalid. This option must be specified if GENERIC is specified, and it can be specified whenever a key is specified. However, if the length specified is different from the length specified in the FCT and the operation is not generic, the INVREQ condition occurs.

If KEYLENGTH is omitted from a READNEXT or READPREV command used in a generic browse, normal browsing occurs.

If KEYLENGTH is specified in a READNEXT or READPREV command used in a generic browse, a new browse is started using the keylength specified and the key in the RIDFLD option.

The use of KEYLENGTH with remote data sets is discussed earlier in the chapter.

LENGTH(parameter)
specifies the length (as a halfword binary value) of the record to be used with READ, READNEXT, READPREV, REWRITE, and WRITE commands. This option must be specified if SYSID and either INTO or FROM are specified.

For a READ, READNEXT, or READPREV command with the INTO option, the parameter must be a data area that specifies the largest record the program will accept. If the value specified is less than zero, zero is assumed. If the record exceeds the value specified, it is truncated to that value and the LENGERR condition occurs. On completion of the retrieval operation, the data area is reset to the original length of the record.

For a READ, READNEXT, or READPREV 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 record, except for a record whose format is undefined, when it is set to the maximum record length.

For a WRITE or REWRITE command, the parameter must be a data value that is the length of the record that is to be written.

This option need not be specified for fixed-length records when the length is known and a data area of the correct size is available.

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

NUMREC(data-area)
specifies a halfword binary data area that is to be set to the number of records deleted.

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

REQID(data-value)
specifies as a halfword binary value a unique request identifier for a browse, used to control multiple browse operations on a 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 data sets), a relative byte address or relative record number (for VSAM data sets), or a block reference, physical key, and deblocking argument (for DAM data sets). For a relative byte address or a relative record number, the format of this field must be fullword binary. For a key, the format must be that of the key of the record when adding records.

RRN (VSAM only)
specifies that the record identification field specified in the RIDFLD option contains a relative record number. This option should only be used with relative record data sets.

SEGSET(name)
specifies the name of the segment set to be retrieved. The name may be up to eight characters and must have been defined in the segment control section of the FCT. The data set must contain segmented records. SEGSET cannot be used with UPDATE.

SEGSETALL
specifies that the entire record in an unpacked and aligned format is required. The data set must contain segmented records. If neither SEGSET nor SEGSETALL is specified in a command, and the data set contains segmented records, the record is returned in its packed unaligned format.

SET(ptr-ref)
specifies the pointer-reference which is to be set to the address of the retrieved record. This option implies locate-mode access.

In assembler language, if the DUPKEY exceptional condition occurs, the register specified will not have been set, but can be loaded from DFHEITP1.

SYSID(name)
specifies the name of the system whose resources are to be used for intercommunication facilities. The name may be up to four characters in length.

When this option is specified, LENGTH and KEYLENGTH must be specified in some situations where normally they need not be, as follows. If neither RBA nor RRN is specified, KEYLENGTH must be specified; it cannot be found in the FCT. If SET is not specified, LENGTH must either be specified explicitly or must be capable of being defaulted from the INTO or FROM option using the length attribute reference in assembler language, or STG and CSTG in PL/I. LENGTH must be specified explicitly for COBOL.

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

FILE CONTROL EXCEPTIONAL CONDITIONS

DSIDERR
occurs if a data set name referred to in the DATASET option cannot be found in the FCT.

Default action: terminate the task abnormally.

DUPKEY
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 command that reads the last of the records having the non-unique key.

In assembler language, if the SET option is being used, the register specified will not have been set, but can be loaded from DFHEITP1.

Default action: terminate the task abnormally.

DUPREC
occurs if an attempt is made to add a record to a data set 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.

Default action: terminate the task abnormally.

ILLOGIC (VSAM only)
occurs if a VSAM error occurs that does not fall within one of the other CICS/VS response categories. Further information is available in the EXEC interface block (refer to "Appendix A. EXEC Interface Block" on page 239 for details).

Default action: terminate the task abnormally.

INVREQ
occurs if any of the following situations exist:

- A requested file control operation is not provided for or allowed according to the data set entry specification in the FCT.
- A REWRITE command, or a DELETE command without the RIDFLD option, is issued for a data set for which no previous READ UPDATE command has been issued.
- A READNEXT, READPREV, ENDBR, or RESETBR command is issued for a data set for which no previous STARTBR command has been issued.
- A READPREV command is issued for a data set for which the previous STARTBR command has the GENERIC option.
- The KEYLENGTH option is specified (but the GENERIC option is not specified), and the specified length does not equal the entry in the FCT for the data set.
- The KEYLENGTH and GENERIC options are specified, and the length specified in the KEYLENGTH option is either less than zero, or greater than or equal to the length in the FCT entry.
- A DELETE command is issued for an ISAM or DAM data set.
- A DELETE command with the RIDFLD option specified is issued for a VSAM data set when a READ UPDATE command is outstanding.
- Following a READ UPDATE command for a data set, a WRITE or READ UPDATE command is issued for the same data set before exclusive control is released by a REWRITE, UNLOCK, or DELETE command.
- The data area specified in the RIDFLD option is not the same one in all the commands of a browse.
- An attempt is made to start a browse with a REQID already in use for another browse.
- The method (for example, key or relative record number) used to access a file during a browse is changed by a READNEXT or READPREV command.
- SEGSET or SEGSETALL is specified but the data set does not contain segmented records, or is on a remote system.

Further information is available in the EXEC interface block (refer to "Appendix A. EXEC Interface Block" on page 239 for details).

Default action: terminate the task abnormally.

IOERR
occurs if there is an I/O error during a file control operation. An I/O error is any unusual event that is not covered by a CICS/VS exceptional condition.

Default action: terminate the task abnormally.

ISCINVREQ
occurs when the remote system indicates a failure which does not correspond to a known condition.

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.

NOSPACE
occurs if no space is available on the direct-access device for adding records to a data set.

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. This could occur on a REWRITE command if the RIDFLD data area has changed since the previous READ command. It may occur also on a READPREV command immediately following a STARTBR command which specifies the key of a record that does not exist on the data set.

Default action: terminate the task abnormally.

NOTOPEN
occurs if the requested data set is not open. This condition can occur in response to any file control command except UNLOCK and ENDBR, because a data-base data set can be closed dynamically at any time without regard to outstanding activity on the data set.

Default action: terminate the task abnormally.

SEGIDERR
occurs when the name specified in the SEGSET option is not defined in the FCT.

Default action: terminate the task abnormally.

SYSIDERR
occurs when the SYSID option specifies either a name which is not defined in the intersystem table or a system to which the link is closed.

Default action: terminate the task abnormally.

Chapter 2.3. DL/I Services (DL/I CALL Statement)

DL/I is a general-purpose data base control system that executes in a virtual-storage environment under VSE, OS/VS1, or OS/VS2. It simplifies the creation and maintenance of data bases that can be created by CICS/VS application programs.

For VSE, the DL/I program product (program number 5746-XX1) is used, running as part of the CICS/VS partition. For further information about DL/I, refer to the CICS/VS System/Application Design Guide.

For OS/VS, the IMS/VS program product (program number 5740-XX2) is used, running as part of the CICS/VS region. For further information about IMS/VS, refer to the CICS/VS System/Application Design Guide.

For assembler language, COBOL, and PL/I application programs using the command-level interface, all CICS/OS/VS requests, and CICS/DOS/VS assembler-language requests must be in the form of DL/I CALL statements which are identical to DL/I data base CALL statements running in batch mode or under IMS/VS data communication. (For assembler-language application programs, the CALLDLI macro, rather than the CALL macro, should be used when running under CICS/VS.)

However, for CICS/DOS/VS requests for COBOL and PL/I application programs, the DL/I command-level interface provides a simpler method (by means of the EXEC DLI command) of accessing DL/I data bases.

This chapter describes only the CALL DL/I method of accessing DL/I data bases. The use of the EXEC DLI command for COBOL and PL/I users is described in "Chapter 2.4. DL/I Services (EXEC DLI Command)" on page 77.

The two methods of accessing DL/I data bases cannot both be used in the same task. However, it is possible for different tasks in the same system to use different methods.

The CICS/VS application program can request DL/I services by means of a DL/I CALL statement. In response to such a request, control is passed to a CICS/VS-DL/I routine that acts as an interface between the CICS/VS application program and DL/I. This interface routine checks the validity of the CALL list, sets up DL/I to handle the request, and passes control and the CALL list to DL/I. When the interface routine regains control, it, in its turn, returns control to the calling program, unless a DL/I pseudo-ABEND has occurred, in which case the CICS/VS task is abnormally terminated.

Under CICS/VS, two or more tasks may require access to the same application program at the same time. Because CICS/VS application programs must be quasi-reenterable, DL/I areas that may be modified under CICS/VS, such as PCB pointers, segment search arguments, and I/O work areas, should be placed in dynamic storage. For assembler language this will be in the DFHEISTG DSECT, for COBOL in working storage, and for PL/I in AUTOMATIC storage.

The DL/I data-base access capabilities of a CICS/VS application program are defined in a program specification block (PSB) which is created, by the system programmer, by means of a PSB generation utility program.

The PSB contains one or more program communication blocks (PCBs) that describe the data-base access requirements of each DL/I data base to be accessed by the application program.

A CICS/VS application program designed to access DL/I data bases must schedule its access to DL/I. Scheduling involves, for example, ensuring that the PSB is valid, that the application is not already scheduled, that the referenced data bases are open and enabled, and that there is no intent conflict between the PSB and already scheduled PSBs from other application programs. Negative responses to any of the above will prevent scheduling.

The scheduling call, if successful, returns a list of addresses of the PCBs within the scheduled PSB. The application program in a subsequent CALL statement can specify, from this list, the address of the PCB corresponding to the data base to be accessed. If the addresses cannot be obtained, an INVREQ (invalid request) indicator is returned in response to subsequent DL/I CALL statements in the application program.

A task may schedule only one PSB at a time. Any attempt to schedule a second PSB while one is still scheduled causes the INVREQ indicator to be returned.

A sync point request (see "Chapter 5.6. Recovery (Sync Points)" on page 231) by a task that is scheduled to use DL/I resources implies the release of those resources. This means that if, after issuing a sync point request, access to a

DL/I data base is required, the PSB must be rescheduled. The previous position of the data base has been lost.

To access DL/I data bases, the following steps are required.

1. Issue a DL/I call to schedule the PSB and obtain PCB addresses.

2. Issue a DL/I call to access the required data base.

3. Check the results immediately following each DL/I call.

4. Issue a DL/I call, when all DL/I access is complete, to terminate the connection by releasing the PSB.

USER INTERFACE BLOCK (UIB)

The CICS/VS-DL/I routine that acts as the interface between the CICS/VS application program and DL/I passes information to the application program in a User Interface Block (UIB). A definition of the UIB must be included in the application program. The UIB is acquired by the interface routine when an application program issues a schedule request specifying a pointer reference to be set with the address of the UIB. The UIB contains the address of the PCB address list (UIBPCBAL) from the schedule request and, for each DL/I request, the response (UIBRCODE) from the interface routine, as follows:

Field	ASM	COBOL	PL/I
UIBPCBAL	DS A	PIC 9(8) COMP	POINTER
UIBRCODE	DS 0XL2	PIC XX	
UIBFCTR	DS X	PIC X	BIT(8)
UIBDLTR	DS X	PIC X	BIT(8)

The fields UIBFCTR and UIBDLTR are overlays for the first and second bytes respectively of the return code.

ASM
The UIB definition is included by invoking the DLIUIB macro.

COBOL
The UIB definition is included by a COPY DLIUIB statement in the Linkage Section of the program.

PL/I
The UIB definition is included by a %INCLUDE DLIUIB statement.

Examples of these are given at the end of the chapter.

SCHEDULE THE PSB AND OBTAIN PCB ADDRESSES

The format of the CALL statement to request scheduling of the PSB and to obtain the associated PCB addresses is as follows:

ASM:

```
CALLDLI ASMTDLI,([parmcount,]
        function,psbname,pointer-ref)
```

COBOL:

```
CALL 'CBLTDLI' USING [parmcount,]
     function,psbname,pointer-ref
```

PL/I:

```
CALL PLITDLI ([parmcount,]
     function,psbname,pointer-ref)
```

where:

"parmcount"
is a binary fullword containing a count of the arguments that follow.

"function"
is the name of the field containing the four-character function 'PCBb'.

"psbname"
is an eight-byte field containing the PSB generation name (one through seven characters for VSE, and one through eight for OS/VS) accessed by the application program. It is left justified and padded right with blanks as appropriate. If the PSB name is specified as '*' padded right with blanks, a default name is supplied. For CICS/DOS/VS this default is the first PSB name associated with the application program in the DL/I DOS/VS Application Control Table (ACT) as defined during DL/I DOS/VS system generation. For CICS/OS/VS, this default is the name of the application program associated with this task in the CICS/VS Program Control Table (PCT).

If the call is successful, field UIBPCBAL in the UIB will contain the address of the list of PCB addresses. The order of the addresses is the same as the PCBs within the PSB as specified when the PSB was generated.

If the call is unsuccessful, the reason for the failure will be indicated in field UIBRCODE in the UIB.

"pointer-ref"
is a pointer reference that will be set to the address of the UIB after the call has been processed. The UIB contains the address of the PCB address list and the response from the CICS/VS-DL/I interface.

SEGMENT SEARCH ARGUMENTS (SSAS)

Segment search arguments (SSAs) are used to identify segments of a DL/I data base. SSAs may be simple segment names or they may be qualified to include constraints made upon the values of fields within the named segment types. (For information on how to build an SSA, refer to the publications DL/I DOS/VS Application Programming Reference Manual or IMS/VS Application Programming Reference Manual.)

Except for a read-only operation, when it is unnecessary, SSAs used by a CICS/VS application program must be in dynamic storage because of the requirement for the program to be quasi-reenterable.

- For assembler-language programs, the SSAs should be placed in the dummy section called DFHEISTG.
- For COBOL programs, the SSAs should be in the Working-Storage Section.
- For PL/I programs, the SSAs should be in AUTOMATIC storage.

I/O WORK AREA FOR DL/I SEGMENTS

An I/O work area is required by DL/I to hold the segment being retrieved or to hold the segment being written to the data base. Like SSAs, this work area must be in dynamic storage. The address of the work area is specified as the address of the first byte of the data area.

ISSUE A DL/I DATA BASE CALL

The format of the CALL statement to request DL/I services is as follows:

ASM:

```
CALLDLI ASMTDLI[,([parmcount,]function
        ,pcb,workarea[,ssa1,ssa2,...])]
```

COBOL:

```
CALL 'CBLTDLI' USING [parmcount,]
     function,pcb,workarea[,ssa1,ssa2,...]
```

PL/I:

```
CALL PLITDLI ([parmcount,]function
     ,pcb,workarea[,ssa1,ssa2,...])
```

where:

"parmcount"
is the name of a binary fullword containing a count of the arguments that follow.

"function"
is the 2-4 byte name of the function to be performed. Valid function names for a CICS/VS application program are as follows:

"CHKP"
request that a checkpoint be issued. (VSE only).

"GU"
get a unique segment identified by SSAs.

"GN"
get the next segment in the data base, optionally qualified by SSAs.

"GNP"
get the next segment within the scope of the current hierarchy in the data base, optionally qualified by SSAs.

"GHU"
as for "GU", but in addition, hold the segment for subsequent update.

"GHN"
as for "GN", but in addition, hold the segment for subsequent update.

"GHNP"
as for "GNP", but in addition, hold the segment for subsequent update.

"ISRT"
insert a new segment at the current position; also used in the initial load of a data base.

"REPL"
replace a segment at the current position.

"DLET"
delete the segment at the current position.

"pcb"
is a field containing the address of the PCB corresponding to the data base specified in the call. This address is one of the addresses returned in the address list by the scheduling call.

"workarea"
specifies the workarea that contains the segment being passed to DL/I or is to contain the segment being retrieved from DL/I.

"ssa1,ssa2,..."
are the names of the SSAs.

For details of VSE calls, refer to the DL/I DOS/VS Application Programmer's Reference Manual.

RELEASE A PSB IN THE CICS/VS APPLICATION PROGRAM

When all DL/I operations have been completed, the PSB should be released (or terminated), so that it can be used by other application programs. The releasing application program can reuse the PSB or a different PSB as required.

The DL/I CALL statement is used to release a PSB. It causes all data base records used by the application program, and all associated log records to be written out. It also causes a CICS/VS sync point to be taken, which commits all activity performed by this task, both related to DL/I and to CICS/VS protected resources. (A sync point is taken by means of the SYNCPOINT command, as described in "Chapter 5.6. Recovery (Sync Points)" on page 231.)

Changes performed prior to the execution of the command will not be backed out either in the event of Dynamic Transaction Backout for a single failing task, or in the event of an emergency restart following an abnormal termination of the system. A CICS/VS sync point generates implicitly a DL/I release statement. CALL statements and sync points should be specified only at points in the transaction where logically related processing ends.

The PSB must be rescheduled explicitly after it has been released (by a CALL or sync point) if further access to the data base is required, because the position of the data base has been lost by the release mechanism.

The format of the CALL statement to release a PSB is as follows:

ASM:

```
CALLDLI ASMTDLI,([parmcount,]function)
```

COBOL:

```
CALL 'CBLTDLI' USING [parmcount,]function
```

PL/I:

```
CALL PLITDLI (parmcount,function);
```

where:

"parmcount"
is the name of the binary fullword containing the parameter count value of one.

"function"
is the name of the field containing the four-character function 'TERM' or 'Tbbb'.

CHECK THE RESPONSE TO A DL/I CALL

The response to a DL/I CALL statement should always be checked so that, if unsuccessful, alternative processing can be initiated. Two types of check can be performed, as follows:

- A check that the CICS/VS-DL/I interface has been used correctly by the application program (for example, the required PSB not being found in the directory of PSBs would cause a response code to be returned). The response codes for this type of error appear in the UIB for the task.
- A check that the specified DL/I function has been performed correctly according to the rules of DL/I (for example, a segment that cannot be located from the specified SSA would cause an error indication). This type of error is detected internally by DL/I and is explained in the appropriate DL/I application programming reference manual. DL/I may also issue a pseudo-ABEND which causes the task to be terminated rather than control to be returned to the CICS/VS application program. For CICS/DOS/VS the task is terminated with an ABEND code of "Dnnn", where "nnn" is the DL/I pseudo-ABEND code; for CICS/OS/VS the code is ADLA.

For the first type of check, the response codes are returned in fields UIBFCTR and UIBDLTR in the UIB; these two fields are known collectively as UIBRCODE. Figure 11 on page 73 lists the response codes. These fields should be examined first and, if normal, the DL/I response in the PCB should be examined.

EXAMPLE OF DL/I REQUEST USING CALL

The example at the end of the chapter shows, in the different application programming languages, the use of the DL/I CALL statements to request DL/I services.

Condition	UIBFCTR Response Code		
	ASM	COBOL	PL/I
NORESP (normal response)	X'00'	LOW-VALUES	00000000
NOTOPEN (not open)	X'0C'	12-4-8-9	00001100
INVREQ (invalid request)	X'08'	12-8-9	00001000
Invalid PCB address	X'10'	12-11-1-8-9	00010000
Following codes returned in UIBDLTR after NOTOPEN condition raised			
Data base not open; request issued in OS/VS system	X'00'	12-0-1-8-9	00000000
Data base not open; request issued in VSE system	X'01'	12-1-9	00000001
Intent scheduling conflict	X'02'	12-2-9	00000010
Following codes returned in UIBDLTR after INVREQ condition raised			
Data base not in FCT, or not open according to FCR, or invalid argument passed to DL/I	X'00'	12-0-1-8-9	00000000
PSBNF (PSB not found)	X'01'	12-1-9	00000001
TASKNA (task not authorized)[1]	X'02'	12-2-9	00000010
PSBSCH (PSB already scheduled)	X'03'	12-3-9	00000011
LANGCON (language conflict)[1]	X'04'	12-4-9	00000100
PSBFAIL(PSB initialization failed)	X'05'	12-5-9	00000101
PSBNA (PSB not authorized)[1]	X'06'	12-6-9	00000110
TERMNS (termination unscheduled)	X'07'	12-7-9	00000111
FUNCNS (function unscheduled)	X'08'	12-8-9	00001000
DLINA (DL/I not active)	X'FF'	12-11-0-7-8-9	11111111
[1] CICS/DOS/VS only			

Figure 11. CICS/VS-DL/I Interface Response Codes

ASM

```
DFHEISTG DSECT
UIBPTR   DS    F
IOAREA   DS    0CL40
AREA1    DS    CL3
AREA2    DS    CL37
         DLIUIB
         USING UIB,8
PCBPTRS  DSECT
PCB1PTR  DS    F
PCB1     DSECT
         USING PCB1,6
DBPC1DBD DS    CL8
DBPC1LEV DS    CL2
DBPC1STC DS    CL2
DBPC1PRO DS    CL4
DBPC1RSV DS    F
DBPC1SFD DS    CL8
DBPC1MKL DS    F
DBPC1NSS DS    F
DBPC1KFD DS    0CL256
DBPC1NM  DS    0CL12
DBPC1NMA DS    0CL14
DBPC1NMP DS    CL17
ASMUIB   CSECT
         B     SKIP
PSBNAME  DC    CL8'ASMPSB'
PCBFUN   DC    CL4'PCB'
REPLFUN  DC    CL4'REPL'
TERMFUN  DC    CL4'TERM'
GHUFUN   DC    CL4'GHU'
BLANKS   DC    CL3'   '
SSA1     DC    CL9'AAAA4444'
GOODRC   DC    XL1'00'
GOODSC   DC    CL2'  '
SKIP     DS    0H
         CALLDLI ASMTDLI,(PCBFUN,PSBNAME,UIBPTR)
         L     8,UIBPTR
         CLC   UIBFCTR,X'00'
         BNE   ERROR1
         L     4,UIBPCBAL
         USING PCBPTRS,4
         L     6,PCB1PTR
         CALLDLI ASMTDLI,(GHUFUN,PCB1,IOAREA,SSA1)
         CLC   UIBFCTR,GOODRC
         BNE   ERROR2
         CLC   DBPC1STC,GOODSC
         BNE   ERROR3
         MVC   AREA1,BLANKS
         CALLDLI ASMTDLI,(REPLFUN,PCB1,IOAREA,SSA1)
         CLC   UIBFCTR,GOODRC
         BNE   ERROR4
         CLC   DBPC1STC,GOODSC
         BNE   ERROR5
         B     TERM
ERROR1   DS    0H
*                INSERT ERROR DIAGNOSTIC CODE
ERROR2   DS    0H
*                INSERT ERROR DIAGNOSTIC CODE
ERROR3   DS    0H
*                INSERT ERROR DIAGNOSTIC CODE
ERROR4   DS    0H
*                INSERT ERROR DIAGNOSTIC CODE
ERROR5   DS    0H
*                INSERT ERROR DIAGNOSTIC CODE
TERM     DS    0H
         CALLDLI ASMTDLI,(TERMFUN)
         END   ASMUIB
```

COBOL

```
       IDENTIFICATION DIVISION.
       PROGRAM-ID. 'CBLUIB'.
       ENVIRONMENT DIVISION.
       DATA DIVISION.
       WORKING-STORAGE SECTION.
       77  PSB-NAME PIC X(8) VALUE 'CBLPSB  '.
       77  PCB-FUNCTION PIC X(4) VALUE 'PCB '.
       77  TERM-FUNCTION PIC X(4) VALUE 'TERM'.
       77  GHU-FUNCTION PIC X(4) VALUE 'GHU '.
       77  REPL-FUNCTION PIC X(4) VALUE 'REPL'.
       77  THREE-BLANKS PIC X(3) VALUE '   '.
       77  SSA1 PIC X(9) VALUE 'AAAA4444 '.
       77  SUCCESS-MESSAGE PIC X(40).
       77  GOOD-STATUS-CODE PIC XX VALUE '  '.
       77  GOOD-RETURN-CODE PIC X VALUE LOW-VALUE.
       01  MESSAGE.
           02 MESSAGE1 PIC X(38).
           02 MESSAGE2 PIC XX.
       01  DLI-IO-AREA.
           02 AREA1 PIC X(3).
           02 AREA2 PIC X(37).
       LINKAGE SECTION.
       01  BLLCELLS.
           02 FILLER PIC  S9(8) COMP.
           02 UIB-PTR PIC S9(8) COMP .
           02 B-PCB-PTRS PIC S9(8) COMP.
           02 PCB1-PTR PIC S9(8) COMP.
       01  DLIUIB COPY DLIUIB.
       01  PCB-PTRS.
           02 B-PCB1-PTR PIC 9(8) COMP.
       01  PCB1.
           02 PCB1-DBD-NAME PIC X(8).
           02 PCB1-SEG-LEVEL PIC XX.
           02 PCB1-STATUS-CODE PIC XX.
           02 PCB1-PROC-OPT PIC XXXX.
           02 FILLER PIC S9(5) COMP.
           02 PCB1-SEG-NAME PIC X(8).
           02 PCB1-LEN-KFB PIC S9(5) COMP.
           02 PCB1-NU-SENSEG PIC S9(5) COMP.
           02 PCB1-KEY-FB PIC X(256).
       PROCEDURE DIVISION.
            CALL 'CBLTDLI' USING PCB-FUNCTION, PSB-NAME, UIB-PTR.
            IF UIBFCTR IS NOT EQUAL LOW-VALUES THEN
      *       INSERT ERROR DIAGNOSTIC CODE
              EXEC CICS RETURN END-EXEC.
            MOVE UIBPCBAL TO B-PCB-PTRS.
            MOVE B-PCB1-PTR TO PCB1-PTR.
            CALL 'CBLTDLI' USING GHU-FUNCTION, PCB1, DLI-IO-AREA, SSA1.
            SERVICE RELOAD UIB-PTR
            IF UIBFCTR IS NOT EQUAL GOOD-RETURN-CODE THEN
      *       INSERT ERROR DIAGNOSTIC CODE
              EXEC CICS RETURN END-EXEC.
            IF PCB1-STATUS-CODE IS NOT EQUAL GOOD-STATUS-CODE THEN
      *       INSERT ERROR DIAGNOSTIC CODE
              EXEC CICS RETURN END-EXEC.
            MOVE THREE-BLANKS TO AREA1.
            CALL 'CBLTDLI' USING REPL-FUNCTION, PCB1, DLI-IO-AREA, SSA1.
            IF UIBFCTR IS NOT EQUAL GOOD-RETURN-CODE THEN
      *       INSERT ERROR DIAGNOSTIC CODE
              EXEC CICS RETURN END-EXEC.
            IF PCB1-STATUS-CODE IS NOT EQUAL GOOD-STATUS-CODE THEN
      *       INSERT ERROR DIAGNOSTIC CODE
              EXEC CICS RETURN END-EXEC.
            CALL 'CBLTDLI' USING TERM-FUNCTION.
             EXEC CICS RETURN END-EXEC.
```

PL/I

```
PLIUIB: PROC OPTIONS(MAIN);
 DCL PSB_NAME CHAR(8) STATIC INIT('PLIPSB  ');
 DCL PCB_FUNCTION CHAR(4) STATIC INIT('PCB ');
 DCL TERM_FUNCTION CHAR(4) STATIC INIT('TERM');
 DCL GHU_FUNCTION CHAR(4) STATIC INIT('GHU ');
 DCL REPL_FUNCTION CHAR(4) STATIC INIT('REPL');
 DCL THREE_BLANKS CHAR(3) STATIC INIT('   ');
 DCL SSA1 CHAR(9) STATIC INIT('AAAA4444 ');
 DCL PARM_CT_1 FIXED BIN(31) STATIC INIT(1);
 DCL PARM_CT_3 FIXED BIN(31) STATIC INIT(3);
 DCL PARM_CT_4 FIXED BIN(31) STATIC INIT(4);
 DCL GOOD_RETURN_CODE BIT(8) STATIC INIT('0'B);
 DCL GOOD_STATUS_CODE CHAR(2) STATIC INIT('  ');
 DCL IO_AREA_PTR POINTER;
 %INCLUDE DLIUIB;
 DCL 1 PCB_POINTERS BASED(UIBPCBAL),
       2 PCB1_PTR POINTER;
 DCL 1 DLI_IO_AREA,
       2 AREA1 CHAR(3),
       2 AREA2 CHAR(37);
 DCL 1 PCB1 BASED(PCB1_PTR),
       2 PCB1_DBD_NAME CHAR(8),
       2 PCB1_SEG_LEVEL CHAR(2),
       2 PCB1_STATUS_CODE CHAR(2),
       2 PCB1_PROC_OPTIONS CHAR(4),
       2 PCB1_RESERVE_DLI FIXED BIN (31,0),
       2 PCB1_SEGNAME_FB CHAR(8),
       2 PCB1_LENGTH_FB_KEY FIXED BIN(31,0),
       2 PCB1_NUMB_SENS_SEGS FIXED BIN(31,0),
       2 PCB1_KEY_FB_AREA CHAR(17);
   CALL PLITDLI(PARM_CT_3,PCB_FUNCTION,PSB_NAME,UIBPTR);
   IF UIBFCTR¬=GOOD_RETURN_CODE THEN DO;
     /*  INSERT ERROR DIAGNOSTIC CODE  */
   END;
   CALL PLITDLI(PARM_CT_4,GHU_FUNCTION,PCB1,DLI_IO_AREA,SSA1);
   IF UIBFCTR¬=GOOD_RETURN_CODE THEN DO;
     /*  INSERT ERROR DIAGNOSTIC CODE  */
   END;
   IF PCB1_STATUS_CODE¬=GOOD_STATUS_CODE THEN DO;
     /*  INSERT ERROR DIAGNOSTIC CODE  */
   END;
   AREA1=THREE_BLANKS;
   CALL PLITDLI(PARM_CT_4,REPL_FUNCTION,PCB1,DLI_IO_AREA,SSA1);
   IF UIBFCTR¬=GOOD_RETURN_CODE THEN DO;
     /*  INSERT ERROR DIAGNOSTIC CODE  */
   END;
   IF PCB1_STATUS_CODE¬=GOOD_STATUS_CODE THEN DO;
     /*  INSERT ERROR DIAGNOSTIC CODE  */
   END;
   CALL PLITDLI(PARM_CT_1,TERM_FUNCTION);
   END PLIUIB;
```

Chapter 2.4. DL/I Services (EXEC DLI Command)

This chapter outlines the EXEC DLI command that can be used in CICS/DOS/VS command-level application programs that are used to access DL/I data bases under VSE. These programs, which can be written only in COBOL or PL/I, require the installation of the DL/I DOS/VS program product (program number 5746-XX1), which runs as part of the CICS/VS partition in the VSE system.

These commands have a syntax and format that are similar to CICS/VS commands (EXEC DLI instead of EXEC CICS). Full details of the commands are given in the publication DL/I DOS/VS Application Programmer's Reference Manual.

The commands are translated by the appropriate command language translator (see "Chapter 1.2. Command Format and Argument Values" on page 5) into calls to the CICS/VS link-edit stub. At execution, DFHEIP is invoked which in turn invokes a DL/I interface program to perform the requested operations.

There are no exceptional conditions for DL/I commands, though the HANDLE ABEND command can be used if desired to handle abends issued by DL/I.

GENERAL FORMAT OF EXEC DLI COMMAND

The general format of the EXEC DLI command is as follows:

```
EXECUTE|EXEC) DLI function
[option[(argument)]]...
```

The functions, options, and arguments that can be used are as follows:

CHECKPOINT|CHKP Request a checkpoint
 ID(char-expr)

DELETE|DLET Delete a segment
 [USING PCB(integer-expr)]
 [VARIABLE]
  SEGMENT(name)
  FROM(data-area)
 [SEGLENGTH(integer-expr)]

GET UNIQUE|GU or **GET NEXT|GN** or
GET NEXT IN PARENT|GNP Get a segment
 [USING PCB(integer-expr)]
 [VARIABLE]
 [FIRST|LAST]
 [SEGMENT(name)]
 [LOCKED]
  INTO(data-area)
 [SEGLENGTH(integer-expr)]
 [WHERE(name operator data area)]
 [FIELDLENGTH(integer-expr)]
 [OFFSET(integer-expr)]

INSERT|ISRT Insert a segment
 [USING PCB(integer-expr)]
 [VARIABLE]
 [FIRST|LAST]
  SEGMENT(name)
 [SEGLENGTH(integer-expr)]
  FROM(data-area)
 [WHERE(name operator data area)]
 [FIELDLENGTH(integer-expr)]

REPLACE|REPL Replace a segment
 [USING PCB(integer-expr)
 [VARIABLE]
  SEGMENT(name)
 [SEGLENGTH(integer-expr)]
  FROM(data-area)

SCHEDULE|SCHD Schedule the PSB
 [PSB(name)]

TERMINATE|TERM Terminate access

SEGLENGTH is required in COBOL whenever FROM or INTO is specified. It is never required in PL/I.

On the GET, INSERT, and REPLACE commands, the segment-oriented keywords (that is, all those except USING PCB) may be repeated for each segment. Keywords preceding the keyword SEGMENT in the above list must be written immediately preceding the segment to which they apply, but within themselves may be written in any order. Similarly, keywords which follow the keyword SEGMENT in the above list must be written immediately following the segment to which they apply, but within themselves they may be written in any order.

The command must be delimited, in the same way as an EXEC CICS command, by END-EXEC for COBOL and by a semicolon for PL/I, for example:

```
EXEC DLI GET SEGMENT(SKILL)
         WHERE(SKILLTYPE='PLUMBER')
         INTO(SKILLSTRUCT) END-EXEC
```

DL/I INTERFACE BLOCK (DIB)

The CICS/VS-DL/I interface module passes information to the CICS/VS application program in a DL/I Interface Block (DIB). The DIB contains the response from the interface module in the field DIBSTAT. The DIB structure is included automatically in the application program by the translator, and unlike the EIB, no copy book exists in the source statement

library. The fields and their descriptions are as follows:

Field	COBOL	PL/I
DIBFLAG	PIC X	CHAR(1)
DIBSEGLV	PIC XX	CHAR(2)
DIBSEGM	PIC X(8)	CHAR(8)
DIBSTAT	PIC XX	CHAR(2)

Field DIBFLAG is a flag indicating that an online task had to wait for a resource owned by an MPS batch task. The value is either X'FF' (HIGH-VALUE in COBOL, HIGH(1) in PL/I) or X'00' (LOW-VALUE in COBOL, LOW(1) in PL/I).

Field DIBSEGLV gives the hierarchical level of the object segment or lowest level parent segment actually retrieved.

Field DIBSEGM is the name of the object segment or the lowest level parent segment actually retrieved.

Field DIBSTAT is the DL/I status code.

EXAMPLE OF DL/I REQUEST USING EXEC DLI

The following example (in COBOL and PL/I) shows the use of the EXEC DLI command to request DL/I services; it provides the same functions as the example in the previous chapter.

COBOL

```
CBL XOPTS(DLI,CICS)
      IDENTIFICATION DIVISION.
      PROGRAM-ID. EXAMPL.
      ENVIRONMENT DIVISION.
      DATA DIVISION.
      WORKING-STORAGE SECTION.
      01  SEGDATA.
        02  AREA1 PICTURE X(3).
        02  AREA2 PICTURE X(37).
      01 SEGDATAL COMPUTATIONAL PICTURE S9999 VALUE IS +40
      PROCEDURE DIVISION.
          EXEC DLI SCHEDULE PSB(CBLPSB) END-EXEC
          IF DIBSTAT IS NOT EQUAL SPACES THEN
      *     INSERT ERROR CODE
          EXEC DLI GET UNIQUE SEGMENT(AAAA4444)
          INTO(SEGDATA) SEGLENGTH(SEGDATAL) END-EXEC
          IF DIBSTAT IS NOT EQUAL SPACES THEN
      *     INSERT ERROR CODE
          MOVE SPACES TO AREA1.
          EXEC DLI REPLACE SEGMENT(AAAA4444)
          FROM(SEGDATA) SEGLENGTH(SEGDATAL) END-EXEC
          IF DIBSTAT IS NOT EQUAL SPACES THEN
      *     INSERT ERROR CODE
          EXEC DLI TERMINATE END-EXEC
          EXEC CICS RETURN END-EXEC
          GOBACK.
```

PL/I

```
* PROCESS XOPTS(DLI,CICS),INCLUDE;
EXAMPLE: PROC OPTIONS(MAIN);
DCL 1 SEG_DATA,
       2 AREA1 CHAR(3),
       2 AREA2 CHAR(37);
EXEC DLI SCHEDULE PSB(PLIPSB);
IF DIBSTAT ¬= ' ' THEN CALL ERROR;
EXEC DLI GET UNIQUE SEGMENT(AAAA4444) INTO(SEG_DATA);
IF DIBSTAT ¬= ' ' THEN CALL ERROR;
AREA1 = 'XXX';
EXEC DLI REPLACE SEGMENT(AAAA4444) FROM (SEG_DATA);
IF DIBSTAT ¬=' ' THEN CALL ERROR;
EXEC DLI TERMINATE;
ERROR: PROC;
 /* INSERT USER ERROR ROUTINE */
END;
END;  /* EXAMPLE */
```

Part 3. Data Communication Operations

Chapter 3.1. Introduction to Data Communication Operations

Three methods are available to the CICS/VS application programmer for communicating with the terminals and logical units in the subsystems of the network that forms part of the CICS/VS system. The methods dealt with are:

- Terminal control
- Basic mapping support (BMS)
- Batch data interchange

Terminal control is the basic method for communicating with devices, whereas both BMS and batch data interchange extend the facilities of terminal control to simplify further the manipulation of data streams. Both BMS and batch data interchange use terminal control facilities when invoked by an application program. Terminal control provides commands and options that can be specified in various combinations according to the requirements of the devices. However, application programs written in this way are dependent on the data formatting requirements of these devices and a detailed knowledge of the devices is required. Terminal control is described in "Chapter 3.2. Terminal Control" on page 85.

Basic mapping support provides commands and options that can be used to format data in a standard manner. BMS converts data streams provided by the application program to conform to the requirements of the devices. Conversely, data received from a device is converted by BMS to a standard form. However, not all devices supported by CICS/VS can be used with BMS and, for those that cannot, terminal control must be used. Also, in some cases, the overhead incurred to achieve data stream independence may outweigh the advantages. The choice as to whether BMS should be used is a matter for application design and is discussed more fully in the CICS/VS System/Application Design Guide. BMS is described in "Chapter 3.3. Basic Mapping Support (BMS)" on page 125.

Batch data interchange provides commands and options that may be used, possibly in conjunction with BMS commands, to communicate with the 6670 logical unit and with the batch logical units of the 3770 and 3790 subsystems. Batch data interchange is described in "Chapter 3.4. Batch Data Interchange" on page 169.

Chapter 3.2. Terminal Control

The CICS/VS terminal control program provides for communication between user-written application programs and terminals and logical units, by means of terminal control commands.

Terminal control uses the standard access methods available with the host operating system. The Basic Telecommunications Access Method (BTAM) is used by CICS/VS for most start-stop and BSC terminals. As an option for OS/VS, the Telecommunications Access Method (TCAM) can be specified. The Sequential Access Method (SAM) is used where keyboard terminals are simulated by sequential devices such as card readers and line printers. The Virtual Telecommunications Access Method (ACF/VTAM) or the Telecommunications Access Method (TCAM) is used for Systems Network Architecture (SNA) terminal systems.

Terminal control polls terminals to see if they are ready to transmit or receive data. Terminal control handles code translation, transaction validation, synchronization of input and output operations, and the line control necessary to read from or write to a terminal. The application program is freed from having to physically control terminals. During processing, an application program is connected to one terminal for one task and the terminal control program monitors which task is associated with which terminal. The task to be initiated is determined as described later in this chapter under "Terminal-Oriented Task Identification".

Terminal control is used for communication with terminals. In SNA systems, however, it is used also to control communication with logical units or with another CICS/VS system. A logical unit (LU) represents either a terminal directly, or a program stored in a subsystem controller which in turn controls one or more terminals. The CICS/VS application program communicates, by means of the logical unit, either with a terminal or with the stored program. For example, a 3767 terminal is represented by a single logical unit without an associated user-written application program. In contrast, a 3790 subsystem is represented by a 3791 controller, user-written 3790 application programs, and one or more 3790 terminals; when the subsystem is configured, one or more logical units are designated by the user.

Terminal control is used also for communicating with terminals or logical units in a remote system by means of Distributed Transaction Processing (DTP). SNA protocols are available, through terminal-control commands, to initiate and terminate a conversation (a session) with a remote LU6 logical unit.

This conversation is carried on between a principal facility and one or more alternate facilities.

A **principal facility** for a task is a terminal or LU6 session that is made available to the application program when the task is initiated.

An **alternate facility** for a task is a terminal or LU6 session acquired as needed by the application program. In general, terminal-control commands that refer to an alternate facility should include the SESSION option.

The ALLOCATE and FREE commands allow the application program to acquire and release these alternate facilities and allow both principal and alternate facilities to be used at the same time.

The BUILD ATTACH and EXTRACT ATTACH commands, together with the ATTACHID option of the SEND command, allow the application program to attach a transaction in a remote system.

Fields in the EIB allow access to indicators that give the status of the conversation after execution of RECEIVE or CONVERSE commands. For example, EIBEOC, EIBATT, and EIBFMH provide more information about the received data, and EIBSYNC, EIBFREE, and EIBRECV provide more information about the session.

The INVITE option of the SEND command allows the optimization of SNA flows that occur when communicating with another transaction, or with IMS/VS.

Distributed transaction processing is described fully in the CICS/VS System/Application Design Guide.

Commands and options that apply specifically to logical units are described later in the chapter.

Terminal control commands are provided to request the following services that are applicable to most, or all, of the types of terminal or logical unit supported by CICS/VS:

- Read data from a terminal or logical unit (RECEIVE).

- Write data to a terminal or logical unit (SEND).
- Converse with a terminal or logical unit (CONVERSE).
- Synchronize terminal input/output for a transaction (WAIT TERMINAL).
- Send an asynchronous interrupt (ISSUE SIGNAL).
- Relinquish use of a communication line (ISSUE RESET).
- Disconnect a switched line or terminate a session with a logical unit (ISSUE DISCONNECT).

It is possible to read records from a card reader and read records from or write records to a disk data set, magnetic tape unit, or a line printer defined by the system programmer as a card-reader-in/line-printer-out (CRLP) terminal. For additional information, see the section "Sequential Terminal Support" in "Chapter 5.1. Introduction to Recovery and Debugging" on page 213.

Other services available in response to terminal control commands apply to specific types of terminal. The permissible commands and options that can be used by specific terminal types are detailed later in this chapter. Because many types of terminal are supported by CICS/VS, many special services are provided. (For a list of terminals supported by CICS/VS, see the publication CICS/VS General Information.) In particular, a large number of commands are provided for communicating with display devices such as the 3270 Information Display System; these are described in the section "Display Device Operations" later in this chapter.

The options that follow the command depend on the terminal or logical unit (and sometimes, access method) used and the operations required. Options included in a terminal control command that do not apply to the device being used will be ignored.

The HANDLE CONDITION and IGNORE CONDITION commands, and the NOHANDLE option, can be used to deal with exceptional conditions that occur during the execution of terminal control commands. Refer to "Chapter 5.1. Introduction to Recovery and Debugging" on page 213 for further information about exceptional conditions.

COMMANDS AND OPTIONS FOR TERMINALS AND LOGICAL UNITS

The commands and options described in this section apply to all terminals and logical units. There may, however, be others that apply to specific devices. If so, details are given later in the chapter under headings for the device types.

READ FROM TERMINAL OR LOGICAL UNIT (RECEIVE)

The RECEIVE command is used to read data from a terminal or logical unit. The INTO option is used to specify the area into which the data is to be placed, in which case the maximum length of data that the program will accept must be specified in the LENGTH option. If the data exceeds the specified maximum, it is truncated and the LENGERR condition occurs. If the LENGTH option is specified, the named data area is set to the actual data length (before truncation occurs) when data has been received.

Alternatively, a pointer reference can be specified in the SET option. CICS/VS acquires an area large enough to hold the data and sets the pointer reference to the address of that area. When data has been received, the data area specified in the LENGTH option is set to the data length.

The first RECEIVE command in a terminal-initiated task will not issue a terminal-control read but will simply copy the input buffer, even if the data length is zero. A second RECEIVE must be issued to cause a terminal-control read.

WRITE TO TERMINAL OR LOGICAL UNIT (SEND)

The SEND command is used to write data to a terminal or logical unit. The options FROM and LENGTH specify respectively the data area from which the data is to be taken and the length (in bytes) of the data.

The WAIT Option of the SEND Command

Unless the WAIT option is specified also, the transmission of the data associated with the SEND command is deferred until a later event, such as a sync point, occurs. This deferred transmission reduces the flows of data by allowing data flow controls to be transmitted with the data.

SYNCHRONIZE TERMINAL INPUT/OUTPUT FOR A TRANSACTION (WAIT TERMINAL)

This command is used to ensure that a terminal operation has completed before further processing occurs in a task under which more than one terminal or logical unit operation is performed. Alternatively, the WAIT option can be specified in a SEND command. (A wait is

always carried out for a RECEIVE command.)

Either method may cause execution of a task to be suspended. If suspension is necessary, control is returned to CICS/VS. Execution of the task is resumed when the operation is completed.

Even if the WAIT option is not specified in a SEND command, the EXEC interface program will ensure that the operation is completed before issuing a subsequent RECEIVE or SEND command.

CONVERSE WITH TERMINAL OR LOGICAL UNIT (CONVERSE)

For most terminals or logical unit types a conversational mode of communication is permissible. The CONVERSE command is used for this purpose. In general, the CONVERSE command can be considered as a combination of a SEND command followed immediately by a WAIT TERMINAL command and then by a RECEIVE command. However, not all options of the SEND and RECEIVE commands are valid for the CONVERSE command. Specific rules are given in the syntax descriptions for different devices later in this chapter. The TOLENGTH option is equivalent to the LENGTH option of the RECEIVE command, and the FROMLENGTH option is equivalent to the LENGTH option of the SEND command.

SEND AN ASYNCHRONOUS INTERRUPT (ISSUE SIGNAL)

This command is used, in a transaction in receive mode, to inform the sending transaction that it wishes to change modes. The execution of the command will raise the SIGNAL condition on the next SEND or RECEIVE command executed in the sending transaction, and a previously executed HANDLE CONDITION command for this condition can be used either to action the request or to ignore it.

RELINQUISH A COMMUNICATION LINE (ISSUE RESET)

This command is used to relinquish use of a communication line. The command applies only to binary synchronous devices using BTAM. The next BTAM operation will be a read or write initial.

DISCONNECT A SWITCHED LINE (ISSUE DISCONNECT)

This command is used to break a line connection between a terminal and the processor, or to break a session between TCAM or ACF/VTAM logical units, when the transaction is completed. If the terminal is a buffered device, the data in the buffers will be lost.

TERMINAL-ORIENTED TASK IDENTIFICATION

When CICS/VS receives input from a terminal to which no task is attached, it has to determine which transaction should be initiated. The methods by which the user can specify the transaction to be initiated and the sequence in which CICS/VS checks these specifications are as follows (see also Figure 12 on page 89). The system macros referred to in the following tests are described in the CICS/VS System Programmer's Reference Manual.

Test 1:
Is the input from a PA key (of a 3270 terminal) that has been defined at system initialization as the print request key?

If yes, printing of the data displayed on the screen is initiated.

Test 2:
(a) Is this terminal of a type supported by BMS terminal paging?

(b) Is the input a paging command? (The terminal operator can enter paging commands defined in the DFHSIT system macro.)

If yes to both (a) and (b), the transaction CSPG, which processes paging commands, is initiated.

Test 3:
If an attach FMH is present in the data stream and Tests 4 and 5 are not fulfilled, the transaction specified in the attach FMH is initiated. The architectured attach names are converted to "CSMI".

Test 4:
Does the terminal control table entry for the terminal include a transaction identification (specified by the TRANSID operand of the DFHTCT TYPE=TERMINAL system macro.)

If yes, the specified transaction is initiated.

Test 5:
Is a transaction specified by the TRANSID option of a program control RETURN command (or by the application program moving the transaction name into TCANXTID)?

If yes, the specified transaction is initiated.

Test 6:

(a) Is the terminal a 3270 (including 3270 logical unit and 3650 host-conversational (3270) logical unit, connected via VTAM)?

(b) Is the input from a PA key, PF key, light pen attention, or operator identification card reader?

(c) Is this input specified by the TASKREQ operand of the DFHPCT TYPE=ENTRY system macro?

If yes to (a), (b), and (c), the program specified by the PROGRAM operand of the same DFHPCT TYPE=ENTRY macro is given control.

Test 7:

Is a valid transaction identification specified by the first one to four characters of the terminal input?

If yes, the specified transaction is initiated.

For all PA keys and some LPAs there cannot be terminal input. If there is no terminal input an "invalid transaction identification" message is sent to the terminal.

If none of the above tests is met, an invalid transaction identification exists, and message DFH2001 (INVALID TRANSACTION IDENTIFICATION - PLEASE RESUBMIT) is sent to the terminal.

The 3735 Programmable Buffered Terminal makes an exception to this sequence when operating in inquiry mode. The test of input from the terminal (Test 7 above) is given highest priority.

COMMANDS AND OPTIONS FOR LOGICAL UNITS

An application program communicates with a TCAM or VTAM logical unit in much the same way as it does with BTAM or TCAM terminals (that is, by using the terminal control commands described above). However, communication with logical units is governed by the conventions (protocols) that apply to each type of logical unit. This section describes the additional commands and options provided by CICS/VS to enable application programs to comply with these protocols.

The types of logical units and the related protocols for each of the SNA subsystems supported by CICS/VS are described in the CICS/VS guides for the subsystems. (See the Bibliography).

SEND/RECEIVE MODE

For SNA logical units, only one of the two ends of the session can be in send mode at any one time, that is, one is in send mode, the other is in receive mode. An application program in send mode can issue any commands for the logical unit. On the other hand, one in receive mode, can issue only RECEIVE commands until the mode is changed back to send. The EIB indicator EIBRECV informs the application program that it is in receive mode and that it must perform the above operations.

If the above protocols are not followed, the transaction will be abended, unless the read-ahead queueing feature (RAQ=YES) is specified in the DFHPCT TYPE=ENTRY system macro. This option allows the application program to ignore the EIBRECV indicator and to send and receive at any time. However, it should only be used with transactions that support both bisynchronous devices and logical units.

For displays, the transaction would normally be in send mode, provided that the INVITE option is not used, and can ignore the EIBRECV indicator. Displays work with a subset of the full protocols (see the CICS/VS System/Application Design Guide for further information).

SEND/RECEIVE PROTOCOL (INVITE OPTION)

The INVITE option of a SEND command informs the session partner that it is now in send mode and that it should send a reply. At the same time it places the transaction in receive mode. The transaction should now issue a RECEIVE command as its next operation.

CHAINING OF INPUT DATA

The unit of data from a logical unit is the request/response unit (RU). One or more RUs can be grouped together and treated as a chain.

The last RU in a chain (even if it is the only RU in the chain) raises an end-of-chain (EOC) condition. When this occurs, a HANDLE CONDITION EOC command will give control to a user-written routine, which can do any additional processing required when the complete chain has been received.

For logical units that do not send chained data (for example, the 3270 logical unit), the EOC condition occurs for every RECEIVE request. For logical units that send chained data, the EOC condition usually occurs for every RECEIVE request, but it may not, depending on the length of the data and on whether the terminal control table

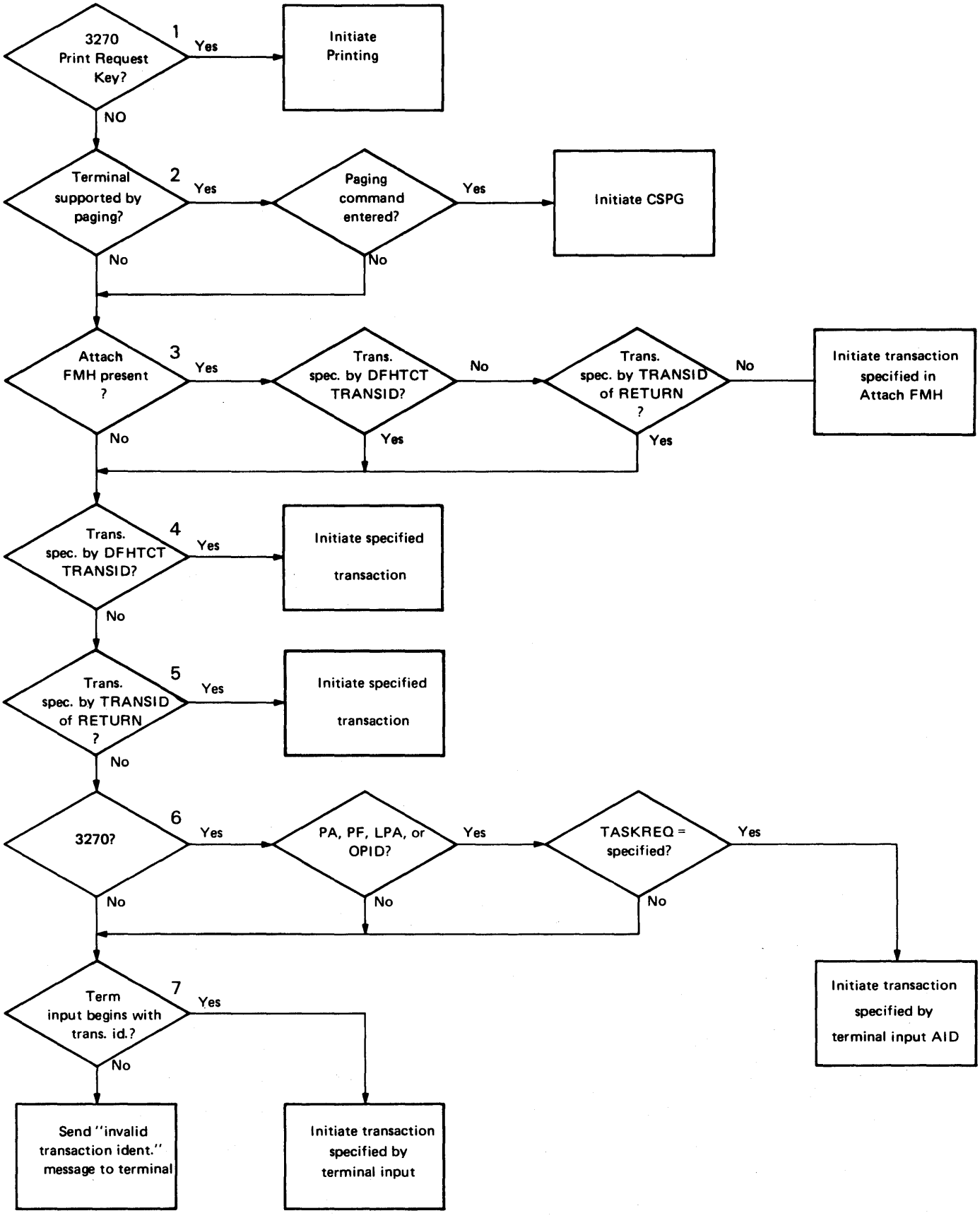


Figure 12. Terminal-Oriented Task Identification

CHNASSY option is specified by the system programmer. The syntax descriptions for individual logical units in this chapter omit the EOC condition unless it is likely that meaningful use may be made of the fact that it has not been received. The IGNORE CONDITION command can be used to ignore the EOC condition in cases where it is raised on every RECEIVE command.

The EOC condition may occur simultaneously with the EODS (end-of-data-set) and/or INBFMH (inbound-FMH) conditions. When this happens, the user-written routine for the EODS or INBFMH conditions will be given control rather than the EOC routine.

The system programmer specifies, in the TCTTE, whether or not chaining is to occur. If chain assembly is specified, instead of an input request being satisfied by one RU at a time until the chain is complete, the whole chain is assembled and is sent to the CICS/VS application program satisfying just one request. This ensures that the integrity of the whole chain is known before it is presented to the application program.

CHAINING OF OUTPUT DATA

As in the case of input data, output data is transmitted as request/response units (RUs). If the length of the data to be sent exceeds the RU size, CICS/VS automatically breaks up the data into RUs and transmits these RUs as a chain. During transmission from CICS/VS to the logical unit, the RUs are marked FOC (first-of-chain), MOC (middle-of-chain), or EOC (end-of-chain) to denote their position in the chain. An RU that is the only one in a chain is marked OC (only-in-chain).

If the system programmer specified that the application program can control the chaining of outbound data, the application program uses the CNOTCOMPL (chain-not-complete) option of the SEND command to indicate continuation of the chain. In general, the CNOTCOMPL option should not be used. Once an output request with CNOTCOMPL specified has been made, subsequent output requests may not use the FMH, LAST, or (for the 3600 (3601) logical unit) LDC options until the beginning of the next chain (that is, the first output request following an output request in which CNOTCOMPL is omitted).

LOGICAL RECORD PRESENTATION

Each RECEIVE command results in one RU (or one chain of RUs if chain assembly is specified) being presented to the application program. An RU may consist of one or more logical records. If an RU contains more than one logical record, the records will be separated by new line(NL), inter-record separator(IRS), or transparent(TRN) characters. Except for LUTYPE4 devices, a logical record cannot be transmitted in more than one RU; the end of the RU is always the end of the logical record. Data from an LUTYPE4 may contain logical records that span RUs, in which case, chain assembly should be specified.

The system programmer can specify in the PCT, for specific application programs, that the application program will be presented with logical records instead of with RUs or chains. For those application programs for which this option is specified, each RECEIVE command results in one logical record being presented to the application program, regardless of whether chain assembly is specified or not.

If the logical records are separated by IRS or TRN characters, these are removed, and do not appear in the data. Therefore, a blank card will have a length of zero. If NL characters are used to separate the logical records, they are not removed, and the NL character from the end of each logical record appears at the end of the data. If the delimiter is a transparent (TRN) character, CICS/VS will pass up to 256 bytes in one logical record. This logical record can contain any characters, including NL and IRS characters, all of which will be treated as data .

All communication features for logical units are still in operation, that is, notification of end-of-chain conditions, and (for batch logical units only) notification of end-of-data-set conditions and presentation of the inbound FMH at the beginning of a chain, still occurs.

If chain assembly has been specified, a logical record ends with a delimiter (NL, IRS, or TRN), or the end of the assembled chain. The end of chain notification occurs in the last logical record of the chain.

DEFINITE RESPONSE

The type of response requested by CICS/VS for outbound data is generally determined by the system programmer in the PCT; it can be specified that all outbound data for an application program will require a definite response, or that exception-response protocol is to be used, that is, a response will be made only if an error occurs.

The use of definite-response protocol has some performance disadvantages, but may be necessary for some application

programs. To provide a more flexible method of specifying the protocol to be used, the DEFRESP option is provided for use on the SEND command. One example of the use of this option is to request a definite response for every tenth output command, exception response being the general rule.

Because a definite response can be requested only on the last element in the chain, the DEFRESP and CNOTCOMPL options are mutually exclusive.

FUNCTION MANAGEMENT HEADER (FMH)

A function management header (FMH) is a field that can be included at the beginning of an input or output message. It is used to convey information about the message and how it should be handled. For some logical units, the use of an FMH is mandatory, for others it is optional, and in some cases FMHs cannot be used at all.

For output, the FMH can be built by the application program or by CICS/VS. For input, the FMH can be passed to the application program or it can be suppressed by CICS/VS.

The FMH option of the SEND command is used to specify that the application program will provide the FMH in the data to be transmitted.

The ATTACHID option specifies a set of values that CICS/VS puts into an LU6 attach FMH which is concatenated ahead of the user data.

Further information about FMHs is given in the CICS/VS guides for the subsystems. (See the Bibliography.)

Inbound FMH

An application program can request notification when an FMH is included in the data received from a batch logical unit.

Whether or not inbound FMHs will be passed to the application program is specified in the INBFMH operand of the DFHPCT TYPE=ENTRY system macro. It can be specified that no inbound FMHs will be passed, or that only the FMH at the end of the data set will be passed, or that all inbound FMHs will be passed.

If inbound FMHs are to be passed to the application program, a HANDLE CONDITION INBFMH command will allow control to be passed to a user-written routine whenever an inbound FMH is received. These user-written routines can investigate the contents of the FMH and take some action depending on, for example, the device from which the data has come. The contents of the FMH can be accessed also by means of the EIBFMH field of the EIB.

If an inbound FMH, containing an attach FMH, is passed to the application program, the attach FMH can be removed as long as this has been allowed for by the system programmer in the PCT. The values of the attach FMH may be examined by using the EXTRACT ATTACH command.

When input data is received as a chain of RUs, only the first (or only) RU of the chain is preceded by an FMH.

Outbound FMH

If the user data contains one or more FMHs, the output request must specify the FMH option. When sending output data to a logical unit that expects an FMH, the FMH must be at the start of the user data to be transmitted.

UNSOLICITED INPUT

If unsolicited input arrives from a logical unit, it is queued and used to satisfy future input requests for that logical unit. However, for 3270 logical units, unsolicited input will be discarded if the PUNSOL operand is specified in the DFHSG PROGRAM=TCP system macro.

BRACKET PROTOCOL (LAST OPTION)

Bracket protocol prevents the interruption of a transaction between CICS/VS and a logical unit. A bracket can, generally, be begun either by CICS/VS or by the logical unit, or ended only by CICS/VS unless it is for an LU6 logical unit, in which case the logical unit can end it. A bracket also can delimit conversation between CICS/VS and the logical unit or merely the transmission of a series of data chains in one direction.

Bracket protocol is used when CICS/VS communicates with some logical units. The use of brackets is usually transparent to the application program.

Only on the last output request of a task to a logical unit does the bracket protocol become apparent to the application program. On the last output request to a logical unit, the application program may specify the LAST option on the SEND command. The last output request is defined as either the last SEND command specified for a task without chain control; or as the output request that transmits the FOC or OC marker of the last chain of a transaction with chain control. The LAST option causes CICS/VS to transmit an end-bracket indicator with the final output message

to the logical unit. This indicator notifies the logical unit that the current transaction is ending. If the LAST option is not specified, CICS/VS waits until the task detaches before sending the end-bracket indicator. Since an end-bracket indicator is transmitted only with the first RU of a chain, the LAST option is ignored for a transaction with chain control unless FOC or OC is also specified.

Including a FREE command after a SEND command with the LAST option may be useful if the transaction does not terminate immediately after issuing the SEND command. This allows another transaction to be initiated from the LU or from CICS/VS.

SUSPEND A TASK (WAIT SIGNAL)

```
WAIT SIGNAL

Condition:  SIGNAL
```

This command is used, for a principal facility only, to suspend a task until a SIGNAL condition occurs. Some logical units can interrupt the normal flow of data to the application program by a SIGNAL data-flow-control command to CICS/VS, signaling an attention, which in turn causes the SIGNAL condition to occur.

The HANDLE CONDITION SIGNAL command will cause a branch to an appropriate user-written routine when an attention is received.

TERMINATE A SESSION (ISSUE DISCONNECT)

```
ISSUE DISCONNECT
```

This command is used to terminate a session between CICS/VS and a logical unit, but only if the system programmer has specified RELREQ=(,YES) in the DFHTCT TYPE=TERMINAL macro for the logical unit.

RETURN A FACILITY TO CICS/VS (FREE)

```
FREE [SESSION(name)]

Conditions:
INVREQ, NOTALLOC, SESSIONERR
```

This command is used to return a facility (a principal facility or a previously allocated alternate facility) to CICS/VS when a transaction owning it no longer requires it. The facility then can be allocated for use by other transactions.

Facilities not freed explicitly will be freed by CICS/VS when the task terminates.

TCAM-SUPPORTED TERMINALS AND LOGICAL UNITS (CICS/OS/VS ONLY)

Because TCAM permits many applications to share a single network, the CICS/VS-TCAM interface supports data streams rather than specific terminals or logical units.

Operations for terminals supported by TCAM use the same options as the terminals supported by other access methods. With the exception of the BUFFER option for the 3270, all options applicable for input operations are supported by CICS/VS-TCAM. However, the exceptional conditions ENDINPT and EOF will not occur.

All output requests are the same for TCAM as for other CICS/VS supported terminals, except that:

- the ISSUE RESET command cannot be used
- the ISSUE COPY and ISSUE PRINT commands for the 3270 cannot be used
- the DEST option is available on the SEND command, in addition to other appropriate options

With the exception of 3650 logical units, operations for logical units supported by TCAM use the same options as logical units supported by VTAM.

The 2260 compatibility facilities for the 3270 cannot be used with TCAM.

BTAM PROGRAMMABLE TERMINALS

When BTAM is used by CICS/VS for programmable binary synchronous communication line management, CICS/VS initializes the communication line with a BTAM read initial (TI); the terminal response must be a write initial (TI) or the equivalent. If an application program makes an input request, CICS/VS issues a read continue (TT) to that line; if the application program makes an output request, CICS/VS issues a read interrupt (RVI) to that line. If end of transmission (EOT) is not received on the RVI, CICS/VS issues a read continue(TT) until the EOT is received. When TCAM is used, all of this line control is handled by the MCP rather than by CICS/VS.

The programmable terminal response to a read interrupt must be "end of

transmission" (EOT). The EOT response may, however, be preceded by writes, in order to exhaust the contents of output buffers; this is provided the input buffer size is not exceeded by this data. The input buffer size is specified by the system programmer during preparation of the terminal control table. CICS/VS issues a read continue until it receives an EOT, or until the input message exceeds the size of the input buffer (an error condition).

After receiving an EOT, CICS/VS issues a write initial (TI) or the equivalent (depending on the type of line). The programmable terminal response must be a read initial (TI) or the equivalent.

If the application program makes another output request, CICS/VS issues a write continue (TT) to that line. If the application program makes an input request after it has made an output request, CICS/VS turns the line around with a write reset (TR). (CICS/VS does not recognize a read interrupt.)

To ensure that binary synchronous terminals (for example, System/370, 1130, 2780) remain coordinated, CICS/VS processes the data collection or data transmission transaction on any line to completion, before polling other terminals on that line.

The programmable terminal actions required for the above activity, with the corresponding user application program commands and CICS/VS actions, are summarized in Figure 13.

Automatically initiated transactions attached to a device will cause message DFH2503 to be sent to the device which must be prepared to action it.

Input data is deblocked to ETX, ETB, RS, and US characters. These characters are moved with the data but are not included in the data length. Characters such as NL, CR, LF, and EM are included as data in a CICS/VS application program.

Application Program Command	CICS/VS[1]	Programmable Terminal Program
	Read initial (TI)	Write initial (TI)
RECEIVE	Read continue (TT)	Write continue (TT)
SEND	Read interrupt (RVI)[2]	Write reset (TR) or
	Read continue (TT)[3]	Write continue Write reset
	Write initial (TI)	Read initial (TI)
SEND	Write continue (TT)	Read continue (TT)
RECEIVE	Write reset (TR)[4]	Read continue (TT)
	Read initial (TI)	Write initial (TI)

[1] CICS/VS issues the macro shown, or, if the line is switched, the equivalent. The user-written programmable terminal program must issue the equivalent of the BTAM operation shown.

[2] An RVI sequence is indicated by the DECFLAGS field of the data event control block (DECB) being set to X'02' and a completion code of X'7F' being returned to the event control block (ECB).

[3] The read continue is issued only if the EOT character is not received on the the read interrupt.

[4] Write reset is issued only for point-to-point terminals.

Figure 13. BTAM Programmable Terminal Programming

TELETYPEWRITER PROGRAMMING

The teletypewriter (World Trade only) uses two different control characters for print formatting, as follows:

- < carriage return, (X'22' in ITA2 code or X'15' in EBCDIC)
- = line feed, (X'28' in ITA2 code or X'25' in EBCDIC)

The character < should always be used first; that is <= or <===, but never =<, otherwise following characters (data) may be printed while the typebar is moving to the left.

MESSAGE FORMAT

Message Begin: To start a message on a new line at the left margin, the message text must begin with X'1517' (EBCDIC). CICS/VS recognizes the X'17' and changes it to X'25' (X'17' is an IDLE character).

Message Body: To write several lines with a single transmission, the lines must be separated by X'1525', or if multiple blank lines are required, by X'152525...25'.

Message End Before Next Input: To allow input of the next message on a line at the left margin, the preceding message must end with X'1517'. CICS/VS recognizes X'15' and changes the character following it to X'25'.

Message End Before Next Output: In the case of two or more successive output messages, the message begin and the message end look the same; that is X'1517', except for the last message (see above). To make the message end of the preceding message distinguishable from the message begin of the next message, the penultimate character of the message end must not be X'15'.

MESSAGE LENGTH

It is recommended that messages for teletypewriter terminals do not exceed a length of about 3000 bytes or approximately 300 words.

CONNECTION THROUGH VTAM

Both the TWX Model 33/35 Common Carrier Teletypewriter Exchange and the WTTY Teletypewriter (World Trade only) can be connected to CICS/VS through BTAM, or through VTAM using NTO.

If a device is connected through VTAM using NTO, the protocols used are the same as for the 3767 logical unit, and the application program can make use of these protocols (for example, HANDLE CONDITION SIGNAL). However, the data stream is not translated to a 3767 data stream but remains as that for a TWX/WTTY.

DISPLAY DEVICE OPERATIONS

Besides the standard terminal control commands for sending and receiving data, several additional commands and lists are provided for use with display devices such as the 3270, as follows:

- Print displayed information (ISSUE PRINT).
- Copy displayed information (ISSUE COPY).
- Erase all unprotected fields (ISSUE ERASEAUP).
- Input operation without Data (RECEIVE).
- Standard Attention Identifier List (DFHAID).
- Handling Attention Identifiers (HANDLE AID).
- Standard Attribute and Printer Control Character List (DFHBMSCA).

For devices with switchable screen sizes, the size of the screen that can be used, and the size to be used for a given transaction, are defined by CICS/VS table generation. These values can be obtained by means of the ASSIGN command, described in "Chapter 1.6. Access to System Information" on page 29.

The ERASE option should always be included in the first SEND command to clear the screen and format it according to the transmitted data. This first SEND with ERASE will select also the screen size to be used, as specified in the PCT and TCT. If ERASE is omitted, the screensize will be the same as its previous setting, which may be incorrect.

Use of the CLEAR key outside of a transaction will set the screen to its default size.

PRINT DISPLAYED INFORMATION (ISSUE PRINT)

If the 3270 print request facility is included in the terminal control program at CICS/VS system generation, the ISSUE PRINT command will cause the displayed data to be printed on the first available, print-request-eligible printer. For a BTAM-supported 3270, this is a printer on the same control unit. For a 3270 logical unit or a 3650 host-conversational (3270) logical unit, it is a printer predesignated by the

system programmer using the PRINTTO or ALTPRT operands of the DFHTCT TYPE=TERMINAL macro. For a 3270-display logical unit with the PTRADAPT feature (LUTYPE2 specified in the TRMTYPE= operand and PTRADAPT specified in the FEATURE=operand of the DFHTCT TYPE=TERMINAL system macro) used with a 3274 or 3276, it is a printer allocated by the printer authorization matrix. (See the IBM 3270 Information Display System Component Description for details of this matrix.) For a 3790 (3270-display) logical unit, it is a printer allocated by the 3790.

For a printer to be available it must be in service and not currently attached to a task.

For a BTAM printer to be eligible, it must be attached to the same control unit as the display, must have a buffer capacity equal to or greater than that of the display, and must have FEATURE=PRINT specified in the associated DFHTCT TYPE=TERMINAL system macro.

For a 3270 logical unit to be eligible, it must have been specified by the system programmer, using the PRINTTO or ALTPRT operands, and it must have the correct buffer capacity; FEATURE=PRINT is not necessary. If COPY is specified with the ALTPRT or PRINTTO operands, the printer must be on the same control unit.

For some 3270 displays, it is possible also to print the displayed information without using CICS/VS. For further details see under "printer authorization matrix" in the IBM 3270 Information Display System Component Description.

COPY DISPLAYED INFORMATION (ISSUE COPY)

The ISSUE COPY command is used to copy the format and data contained in the buffer of a specified terminal into the buffer of the terminal that started the transaction. This command cannot be used for an LUTYPE2. Both terminals must be attached to the same remote control unit. The terminal whose buffer is to be copied is identified in the TERMID option. If the terminal identifier is invalid, that is, it does not exist in the TCT, the TERMIDERR condition will occur. The copy function to be performed is defined by the Copy Control Character (CCC) specified in the CTLCHAR option of the ISSUE COPY command.

The WAIT option of the ISSUE COPY command ensures that the operation has been completed before control is returned to the application program.

ERASE ALL UNPROTECTED FIELDS (ISSUE ERASEAUP)

The ISSUE ERASEAUP command is used to erase all unprotected fields of a 3270 buffer. The following actions are performed:

1. All unprotected fields are cleared to nulls (X'00').
2. The modified data tags (MDTs) in each unprotected field are reset to zero.
3. The cursor is positioned to the first unprotected field.
4. The keyboard is restored.

The WAIT option of the ISSUE ERASEAUP command ensures that the operation has been completed before control is returned to the application program.

INPUT OPERATION WITHOUT DATA (RECEIVE)

The RECEIVE command with no options causes input to take place and the EIB to be updated. However, data received by CICS/VS is not passed on to the application program and is lost. A wait will be implied. Two of the fields in the EIB that are updated are described below:

Cursor Position (EIBCPOSN) - For every terminal control (or BMS) input operation associated with a display device, the screen cursor address (position) is placed in the EIBCPOSN field in the EIB. The cursor address is in the form of a halfword binary value and remains until updated by a new input operation.

Attention Identifier (EIBAID) - For every terminal control (or BMS) input operation associated with a display device, an attention identifier (AID) is placed in field EIBAID in the EIB. The AID indicates which method the terminal operator has used to initiate the transfer of information from the device to CICS/VS; for example, the ENTER key, a program function key, the light pen, and so on. The field contents remain unaltered until updated by a new input operation.

Field EIBAID can be tested after each terminal control (or BMS) input operation to determine further processing and a standard attention identifier list (DFHAID) is provided for this purpose. Alternatively, the HANDLE AID command can be used to pass control to specified labels when the AIDs are received. The standard attention identifier list and the HANDLE AID command are described in the next two sections.

STANDARD ATTENTION IDENTIFIER LIST (DFHAID)

The standard attention identifier list DFHAID simplifies testing the contents of the EIBAID field. The following list is obtained by copying DFHAID into the application program and shows the symbolic names for the attention identifiers (AIDS) and the corresponding 3270 functions.

For COBOL users, the list consists of a set of 01 statements that must be copied into the working-storage section. For PL/I users, the list consists of DECLARE statements defining elementary character variables.

Name	3270 Function
DFHCLEAR	CLEAR key
DFHENTER	ENTER key
DFHOPID	Operator identification card reader or MSR
DFHMSRE	Extended (standard) MSR
DFHTRIG	Trigger field
DFHPA1	PA1 key
DFHPA2	PA2 key
DFHPA3	PA3 key
DFHPEN	Light pen attention
DFHPF1	PF1 key
DFHPF2	PF2 key
.	.
.	.
DFHPF24	PF24 key

HANDLING ATTENTION IDENTIFIERS (HANDLE AID)

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

This command is used to specify the label to which control is to be passed when an AID is received from a display device. Control is passed after the input command is completed; that is, any data received in addition to the AID has been passed to the application program . In the absence of a HANDLE AID command, control returns to the application program at the point immediately following the input command.

No more than twelve options are allowed in the same command.

A HANDLE AID command will take precedence over a HANDLE CONDITION command (see "Chapter 1.4. Programming Techniques and Restrictions" on page 17); if an AID is received during an input operation, for which a HANDLE AID command is active, control will pass to the label specified in that command, regardless of any conditions that may have occurred (but which did not stop receipt of the AID).

The options that can be specified are:

- Program attention key names (PA1, PA2, or PA3)
- Program function key names (PF1 through PF24)
- CLEAR or ENTER (for the keys of the same names)
- LIGHTPEN (for a light pen attention)
- OPERID (for the operator identification card reader, the magnetic slot reader (MSR), or the extended MSR)
- ANYKEY (any PA key, any PF key, or the CLEAR key, but not the ENTER key)

The HANDLE AID command for a given AID applies only to the task in which it is specified, remaining active until the task is terminated, or until another HANDLE AID command for the same AID 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 AID commands that were active in the higher-level program before control was transferred from it are reactivated, and those in the lower-level program are deactivated. (Refer to "Chapter 4.4. Program Control" on page 189 for information about logical levels.)

If no HANDLE AID command is active for any PA key, any PF key, or the CLEAR key, but one is active for ANYKEY, control will be passed to the label specified for ANYKEY. A HANDLE AID command for an AID overrides the HANDLE AID ANYKEY command for that AID.

The following example shows a HANDLE AID command that specifies one label for the PA1 key AID, a second label for the PA2 and PA3 key AIDs, all of the PF key AIDs except PF10, and the CLEAR key AID. If a PF10 AID is received, control returns to the application program at the instruction immediately following the input command.

```
EXEC CICS HANDLE AID
     PA1(LAB1)
     ANYKEY(LAB2)
     PF10
```

If a task is initiated from a terminal by means of an AID, the first RECEIVE command in the task will not read from the terminal but will copy only the input buffer (even if the length of the data is zero) so that control may be passed by means of a HANDLE AID command for that AID.

A BMS RECEIVE MAP command with the FROM option will not cause a HANDLE AID

command to be invoked because no terminal input is involved.

STANDARD ATTRIBUTE AND PRINTER CONTROL CHARACTER LIST (DFHBMSCA)

The standard list DFHBMSCA simplifies the provision of field attributes and printer control characters. The list is obtained by copying DFHBMSCA into the application program. The symbolic names for the various combinations of attributes and control characters are given below. Combinations other than shown must be generated separately.

Name	Attribute/Control Character
DFHBMPEM	Printer end-of-message
DFHBMPNL	Printer new-line character
DFHBMASK	Autoskip
DFHBMUNP	Unprotected
DFHBMUNN	Unprotected; numeric
DFHBMPRO	Protected
DFHBMBRY	High intensity
DFHBMDAR	Dark; nonprint
DFHBMFSE	MDT set to 1
DFHBMPRF	Protected; MDT set to 1
DFHBMASF	Autoskip; MDT set to 1
DFHBMASB	Autoskip; high intensity
DFHSA[1]	Set attribute order
DFHCOLOR[1]	Color attribute code
DFHPS[1]	PS attribute code
DFHHLT[1]	Highlight attribute code
DFH3270[1]	3270 attribute code
DFHVAL[1]	Validation attribute code
DFHALL[1]	X'00'(Reset all attributes)
DFHERROR	X'3F'(Error code)
DFHDFT	X'FF'(Default for maps)
DFHDFCOL[1]	Default color
DFHBLUE	Blue
DFHRED	Red
DFHPINK	Pink
DFHGREEN	Green
DFHTURQ	Turquoise
DFHYELLO	Yellow
DFHNEUTR	Neutral
DFHBASE[1]	Base PS
DFHDFHI[1]	Default highlight
DFHBLINK	Blink
DFHREVRS	Reverse video
DFHUNDLN	Underline
DFHMFIL[2]	Mandatory fill
DFHMENT[2]	Mandatory enter
DFHMFE[2]	Mandatory fill and enter

[1] For text processing only. Use for constructing embedded set attribute orders in user text

[2] Cannot be used in set attribute orders

For assembler-language users, the list consists of a set of EQU statements. For COBOL users, the list consists of a set of 01 statements that must be copied into the working-storage section. For PL/I users, the list consists of DECLARE statements defining elementary character variables.

The symbolic name DFHDFT must be used in the application structure to override a map attribute with the default. On the other hand, to specify default values in a set attribute (SA) sequence in text build, the symbolic names DFHDFCOL, DFHBASE, OR DFHDFHI should be used.

STANDARD CICS/VS TERMINAL SUPPORT (BTAM OR TCAM)

```
RECEIVE {INTO(data-area)|SET(ptr-ref)}
LENGTH(data-area)

Condition:  LENGERR
```

```
SEND FROM(data-area)
LENGTH(data-value)
[DEST(name)]
[WAIT]
```

```
CONVERSE FROM(data-area)
FROMLENGTH(data-value)
[INTO(data-area)|SET(ptr-ref)]
[TOLENGTH(data-area)]
[DEST(name)]

Condition:  LENGERR
```

```
ISSUE RESET
ISSUE DISCONNECT
```

These commands can be used by all terminals supported by CICS/VS that are not dealt with separately in the following sections.

LUTYPE4 LOGICAL UNIT

```
RECEIVE {INTO(data-area)|SET(ptr-ref)}
LENGTH(data-area)

Conditions:  EOC, EODS, INBFMH,
LENGERR, SIGNAL
```

```
SEND FROM(data-area)
LENGTH(data-value)
[WAIT]
[INVITE|LAST]
[CNOTCOMPL|DEFRESP]
[FMH]

Conditions:  IGREQCD, SIGNAL
```

```
CONVERSE FROM(data-area)
FROMLENGTH(data-value)
[INTO(data-area)|SET(ptr-ref)]
[TOLENGTH(data-area)]
[DEFRESP]
[FMH]

Conditions:  EOC, EODS, IGREQCD,
INBFMH, LENGERR, SIGNAL
```

```
FREE [SESSION(name)]

Conditions:  INVREQ, NOTALLOC,
SESSIONERR
```

```
WAIT SIGNAL

Condition:  SIGNAL
```

```
ISSUE DISCONNECT

Condition:  SIGNAL
```

LUTYPE6 LOGICAL UNIT

```
RECEIVE [SESSION(name)]
{INTO (data-area)|SET(ptr-ref)}
LENGTH(data-area)

Conditions:  INBFMH, NOTALLOC,
LENGERR, SESSIONERR, SIGNAL
```

```
SEND [SESSION(name)]
[WAIT]
[INVITE|LAST]
[ATTACHID(name)]
[FROM(name)]
[LENGTH(name)]
[FMH]
[DEFRESP]

Conditions:  CBIDERR, NOTALLOC,
SESSIONERR, SIGNAL
```

```
CONVERSE [SESSION(name)]
[ATTACHID(name)]
[FROM(name)]
[FROMLENGTH(name)]
[INTO(data-area)|SET(ptr-ref)]
[TOLENGTH(data-area)]
[FMH]
[DEFRESP]

Conditions:  CBIDERR, INBFMH,
LENGERR, NOTALLOC, SESSIONERR, SIGNAL
```

```
ALLOCATE {SYSID(name)|SESSION(name)}
[PROFILE(name)]

Conditions:  CBIDERR, INVREQ,
SESSBUSY, SESSIONERR, SYSBUSY,
SYSIDERR
```

```
BUILD ATTACH
[ATTACHID(name)]
[PROCESS(name)] [RESOURCE(name)]
[RPROCESS(name)] [RRESOURCE(name)]
[QUEUE(name)] [IUTYPE(name)]
[DATASTR(name)] [RECFM(name)]
```

```
EXTRACT ATTACH
[ATTACHID(name)|SESSION(data-area)]
[PROCESS(data-area)]
[RESOURCE(data-area)]
[RPROCESS(data-area)]
[RRESOURCE(data-area)]
[QUEUE(data-area)]
[IUTYPE(data-area)]
[DATASTR(data-area)]
[RECFM(data-area)]

Conditions:  CBIDERR, INVREQ,
NOTALLOC, SESSIONERR
```

EXTRACT TCT NETNAME(name) {SYSID(data-area)|TERMID(data-area)} Condition: INVREQ
FREE [SESSION(name)] Conditions: INVREQ, NOTALLOC, SESSIONERR
POINT [SESSION(name)] Conditions: NOTALLOC, SESSIONERR
WAIT SIGNAL
WAIT TERMINAL [SESSION(name)] Conditions: NOTALLOC, SESSIONERR, SIGNAL
ISSUE DISCONNECT [SESSION(name)] Conditions: NOTALLOC, SESSIONERR
ISSUE SIGNAL [SESSION(name)] Conditions: NOTALLOC

The ALLOCATE command is used to acquire an alternate facility and to select optionally a set of terminal control processing options. If SYSID is specified, CICS/VS will make available to the application program one of the sessions associated with the named system. The name of this session can be obtained from field EIBRSRCE in the EIB. If SESSION is specified, CICS/VS will make the named session available.

The BUILD ATTACH command is used to specify a set of values to be placed in the named attach header control block. This control block contains values that are to be sent in an LU6 attach FMH which is constructed by CICS/VS, and is sent only when a SEND ATTACHID or CONVERSE ATTACHID command is executed. The specified values override existing values in the control block; unspecified values are set to default values.

The EXTRACT ATTACH command is used to retrieve a set of values held in an attach header control block or that have been built previously. This control block contains values received in an attach FMH or that have been built previously.

The EXTRACT TCT command is used to allow the eight-character VTAM network name for a terminal or logical unit to be converted into a corresponding four-character name by which it is known in the local CICS/VS system.

The FREE command is used to return a facility to CICS/VS when a transaction owning it no longer requires it. The facility can then be allocated for use by other transactions. A facility can be freed only when it is in free mode (EIBFREE set to X'FF').

The POINT command is used to obtain information about a named facility, such as whether it owns the given facility.

SESSION STATUS INFORMATION

This information consists of several fields that contain application-oriented and session-oriented information when an LU6 session is in progress. These fields are located in the EIB.

Session status information is set to zeros at the start of execution of every command and is updated whenever a RECEIVE or CONVERSE command naming an LU6 session is executed. If the information is to be retained across the execution of several commands, the user must take steps to preserve it.

APPLICATION-ORIENTED INFORMATION

The application-oriented information determines the action taken by function processing logic. The information consists of, for example, indicators (such as end-of-chain), an attach header, and user data.

The user data is moved to an area specified in the application program; alternatively the address of the user data is passed to the application program.

The indicators, together with an attach header indicator, are passed to the application program in the EIB. The EXTRACT ATTACH command (described earlier in the chapter) can be used to process the attach header data if such data exists.

The following application-oriented fields, each one byte in length, appear in the EIB: EIBATT, EIBEOC, and EIBFMH.

SESSION-ORIENTED INFORMATION

The session-oriented information determines the action taken by session-handling logic, for example, syncpoint requested. This information is available to the application program in

fields EIBSYNC, EIBFREE, and EIBRECV in the EIB, and should be processed in that order, before further operations, such as SEND, RECEIVE, CONVERSE, or FREE, are performed on the session.

SYSTEM/3

```
RECEIVE {INTO(data-area)|
 SET(ptr-ref)}
LENGTH(data-area)
[ASIS]

Condition:  LENGERR
```

```
SEND FROM(data-area)
LENGTH(data-value)
[DEST(name)]
[WAIT]
[ASIS]
[CNOTCOMPL]
```

```
CONVERSE FROM(data-area)
FROMLENGTH(data-value)
[INTO(data-area)|SET(ptr-ref)]
[TOLENGTH(data-area)]
[DEST(name)]

Condition:  LENGERR
```

SYSTEM/370

Support and command syntax as for System/3.

SYSTEM/7

```
RECEIVE {INTO(data-area)|
 SET(ptr-ref)}
LENGTH(data-area)
[PSEUDOBIN][1]
[ASIS]

Condition:  LENGERR
```

```
SEND FROM(data-area)
LENGTH(data-value)
[DEST(name)]
[WAIT]
[PSEUDOBIN][1]
[ASIS]
```

```
CONVERSE FROM(data-area)
FROMLENGTH(data-value)
[INTO(data-area)|SET(ptr-ref)]
[TOLENGTH(data-area)]
[DEST(name)]

Condition:  LENGERR
```

```
ISSUE RESET
ISSUE DISCONNECT
```

Start-stop only

Transactions are normally initiated from the System/7 by issuing a four-character transaction code which is transmitted in BCD mode. Pseudobinary mode can be used only while communicating with an active CICS/VS transaction; it cannot be used to initiate the transaction. The message length is given as the number of words to be transmitted (not as the number of characters).

When a transaction is initiated on a System/7, CICS/VS services that System/7 only for the duration of the transaction; that is, to ensure efficient use of the line, any other System/7s on the same line are locked out for the duration of the transaction. CICS/VS application programs for the multipoint System/7 should be designed with the shortest possible execution time.

The first word (two characters) of every message received by the System/7 must be an identification word, except words beginning with "@" (X'20') which are reserved by CICS/VS.

When the PSEUDOBIN option is specified, the length of the data-area provided by the application program must be at least twice that of the data to be read.

In the case of a System/7 on a dial-up (switched) line, the System/7

application program must, initially, transmit a four-character terminal identification. (This terminal identification is generated during preparation of the TCT through use of the DFHTCT TYPE=TERMINAL, TRMIDNT=parameter specification.) CICS/VS responds with either a "ready" message, indicating that the terminal identification is valid and that the System/7 may proceed as if it were on a leased line, or an INVALID TERMINAL IDENTIFICATION message, indicating that the terminal identification sent by the System/7 did not match the TRMIDNT=parameter specified.

Whenever CICS/VS initiates the connection to a dial-up System/7, CICS/VS writes a null message, consisting of three idle characters, prior to starting the transaction. If there is no program resident in the System/7 capable of supporting the Asynchronous Communication Control Adapter (ACCA), BTAM error routines cause a data check message to be recorded on the CICS/VS (host) system console. This is normal if the task initiated by CICS/VS is to IPL the System/7. Although the data check message is printed, CICS/VS ignores the error and continues normal processing. If a program capable of supporting the ACCA is resident in the System/7 at the time this message is transmitted, no data check occurs.

When a disconnect is issued to a dial-up System/7, the 'busy' bit is sometimes left on in the interrupt status word of the ACCA. If the line connection is reestablished by dialing from the System/7 end, the 'busy' condition of the ACCA prevents message transmission from the System/7. To overcome this problem, the System/7 program must reset the ACCA after each disconnect and before message transmission is attempted. This can be done by issuing the following instruction:

```
PWRI    0,8,3,0     RESET ACCA
```

This procedure is not necessary when the line is reconnected by CICS/VS (that is, by an automatically initiated transaction).

2260 DISPLAY STATION

```
RECEIVE {INTO(data-area)|
 SET(ptr-ref)}
LENGTH(data-area)
[LEAVEKB]

Condition:  LENGERR
```

```
SEND FROM(data-area)
LENGTH(data-value)
[CTLCHAR(data-value)]
[DEST(name)]
[LINEADDR(data-value)]
[WAIT]
[LEAVEKB]
```

```
CONVERSE FROM(data-area)
FROMLENGTH(data-value)
[INTO(data-area)|SET(ptr-ref)]
[TOLENGTH(data-area)]
[CTLCHAR(data-value)]
[DEST(name)]
[LINEADDR(data-value)]

Condition:  LENGERR
```

```
ISSUE RESET
ISSUE DISCONNECT
```

The LINEADDR option specifies on which line of a 2260 screen writing is to begin. A line number in the range 1 through 12 must be provided in the application program.

2265 DISPLAY STATION

Support and command syntax as for the 2260 Display Station except that a line number in the range 1 through 15 must be provided in the application program.

2741 COMMUNICATION TERMINAL

```
RECEIVE {INTO(data-area)|
 SET(ptr-ref)}
LENGTH(data-area)

Conditions:  LENGERR, RDATT(not TCAM)
```

```
SEND FROM(data-area)
LENGTH(data-value)
[DEST(name)]
[WAIT]

Condition:  WRBRK
```

```
CONVERSE FROM(data-area)
FROMLENGTH(data-value)
[INTO(data-area)|SET(ptr-ref)]
[TOLENGTH(data-area)]
[DEST(name)]

Conditions:  LENGERR, RDATT (not
TCAM), WRBRK
```

```
ISSUE RESET
ISSUE DISCONNECT
```

READ ATTENTION

If the terminal operator presses the Attention key on the 2741 after typing a message, it is recognized as a Read Attention if:

- Read Attention support is generated into the system (CICS/OS/VS or CICS/DOS/VS).
- The message is read by a RECEIVE command.

When this occurs, control is transferred to a CICS/VS read attention exit routine, if it has been generated into the system. This routine is a skeleton program that can be tailored by the system programmer to carry out actions such as the following:

- Perform data analysis or modification on a Read Attention.
- Return a common response to the terminal operator following a Read Attention.
- Return a response and request additional input that can be read into the initial input area or into a new area.
- Request new I/O without requiring a return to the task to request additional input.

When the Read Attention exit routine is completed, control is returned to the application program at the address specified in the HANDLE CONDITION RDATT command. The return is made whenever one of the following occurs:

- The exit routine issues no more requests for input.
- The exit routine issues a RECEIVE request and the operator terminates the input with a carriage return. (If the operator terminates the input with an Attention, the exit routine is reentered and is free to issue another RECEIVE request).

If a HANDLE CONDITION RDATT command is not included in the application program or Read Attention support has not been generated, the attention is treated as if the return key had been pressed.

WRITE BREAK (CICS/OS/VS ONLY)

If the terminal operator presses the Attention key on the 2741 while a message is being received, it is recognized as a Write Break if:

- Write Break support is generated into the system (available only in CICS/OS/VS) by the system programmer.
- A HANDLE CONDITION WRBRK command is active in the application program.

When this occurs, the remaining portion of the message is not sent to the terminal. The write is terminated as though it were successful, and a new-line character (X'15') is sent to cause a carrier return. Control is returned to the application program at the address specified for the WRBRK condition.

If a HANDLE CONDITION WRBRK command is not included in the application program or if Write Break support has not been generated, the attention is treated as an I/O error.

2770 DATA COMMUNICATION SYSTEM

Support and command syntax as for System/3. The 2770 recognizes a read interrupt and responds by transmitting the contents of the I/O buffer. After the contents of the buffer have been transmitted, the 2770 responds to the next read continue with an EOT. If the I/O buffer is empty, the 2770 transmits an EOT. CICS/VS issues a read interrupt and read continue to relinquish use of the line and to enable the application program to write to the 2770.

Input from a 2770 consists of one or more logical records. CICS/VS provides one logical record for each read request to the application program. The size of a logical record cannot exceed the size of the I/O buffer. If the input spans multiple buffers, multiple reads must be issued by the application program.

The 2265 component of the 2770 Data Communication System is controlled by data stream characters, not BTAM macro instructions; appropriate screen control characters should be included in the output area.

For 2770 input, data is deblocked to ETX, ETB, RS, and US characters. These characters are moved with the data to the input area but are not included in the data length; characters such as NL, CR, and LF are passed in the input area as data.

2780 DATA TRANSMISSION TERMINAL

Support and command syntax as for System/3. The 2780 recognizes a read interrupt and responds by transmitting the contents of the I/O buffer. After the contents of the buffer have been transmitted, the 2780 responds to the next read continue with an EOT. If the I/O buffer is empty, the 2780 transmits an EOT. CICS/VS issues a read interrupt and read continue to relinquish use of the line and to enable the application program to write to the 2780.

Input from a 2780 consists of one or more logical records. CICS/VS provides one logical record for each read request to the application program. The size of a logical record cannot exceed the size of the I/O buffer. If the input spans multiple buffers, multiple reads must be issued by the application program.

Output to a 2780 requires that the application program contains an appropriate "escape sequence" for component selection associated with the output message. (For programming details, see the publication Component Description: IBM 2780 Data Transmission Terminal.)

For 2780 input, data is deblocked to ETX, ETB, RS, and US characters. These characters are moved with the data to the input area but are not included in the data length; characters such as NL, CR, and LF are passed in the input area as data.

2980 GENERAL BANKING TERMINAL SYSTEM

```
RECEIVE {INTO(data-area)|
 SET(ptr-ref)}
LENGTH(data-area)
PASSBK

Conditions:  LENGERR, NOPASSBKRD
```

```
SEND FROM(data-area)
LENGTH(data-value)
[DEST(name)]
{PASSBK|CBUFF}

Condition:  NOPASSBKWR
```

PASSBOOK CONTROL

All input and output requests to the passbook area of a 2980 are dependent on the presence of a passbook. The PASSBK option is used to specify the passbook area. The conditions NOPASSBKRD and NOPASSBKWR will occur on input and output requests respectively when a passbook is not present. These conditions can be handled by a HANDLE CONDITION command and appropriate handling routines.

If the passbook is present on an input request, the application program generally writes back to the passbook area to update the passbook. If the NOPASSBKWR condition occurs, CICS/VS allows immediate output to the terminal. In a routine for the NOPASSBKWR condition, the application program should send an error message to the journal area of the terminal to inform the 2980 operator of this error condition. To allow the operator to insert the required passbook, CICS/VS automatically causes the transaction to wait 23.5 seconds before continuing.

On regaining control from CICS/VS after sending the error message, the application program can attempt again to update the passbook when it has ensured that the print element is positioned correctly in the passbook area. This is generally accomplished by issuing two carrier returns followed by the number of tabs required to move the print element to the correct position. (See "The DFH2980 Structure" later in this section).

If the NOPASSBKWR condition occurs during the second attempt to write to the passbook area, the application program can send another error message or take some alternative action (for example, place the terminal "out of service").

The presence of the Auditor Key on a 2980 Administrative Station Model 2 is controlled by the SEND PASSBK command and

may be used in a manner similar to that described above.

OUTPUT CONTROL

The unit of transmission for a 2980 is called a segment. A segment is equivalent to the buffer size of the 2972 Control Unit. However, for the passbook and journal areas, CICS/VS allows an application program to send messages that exceed the buffer size. For the passbook area, the maximum length of message is limited to one line of a passbook to avoid spacing (indexing) past the bottom of the passbook. For the journal area, the maximum length of message is specified in the LENGTH option of the SEND command.

For example, consider a 2972 buffer size of 48 characters and a 2980 Teller Station Model 4 passbook print area of 100 characters/line. The application program can send a message of 100 characters to this area; CICS/VS automatically segments the message to adjust to the buffer size. The application program must insert the passbook indexing character (X'25') as the **last** character written in one output request to the passbook area. This is done to control passbook indexing and thereby achieve positive control of passbook presence.

If a message contains embedded passbook index characters, and segmentation is necessary because of the length of the message, the output is terminated if the passbook spaces beyond the bottom of the passbook; the remaining segments are not printed.

OUTPUT TO A COMMON BUFFER

The SEND CBUFF command is used to transmit data to a common buffer. The data is translated to the character set of the receiving 2980 model. If more than one 2980 model type is connected to the 2972 Control Unit, the lengths are automatically truncated if they exceed the buffer size.

THE DFH2980 STRUCTURE

The DFH2980 structure contains constants that may be used when writing only COBOL or PL/I application programs for the 2980. The structure is obtained by copying DFH2980 into the application program.

For COBOL, DFH2980 is copied into the Working Storage section; for PL/I, DFH2980 is included using a %INCLUDE statement.

The station identification is given in the field STATIONID, whose value must be determined by the ASSIGN command. To test whether a normal or alternate station is being used, the STATIONID field is compared with values predefined in DFH2980. The values are:

```
STATION-#-A or STATION-#-N     (COBOL)

STATION_#_A or STATION_#_N     (PL/I)
```

where # is an integer (0 through 9) and A and N signify alternate and normal stations. (The break symbol is "-" (minus) for COBOL, and "_" (underline) for PL/I.)

The teller identification on a 2980 Teller Station Model 4 is given in the one-byte character field TELLERID. An ASSIGN command must be used to determine the TELLERID value.

Tab characters (X'05') must be included in the application program. The number of tabs required to position the print element to the first position of a passbook area is given in the field NUMTAB. An ASSIGN command must be used to determine the NUMTAB value. The value of NUMTAB is specified by the system programmer and may be unique to each terminal.

Other tab characters are inserted as needed to control formatting.

Any of the DFH2980 values TAB-ZERO through TAB-NINE for COBOL and PL/I, may be compared with NUMTAB to determine the number of tab characters that need to be inserted in an output message to obtain correct positioning of the print element. The tab character is included in DFH2980 as TABCHAR.

Thirty special characters are defined in DFH2980. Twenty-three of these can be referred to by the name SPECCHAR-# or SPECCHAR_# (for ANS COBOL or PL/I) where # is an integer (0 through 22). The seven other characters are defined with names that imply their usage, for example, TABCHAR. For further information on these thirty characters, see "Appendix B. Translation Tables for the 2980" on page 243.

Several other characters defined in DFH2980, such as HOLDPCF or TCTTEPCR, are intended for use in application programs using CICS/VS macro-instructions and should not be required in application programs using CICS/VS commands.

3270 INFORMATION DISPLAY SYSTEM (BTAM OR TCAM)

```
RECEIVE {INTO(data-area)|
 SET(ptr-ref)}
LENGTH(data-area)
[ASIS]
[BUFFER]                    (not TCAM)

Condition:  LENGERR
```

```
SEND FROM(data-area)
LENGTH(data-value)
[DEST(name)]                (TCAM only)
[WAIT]
[STRFIELD|[[ERASE]
 [CTLCHAR(data-value)]]]
```

```
CONVERSE FROM(data-area)
FROMLENGTH(data-value)
[INTO(data-area)|SET(ptr-ref)]
[TOLENGTH(data-area)]
[STRFIELD|[[ERASE]
 [CTLCHAR(data-value)]]]

Condition:  LENGERR
```

```
ISSUE PRINT[1]
```

```
ISSUE COPY[1]TERMID(name)
[CTLCHAR(data-value)]
[WAIT]

Condition:  TERMIDERR
```

```
ISSUE ERASEAUP [WAIT]
```

```
ISSUE RESET
ISSUE DISCONNECT
```

[1] The ISSUE PRINT and ISSUE COPY commands cannot be used with TCAM.

3270 IN 2260 COMPATIBILITY MODE (BTAM)

```
RECEIVE {INTO(data-area)|
 SET(ptr-ref)}
LENGTH(data-area)
[LEAVEKB]

Condition:  LENGERR
```

```
SEND FROM(data-area)
LENGTH(data-value)
[LINEADDR(data-value)]
[WAIT]
[ERASE]
[LEAVEKB]
```

```
CONVERSE FROM(data-area)
FROMLENGTH(data-value)
[INTO(data-area)|SET(ptr-ref)]
[TOLENGTH(data-area)]
[LINEADDR(data-value)]
[ERASE]

Condition:  LENGERR
```

```
ISSUE DISCONNECT
```

On output, a SEND ERASE command will clear the screen and set the cursor to the upper left corner before writing starts.

3270 LOGICAL UNIT

```
RECEIVE {INTO(data-area)|
 SET(ptr-ref)}
LENGTH(data-area)
[ASIS]
[BUFFER]

Condition:  LENGERR
```

```
SEND FROM(data-area)
LENGTH(data-value)
[WAIT]
[INVITE|LAST]
[STRFIELD|[[ERASE]
 [CTLCHAR(data-value)]]]
[DEFRESP]
```

```
CONVERSE FROM(data-area)
FROMLENGTH(data-value)
[INTO(data-area)|SET(ptr-ref)]
[STRFIELD|[[ERASE]
 [CTLCHAR(data-value)]]
[TOLENGTH(data-area)]
[DEFRESP]

Condition:  LENGERR
```

```
FREE [SESSION(name)]

Conditions:  INVREQ, NOTALLOC,
SESSIONERR
```

```
ISSUE PRINT
```

```
ISSUE COPY TERMID(name)
[CTLCHAR(data-value)]
[WAIT]

Condition:  TERMIDERR
```

```
ISSUE ERASEAUP [WAIT]
```

```
ISSUE DISCONNECT
```

3270 SCS PRINTER LOGICAL UNIT

```
SEND FROM(data-area)
LENGTH(data-value)
[DEST(name)]
[WAIT]
[INVITE|LAST]
[CNOTCOMPL|DEFRESP]
[DEFRESP]
```

```
FREE [SESSION(name)]

Conditions:  INVREQ, NOTALLOC,
SESSIONERR
```

```
ISSUE DISCONNECT
```

The SCS printer logical unit accepts a character string as defined by SNA (Systems Network Architecture). Some devices connected under SNA can send a signal which can be detected by the HANDLE CONDITION SIGNAL command, which in turn can invoke an appropriate handling routine. If necessary, a WAIT SIGNAL command can be used to make the application program wait for the signal. The PA keys on a 3287 can be used in this way, or with a RECEIVE command.

3270-DISPLAY LOGICAL UNIT (LUTYPE2)

```
RECEIVE{INTO(data-area)|
 SET(ptr-ref)}
LENGTH(data-area)
[ASIS]
[BUFFER]

Condition:  LENGERR
```

```
SEND FROM(data-area)
LENGTH(data-value)
[DEST(name)]
[WAIT]
[STRFIELD|[[ERASE]
 [CTLCHAR(data-value)]]]
[INVITE|LAST]
[DEFRESP]
```

```
CONVERSE FROM(data-area)
FROMLENGTH(data-value)
INTO(data-area)|SET(ptr-ref)]
[TOLENGTH(data-area)]
[STRFIELD|[[ERASE]
 [CTLCHAR(data-value)]]
[DEST(name)]
[DEFRESP]

Condition:  LENGERR
```

```
FREE [SESSION(name)]

Conditions:  INVREQ, NOTALLOC,
SESSIONERR
```

```
ISSUE PRINT
```

```
ISSUE ERASEAUP [WAIT]
```

```
ISSUE DISCONNECT
```

3270-PRINTER LOGICAL UNIT (LUTYPE3)

```
RECEIVE {INTO(data-area)|
 SET(ptr-ref)}
LENGTH(data-area)
[ASIS]
[BUFFER]

Condition:  LENGERR
```

```
SEND FROM(data-area)
LENGTH(data-value)
[DEST(name)]
[WAIT]
[STRFIELD|[[ERASE]
[CTLCHAR(data-value)]]]
[INVITE|LAST]
[DEFRESP]
```

```
CONVERSE FROM(data-area)
FROMLENGTH(data-value)
[INTO(data-area)|SET(ptr-ref)]
[TOLENGTH(data-area)]
[STRFIELD|[[ERASE]
 [CTLCHAR(data-value)]]
[DEST(name)]
[DEFRESP]

Condition:  LENGERR
```

```
FREE [SESSION(name)]

Conditions:  INVREQ, NOTALLOC,
SESSIONERR
```

```
ISSUE PRINT
```

```
ISSUE ERASEAUP [WAIT]
```

```
ISSUE DISCONNECT
```

3600 FINANCE COMMUNICATION SYSTEM (BTAM)

```
RECEIVE {INTO(data-area)|
 SET(ptr-ref)}
LENGTH(data-area)

Condition:  LENGERR
```

```
SEND FROM(data-area)
LENGTH(data-value)
[DEST(name)]
[WAIT]
```

```
CONVERSE FROM(data-area)
FROMLENGTH(data-value)
INTO(data-area)|SET(ptr-ref)]
[TOLENGTH(data-area)]
[DEST(name)]

Condition:  LENGERR
```

```
ISSUE RESET

ISSUE DISCONNECT
```

INPUT

The unit of transmission from a 3601 Finance Communication Controller to CICS/VS is a segment consisting of the start-of-text data link control character (STX), the one byte identification of the 3600 logical work station that issued the processor write, the data, and either an end-of-block (ETB) or an end-of-text (ETX) control character.

A logical work station sends a message either in one segment, in which case the segment ends with ETX, or in more than one segment, in which case only the last segment ends with ETX, all others ending with ETB.

The input area passed to the user-written application program consists of the data only. The one-byte field TCTTEDLM, which may be obtained by means of an ASSIGN DELIMITER command, contains flags describing the data-link control character (ETB, ETX, or IRS) that ended the segment. The application program can issue terminal control commands to read the data until it receives a segment ending with ETX. If blocked data is transmitted, it is received by CICS/VS as blocks of segments. Only the first segment in a block starts with the STX control character, and all segments are separated by IRS characters. None of the segments contain ETB or ETX characters except the last, which has the ETX character.

For blocked input, the flags in TCTTEDLM only indicate end of segment, not end of message. The CICS/VS application program still receives only the data, but user-defined conventions may be required to determine the end of the message.

The field TCTTEDLM also indicates the mode of the input, either transparent or non-transparent. Blocked input is non-transparent.

The terminal control program does not pass input containing a "start of header" (SOH) data link control character to a user-written application program. If it receives an SOH it sets an indicator in TCTTEDLM, passes the input to the user exit in the terminal control program, and then discards it.

OUTPUT

When an application program issues a SEND command, the terminal control program determines, from the value specified in the BUFFER parameter of the DFHTCT TYPE=TERMINAL system macro, the number of segments to be built for the message. It sends the message to the 3600 logical unit either in one segment consisting of a start-of-text character (STX), the data, and an end-of-text character (ETX); or in more than one segment, in which case only the last ends with ETX, all others ending with ETB.

The host input buffer of the 3600 controller and the input segment of the receiving logical unit must be large enough to accommodate the data sent by CICS/VS. However, space for the data link control characters need not be included. The 3600 application program reads the data from the host, by means of an LREAD, until it has received the entire message.

CICS/VS system output messages begin with "DFH" followed by a four-byte message number and the message text. These messages are sent in non-transparent mode. It is suggested that CICS/VS user-written application programs do not send messages starting with "DFH" to the 3601.

RESEND MESSAGE

When a logical unit sends a message to the host and a short-on-storage condition exists or the input is unsolicited (the active task associated with the terminal has not issued a read), the terminal control program sends a "resend" message to the logical unit. The format of this message is DFH1033 RE-ENTER followed by X'15' (a 3600 new line character) followed by the first eight bytes of the text of the message being rejected. No

message is sent to the destinations CSMT or CSTL.

The first eight bytes of data sent to CICS/VS can be used by the 3600 application program to define a convention to associate responses received from CICS/VS with transactions sent to the host, for example, sequence numbers could be used.

If a CICS/VS user-written application program has already issued a SEND command when a resend situation occurs, the resend message is not sent to the 3601 until the user-written application program message has been sent. A 3600 logical unit cannot receive a resend message while receiving a segmented message.

Only one resend message at a time can be queued for a logical unit. If a second resend situation occurs before CICS/VS has written the first, a resend message, containing the eight bytes of data that accompanied the second input transaction from the 3600 logical unit, is sent.

The resend message is sent in transparent mode if the input data from the 3601 to be re-transmitted is received by CICS/VS in transparent mode. Otherwise it is sent in non-transparent mode.

3600 PIPELINE LOGICAL UNIT

SEND FROM(data-area) LENGTH(data-value) [WAIT]
ISSUE DISCONNECT

3600 (3601) LOGICAL UNIT

RECEIVE {INTO(data-area)| SET(ptr-ref)} LENGTH(data-area) Conditions: EOC, EODS, INBFMH, LENGERR, SIGNAL
SEND FROM(data-area) LENGTH(data-value) [LDC(name)|FMH] [DEST(name)] [WAIT] [INVITE|LAST] [CNOTCOMPL|DEFRESP] Condition: SIGNAL
CONVERSE FROM(data-area) FROMLENGTH(data-value) [INTO(data-area)|SET(ptr-ref)] [TOLENGTH(data-area)] [LDC(name)|FMH] [DEST(name)] [DEFRESP] Conditions: EOC, EODS, INBFMH, LENGERR, SIGNAL
FREE [SESSION(name)] Conditions: INVREQ, NOTALLOC, SESSIONERR
WAIT SIGNAL Condition: SIGNAL
ISSUE DISCONNECT Condition: SIGNAL

LOGICAL DEVICE CODE (LDC OPTION)

A logical device code (LDC) is a code that can be included in an outbound FMH to specify the disposition of the data (for example, to which subsystem terminal it should be sent). Each code can be represented by a unique LDC mnemonic. The installation can specify up to 256 two-character mnemonics for each TCTTE, and two or more TCTTEs can share a list of these mnemonics. Corresponding to each LDC mnemonic for each TCTTE is a numeric value (0 through 255). A 3600 device and a logical page size are also associated with each LDC. "LDC" or "LDC value" is used in this publication in reference to the code specified by the user. "LDC mnemonic" refers to the

two-character symbol that represents the LDC numeric value.

When the LDC option is specified in the SEND command, the numeric value associated with the mnemonic for the particular TCTTE, is inserted in the FMH. The numeric value associated with the LDC mnemonic is chosen by the installation, and is interpreted by the 3601 application program.

3600 (3614) LOGICAL UNIT

```
RECEIVE {INTO(data-area)|
 SET(ptr-ref)}
LENGTH(data-area)

Condition:  LENGERR
```

```
SEND FROM(data-area)
LENGTH(data-value)
[DEST(name)]
[WAIT]
[INVITE|LAST]
[CNOTCOMPL|DEFRESP]
```

```
CONVERSE FROM(data-area)
FROMLENGTH(data-value)
[INTO(data-area)|SET(ptr-ref)]
[TOLENGTH(data-area)]
[DEFRESP(name)]
[DEST(name)]

Condition:  LENGERR
```

```
FREE [SESSION(name)]

Conditions:  INVREQ, NOTALLOC,
SESSIONERR
```

```
ISSUE DISCONNECT
```

The data stream and communication format used between a CICS/VS application program and a 3614 is determined by the 3614. The application program is therefore device dependent when handling 3614 communications.

For further information about designing 3614 application programs for CICS/VS, refer to the CICS/VS 3600 Guide.

3630 PLANT COMMUNICATION SYSTEM

Support and command syntax as for the 3600 (3601) logical unit and the 3600 pipeline logical unit as described earlier in this chapter for the 3600 Finance Communication System.

3650/3680 HOST COMMAND PROCESSOR LOGICAL UNIT

```
RECEIVE {INTO(data-area)|
 SET(ptr-ref)}
LENGTH(data-area)

Conditions:  EOC, LENGERR
```

```
SEND FROM(data-area)
LENGTH(data-value)
[WAIT]
[INVITE|LAST]
[CNOTCOMPL|DEFRESP]
[FMH]
```

```
CONVERSE FROM(data-area)
FROMLENGTH(data-value)
[INTO(data-area)|SET(ptr-ref)]
[TOLENGTH(data-area)]
[FMH]
[DEFRESP]

Condition:  LENGERR
```

```
FREE [SESSION(name)]

Conditions:  INVREQ, NOTALLOC,
SESSIONERR
```

```
ISSUE DISCONNECT
```

3650 HOST CONVERSATIONAL (3270) LOGICAL UNIT

```
RECEIVE {INTO(data-area)|
 SET(ptr-ref)}
LENGTH(data-area)

Conditions:  EOC, LENGERR
```

```
SEND FROM(data-area)
LENGTH(data-value)
[CTLCHAR(data-value)]
[WAIT]
[ERASE]
[INVITE|LAST]
[CNOTCOMPL|DEFRESP]
[FMH]
```

```
CONVERSE FROM(data-area)
FROMLENGTH(data-value)
[INTO(data-area)|SET(ptr-ref)]
[TOLENGTH(data-area)]
[CTLCHAR(data-value)]
[ERASE]
[DEFRESP]
[FMH]

Condition:  LENGERR
```

```
FREE [SESSION(name)]

Conditions:  INVREQ, NOTALLOC,
SESSIONERR
```

```
ISSUE PRINT
```

```
ISSUE ERASEAUP [WAIT]
```

```
ISSUE DISCONNECT
```

3650 HOST CONVERSATIONAL (3653) LOGICAL UNIT

```
RECEIVE {INTO(data-area)|
 SET(ptr-ref)}
   LENGTH(data-area)

Conditions:  EOC, LENGERR
```

```
SEND FROM(data-area)
LENGTH(data-value)
[WAIT]
[INVITE|LAST]
[CNOTCOMPL|DEFRESP]
```

```
CONVERSE FROM(data-area)
FROMLENGTH(data-value)
[INTO(data-area)|SET(ptr-ref)]
[TOLENGTH(data-area)]
[DEFRESP]

Conditions:  EOC, LENGERR
```

```
FREE [SESSION(name)]

Conditions:  INVREQ, NOTALLOC,
SESSIONERR
```

```
ISSUE DISCONNECT
```

3650 INTERPRETER LOGICAL UNIT

```
RECEIVE {INTO(data-area)|
 SET(ptr-ref)}
LENGTH(data-area)

Conditions:  EOC, EODS, INBFMH,
LENGERR
```

```
SEND FROM(data-area)
LENGTH(data-value)
[WAIT]
[INVITE|LAST]
[DEFRESP]
[FMH]
```

```
CONVERSE FROM(data-area)
FROMLENGTH(data-value)
[INTO(data-area)|SET(ptr-ref)]
[TOLENGTH(data-area)]
[DEFRESP]
[FMH]

Conditions:  EOC, EODS, INBFMH,
LENGERR
```

```
FREE [SESSION(name)]

Conditions:  INVREQ, NOTALLOC,
SESSIONERR
```

```
ISSUE LOAD PROGRAM(name)
[CONVERSE]

Conditions:  NONVAL, NOSTART
```

```
ISSUE EODS
```

```
ISSUE DISCONNECT
```

The ISSUE LOAD command specifies the name of the 3650 application program that is to be loaded.

The ISSUE EODS command can be used to send an end-of-data-set function management header to the 3650.

3650 PIPELINE LOGICAL UNIT

Support and command syntax as for the 3600 Pipeline Logical Unit.

3650/3680 FULL FUNCTION LOGICAL UNIT

Support and command syntax as for the 3790 Full Function Logical Unit.

3660 SUPERMARKET SCANNING SYSTEM

Support and command syntax as for System/3.

3735 PROGRAMMABLE BUFFERED TERMINAL

```
RECEIVE {INTO(data-area)|
 SET(ptr-ref)}
LENGTH(data-area)

Conditions:  EOF (not TCAM), LENGERR
```

```
SEND FROM(data-area)
LENGTH(data-value)
[DEST(name)]
[WAIT]
[ASIS]
```

```
CONVERSE FROM(data-area)
FROMLENGTH(data-value)
[INTO(data-area)|SET(ptr-ref)]
[TOLENGTH(data-area)]
[DEST(name)]

Conditions:  EOF (not TCAM), LENGERR
```

```
ISSUE RESET
ISSUE DISCONNECT
```

The 3735 Programmable Buffered Terminal may be serviced by CICS/VS in response to terminal-initiated input, or as a result of an automatic or time-initiated transaction. Both are explained below.

3735 TRANSACTIONS - AUTOANSWER

The 3735 transaction is attached by CICS/VS upon receipt of input from a 3735. Data is passed to the application program in 476-byte blocks; each block (one buffer) may contain several logical records. The final block may be shorter than 476 bytes; zero-length final blocks are not, however, passed to the application program. If the block contains several logical records, the application program must perform any necessary deblocking and gathering of partial logical records.

It is recommended that input data from a 3735 be spooled to an intermediate data set (for example, an intrapartition destination) to ensure that all data has been captured before deblocking and processing that data.

The application program must follow 3735 conventions and read to end-of-file before attempting to write FDPs (form description programs) or data to the 3735. For this reason, the application program must include a HANDLE CONDITION command for the EOF condition. When control passes to the EOF routine, FDPs or data may be written to the 3735, or, optionally, CICS/VS requested to disconnect the line.

The 3735 may transmit the EOF condition immediately upon connection of the line, in which case, a HANDLE CONDITION command for the EOF condition must be issued before any other terminal control commands.

The application program must format all special message headers for output to the 3735 (for example, SELECTRIC, POWERDOWN). If FDPs are to be transmitted to a 3735 with ASCII transmission code, the ASIS option must be included in the SEND command for each block of FDP records.

An ISSUE DISCONNECT command must be issued when all output has been transmitted to the 3735. If the application program ends during batch write mode before the ISSUE DISCONNECT command is executed, CICS/VS forces a 3735 "receive abort" condition and all data just transmitted is ignored by the 3735.

3735 TRANSACTIONS - AUTOCALL AND TIME-INITIATED

In automatic and time-initiated transactions, all considerations stated above apply when CICS/VS dials a 3735, except that the EOF condition cannot occur.

CICS/VS connects the line and allows the first terminal control command to indicate the direction of data transfer. If this first command is a SEND and the 3735 has data to send, the 3735 causes the line to be disconnected.

3740 DATA ENTRY SYSTEM

```
RECEIVE {INTO(data-area)|
 SET(ptr-ref)}
LENGTH(data-area)

Conditions:  EOF (except TCAM),
ENDINPT (except TCAM), LENGERR
```

```
SEND FROM(data-area)
LENGTH(data-value)
[DEST(name)]
[WAIT]
[ASIS]
```

```
CONVERSE FROM(data-area)
FROMLENGTH(data-value)
[INTO(data-area)|SET(ptr-ref)]
[TOLENGTH(data-area)]
[DEST(name)]

Condition:  LENGERR
```

```
ISSUE ENDFILE [ENDOUTPUT]
```

```
ISSUE ENDOUTPUT [ENDFILE]
```

```
ISSUE RESET
ISSUE DISCONNECT
```

BATCH MODE APPLICATIONS

In batch mode, many files are exchanged between the 3740 and CICS/VS in a single transmission. The transmission of an input batch must be complete before an output transmission can be started.

On input, the EOF (end-of-file) condition is raised by CICS/VS when a null block (indicating the end of a physical file) is received from the 3740. A HANDLE CONDITION EOF command should be included to specify that processing of the file is to continue. Eventually, the ENDINPUT condition is raised by CICS/VS when all input has been received. No more RECEIVE commands will be executed and a HANDLE CONDITION ENDINPUT command should be included to specify that control is to be returned to CICS/VS so that the 3740 can be set to receive data.

On output, the ISSUE ENDFILE and ISSUE ENDOUTPUT commands are used to indicate the end-of-file and end-of-output conditions, respectively, to the 3740. These two conditions may be specified in one command if required, for example: ISSUE ENDFILE ENDOUTPUT.

3767 INTERACTIVE LOGICAL UNIT

```
RECEIVE {INTO(data-area)|
 SET(ptr-ref)}
LENGTH(data-area)

Conditions:  EOC, LENGERR, SIGNAL
```

```
SEND FROM(data-area)
LENGTH(data-value)
[DEST(name)]
[WAIT]
[INVITE|LAST]
[CNOTCOMPL|DEFRESP]

Condition:  SIGNAL
```

```
CONVERSE FROM(data-area)
FROMLENGTH(data-value)
[INTO(data-area)|SET(ptr-ref)]
[TOLENGTH(data-area)]
[DEST(name)]
[DEFRESP]

Conditions:  EOC, LENGERR, SIGNAL
```

```
FREE [SESSION(name)]

Conditions:  INVREQ, NOTALLOC,
SESSIONERR
```

```
WAIT SIGNAL

Condition:  SIGNAL
```

```
ISSUE DISCONNECT

Condition:  SIGNAL
```

3770 BATCH LOGICAL UNIT

```
RECEIVE {INTO(data-area)|
 SET(ptr-ref)}
LENGTH(data-area)

Conditions:  EOC, EODS, INBFMH,
LENGERR, SIGNAL
```

```
SEND FROM(data-area)
LENGTH(data-value)
[DEST(name)]
[WAIT]
[INVITE|LAST]
[CNOTCOMPL|DEFRESP]
[FMH]

Condition:  SIGNAL
```

```
CONVERSE FROM(data-area)
FROMLENGTH(data-value)
[INTO(data-area)|SET(ptr-ref)]
[TOLENGTH(data-area)]
[DEST(name)]
[DEFRESP]
[FMH]

Conditions:  EOC, EODS, INBFMH,
LENGERR, SIGNAL
```

```
FREE [SESSION(name)]

Conditions:  INVREQ, NOTALLOC,
SESSIONERR
```

```
WAIT SIGNAL

Condition:  SIGNAL
```

```
ISSUE DISCONNECT

Condition:  SIGNAL
```

3770 INTERACTIVE LOGICAL UNIT

Support and command syntax as for 3767 Interactive Logical Unit.

3770 FULL FUNCTION LOGICAL UNIT

Support and command syntax as for 3790 Full Function Logical Unit.

3780 COMMUNICATIONS TERMINAL

Support and command syntax as for System/3.

3790 FULL FUNCTION LOGICAL UNIT

```
RECEIVE {INTO(data-area)|
 SET(ptr-ref)}
LENGTH(data-area)

Conditions:  EOC, EODS, INBFMH,
LENGERR, SIGNAL
```

```
SEND FROM(data-area)
LENGTH(data-value)
[DEST(name)]
[WAIT]
[INVITE|LAST]
[CNOTCOMPL|DEFRESP]
[FMH]

Condition:  SIGNAL
```

```
CONVERSE FROM(data-area)
FROMLENGTH(data-value)
[INTO(data-area)|SET(ptr-ref)]
[TOLENGTH(data-area)]
[DEST(name)]
[FMH]
[DEFRESP]

Conditions:  EOC, EODS, INBFMH,
LENGERR, SIGNAL
```

```
FREE [SESSION(name)]

Conditions:  INVREQ, NOTALLOC,
SESSIONERR
```

```
WAIT SIGNAL

Condition:  SIGNAL
```

```
ISSUE DISCONNECT

Condition:  SIGNAL
```

3790 INQUIRY LOGICAL UNIT

```
RECEIVE {INTO(data-area)|
 SET(ptr-ref)}
LENGTH(data-area)

Conditions:  EOC, EODS, INBFMH,
LENGERR
```

```
SEND FROM(data-area)
LENGTH(data-value)
[DEST(name)]
[WAIT]
[INVITE|LAST]
[CNOTCOMPL|DEFRESP]
[FMH]
```

```
CONVERSE FROM(data-area)
FROMLENGTH(data-value)
[INTO(data-area)|SET(ptr-ref)]
[TOLENGTH(data-area)]
[DEST(name)]
[FMH]
[DEFRESP]

Conditions:  EOC, EODS, INBFMH,
LENGERR
```

```
FREE [SESSION(name)]

Conditions:  INVREQ, NOTALLOC,
SESSIONERR
```

```
ISSUE DISCONNECT
```

3790 SCS PRINTER LOGICAL UNIT

```
SEND FROM(data-area)
LENGTH(data-value)
[DEST(name)]
[WAIT]
[INVITE|LAST]
[CNOTCOMPL|DEFRESP]
[DEFRESP]
```

```
FREE [SESSION(name)]

Conditions:  INVREQ, NOTALLOC,
SESSIONERR
```

```
ISSUE DISCONNECT
```

3790 (3270-DISPLAY) LOGICAL UNIT

```
RECEIVE {INTO(data-area)|
 SET(ptr-ref)}
LENGTH(data-area)
[ASIS]
[BUFFER]

Condition:  LENGERR
```

```
SEND FROM(data-area)
LENGTH(data-value)
[DEST(name)]
[CTLCHAR(data-value)]
[WAIT]
[ERASE]
[INVITE|LAST]
[DEFRESP]
```

```
CONVERSE FROM(data-area)
FROMLENGTH(data-value)
[INTO(data-area)|SET(ptr-ref)]
[TOLENGTH(data-area)]
[DEST(name)]
[DEFRESP]
[CTLCHAR(data-value)]
[ERASE]

Condition:  LENGERR
```

```
FREE [SESSION(name)]

Conditions:  INVREQ, NOTALLOC,
SESSIONERR
```

```
ISSUE PRINT
```

```
ISSUE ERASEAUP [WAIT]
```

```
ISSUE DISCONNECT
```

3790 (3270-PRINTER) LOGICAL UNIT

```
SEND FROM(data-area)
LENGTH(data-value)
[CTLCHAR(data-value)]
[WAIT]
[ERASE]
[INVITE|LAST]
[DEFRESP]
```

```
FREE [SESSION(name)]

Conditions:  INVREQ, NOTALLOC,
SESSIONERR
```

```
ISSUE PRINT
```

```
ISSUE ERASEAUP [WAIT]
```

```
ISSUE DISCONNECT
```

7770 AUDIO RESPONSE UNIT

```
RECEIVE {INTO(data-area)|
 SET(ptr-ref)}
LENGTH(data-area)

Condition:  LENGERR
```

```
SEND FROM(data-area)
LENGTH(data-value)
[DEST(name)]
[WAIT]
```

```
CONVERSE FROM(data-area)
FROMLENGTH(data-value)
[INTO(data-area)|SET(ptr-ref)]
[TOLENGTH(data-area)]

Condition:  LENGERR
```

```
ISSUE RESET
ISSUE DISCONNECT
```

CICS/VS cannot distinguish between special codes (characters) entered at audio terminals (for example, the 2721 Portable Audio Terminal); however, an application program can make use of these codes. The special codes that can be entered from a 2721 are as follows:

Key	Code(hex)
CALL END	37
CNCL	18
#	3B or 7B
VERIFY	2D
RPT	3D
EXEC	26
F1	B1
F2	B2
F3	B3
F4	B4
F5	B5
00	A0
00	3B or B0
IDENT	11, 12, 13, or 14 plus two other characters

For further information concerning the 2721, see the publication IBM 2721 Portable Audio Terminal Component Description.

The special codes A0 and 3B (or B0) are also generated by the keys * and # respectively of a "Touch-Tone" telephone. (Touch-Tone is the trademark of the American Telephone and Telegraph Company.)

If the SET option has been specified in the associated command, codes 26, 37, and 3B (each of which causes a hardware interrupt) will immediately follow the data, but will not be included in the value set by the LENGTH option.

If the end-of-inquiry (EOI) Disable Feature (Feature No. 3540) is installed on the 7770 Model 3, the option of including either or both # and 000 as data is available.

If, after receiving at least one code from a terminal, no other codes have been received by the 7770 for a period of five seconds, the 7770 generates an EOI hardware interrupt that ends the operation.

TERMINAL CONTROL OPTIONS

ASIS
: For System/370, System/7, 2770, 2780, and 3740: indicates that output is to be sent in transparent mode (with no recognition of control characters and accepting any of the 256 possible combinations of eight bits as valid transmittable data).

  For System/7: indicates that the data being written or read is not to be translated.

  For 3735: prevents translation of the Form Description Program (FDP) records that are to be transmitted to a 3735 using ASCII code.

  For 3270 and VTAM terminals: specifies a temporary override of the uppercase translation feature of CICS/VS to allow the current task to receive a message containing both uppercase and lowercase data.

  This option has no effect on the first RECEIVE command of a transaction, as terminal control will perform a read initial and use the terminal defaults to translate the data.

  This option has no effect if the screen contains data prior to a transaction being initiated. This data will be read and translated in preparation for the next task and the first RECEIVE command in that task will retrieve the translated data.

ATTACHID(name)
: specifies, for a BUILD ATTACH command, that the set of values specified is to be placed in an attach header control block identified by the specified name (maximum of eight characters).

  specifies, for a SEND or CONVERSE command, that an attach header (created by a BUILD ATTACH command) is to precede, and be concatenated with, the user data supplied in the FROM option. "Name" (maximum of eight characters) identifies the attach header control block to be used in the local task.

  specifies, for an EXTRACT ATTACH command, that values are to be retrieved from an attach header control block. "Name" (maximum of eight characters) identifies this control block to the local task. If the option is omitted, the attach header control block to be used is that associated with the facility named in the SESSION option.

BUFFER
: specifies that the contents of the 3270 buffer are to be read, beginning at buffer location one and continuing until all contents of the buffer have been read. All character and attribute sequences (including nulls) appear in the input data stream in the same order that they appear in the 3270 buffer.

CBUFF
: specifies that data is to be written to a common buffer in a 2972 Control Unit. The WAIT option is implied.

CNOTCOMPL
: indicates that the request/response unit (RU) sent as a result of this SEND command will not complete the chain. If this option is omitted

and chain assembly has been specified, the RU will terminate the chain.

CONVERSE
specifies that the 3650 application program will communicate with the host CPU. If this option is not specified, the 3650 application program cannot communicate with the host CPU.

CTLCHAR(data-value)
specifies a one-byte Write Control Character (WCC) that controls a SEND command, or the Copy Control Character (CCC) that controls an ISSUE COPY command, for a 3270. An COBOL user must specify a data area containing this character. If the option is omitted from a SEND command, all modified data tags are reset to zero and the keyboard is restored. If the option is omitted from an ISSUE COPY command, the contents of the entire buffer (including nulls) are copied.

DATASTR{(name)|(data-area)}
This corresponds to the data stream profile field, ATTDSP, in an attach FMH.

For communication between two CICS/VS systems, no particular significance is attached by CICS/VS to the data stream profile field in an attach FMH. For most CICS/VS applications, the option may be omitted when a value of "user defined" will be assumed.

For communication between a CICS/VS system and another subsystem, refer to documentation supplied by the subsystem on how to use the data stream profile field in an attach FMH.

When EIBATT is set during execution of a RECEIVE or CONVERSE command, the EXTRACT ATTACH command may be used to examine the data stream profile field received in the attach FMH.

The value is halfword binary; only the low-order byte is used. If this option is omitted, "user defined" is assumed. The bits in the binary value are used as follows:

0-7	reserved - must be set to zero
8-11	0000 - user defined 1111 - SCS datastream 1110 - 3270 datastream 1101 - structured field 1100 - logical record management
12-15	defined by the user if bits 8-11 are set to 0000; otherwise reserved (must be set to zero).

A value of "structured field" indicates that chains begin with four bytes of data that are used to interpret the following data; the four bytes consist of overall length (2 bytes), class identifier (1 byte), and sub-class identifier (1 byte). A value of "logical record management"indicates that chains can be split into separate fields by the data receiver.

These values may be used for communication between a CICS/VS system and another subsystem; for further details of structured fields and logical record management refer to the documentation supplied by the subsystem.

If the option is omitted from the BUILD ATTACH command, a value of "user defined" is assumed.

DEFRESP
indicates that a definite response is required when the output operation has been completed.

DEST(name)
specifies the four-byte symbolic name of the TCAM destination to which the message is to be sent. This option is meaningful only for terminals for which DEVICE=TCAM has been specified in the DFHTCT TYPE=SDSCI system macro.

ERASE
specifies that the screen is to be erased and the cursor returned to the upper left corner of the screen before writing occurs. Normally, ERASE should be specified in the first output command of a transaction. This will clear the screen ready for the new output data.

However, when switching from one screen size to another on a transaction basis, it is important to note that if ERASE is not specified in the first output command of the transaction, the screen size will be unchanged from its previous setting, that is, the previous transaction setting, or the default screen size if the CLEAR key has been pressed.

FMH
specifies that a function management header has been included in the data that is to be written. If the ATTACHID option is specified as well, the concatenated FMH flag will be set in the attach FMH .

FROM(data-area)
specifies the data that is to be written to the terminal or logical unit. This option may be omitted if ATTACHID is specified.

FROMLENGTH(data-value)
See LENGTH(parameter). The FROMLENGTH option of the CONVERSE command is equivalent to the LENGTH option of a SEND command.

INTO(data-area)
specifies the receiving field for the data read from the terminal or logical unit.

INVITE
specifies that the next terminal control command to be executed for this facility is a RECEIVE. This allows optimal flows to occur.

IUTYPE{(name)|(data-area)}
This corresponds to the interchange unit field, ATTIU, in an attach FMH.

For communication between two CICS/VS systems, no particular significance is attached by CICS/VS to the interchange unit field in an attach FMH. For most CICS/VS applications, the option may be omitted when a value of "multiple chain" will be assumed.

For communication between a CICS/VS system and another subsystem, refer to documentation supplied by the subsystem on how to use the interchange unit field in an attach FMH.

When EIBATT is set during execution of a RECEIVE or CONVERSE command, the EXTRACT ATTACH command may be used to examine the interchange unit field received in the attach FMH.

The value is halfword binary; only the low-order 7 bits being used. The bits in the binary value are used as follows:

```
0-10   reserved - must be set to zero
11     0 - not end of multichain
           interchange unit
       1 - end of multichain
           interchange unit
12-13  reserved - must be set to zero
14-15  00 - multichain interchange
           unit
       01 - single chain interchange
           unit
       10 - reserved
       11 - reserved
```

If the option is omitted from the BUILD ATTACH command, values of "not end of multichain interchange unit" and "multiple chain" are assumed.

LAST
specifies that this is the last output operation for a transaction and therefore the end of a bracket.

LDC(name)
specifies the two-character mnemonic used to determine the appropriate logical device code (LDC) numeric value. The mnemonic represents an LDC entry in the DFHTCT TYPE=LDC macro instruction.

LEAVEKB
specifies that the keyboard is to remain locked at the completion of the data transfer. This option is applicable only to CICS/OS/VS but may be used in a CICS/DOS/VS application program if compatibility is required.

LENGTH(parameter)
specifies the length (as a halfword binary value) of the data transmitted by RECEIVE and SEND commands.

For a RECEIVE command with the INTO option, the parameter must be a data area that specifies the maximum length that the program will 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. When the data has been received, the data area is set to the original length of the data.

For a RECEIVE command with the SET option, the parameter must be a data area. When the data has been received, the data area is set to the length of the data.

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

LINEADDR(data-value)
specifies that the writing is to begin on a specific line of a 2260/2265 screen. The data value is a halfword binary value in the range 1 through 12 for a 2260, or 1 through 15 for a 2265.

NETNAME(name)
specifies the eight-character name of the logical unit in the VTAM network.

PASSBK
specifies that communication is with a passbook at a 2980. The WAIT option is implied.

PROCESS{(name)|(data area)}
This corresponds to the process name, ATTDPN, in an attach FMH.

For communication between two CICS/VS systems, a transaction running in one system can acquire a session to the second system and can identify the transaction to be attached in the second system; the identification is carried in the first chain of data sent across the session.

In general, the first four bytes of data will identify the transaction to be attached. However an attach FMH, identifying the transaction to be attached, may be built and sent; the PROCESS option on the BUILD ATTACH command is used to specify the transaction name. (Note that the receiving CICS/VS system will use just the first four bytes of the process name as a transaction name).

No significance is attached by CICS/VS to process names in attach FMHs sent in chains of data other than the first.

For communication between a CICS/VS system and another subsystem, refer to documentation supplied by the sybsystem on how to use the process name field in an attach FMH.

When EIBATT is set during execution of a RECEIVE or CONVERSE command, the EXTRACT ATTACH command may be used to examine the process name received in the attach FMH.

PROFILE(name)
specifies the name (maximum of eight characters) of a set of terminal control processing options, held in the PCT, that are to be used during execution of terminal control commands for the session specified in the SYSID or SESSION options. If this option is omitted, a set of processing options, called DFHCICSA, will be selected.

PROGRAM(name)
specifies the name (maximum of eight characters) of the 3600 application program that is to be loaded.

PSEUDOBIN
specifies that the data being written or read is to be translated from System/7 pseudobinary representation to hexadecimal on a RECEIVE command or from hexadecimal to pseudobinary on a SEND command.

QUEUE{(name)|(data-area)}
This corresponds to the queue name, ATTDQN, in an attach FMH.

For communication between two CICS/VS systems, no significance is attached by CICS/VS to the queue name in an attach FMH.

For communication between a CICS/VS system and another subsystem, refer to documentation supplied by the subsystem on how to use the queue name field in an attach FMH.

When EIBATT is set during execution of a RECEIVE or CONVERSE command, the EXTRACT ATTACH command may be used to examine the queue name received in the attach FMH.

RECFM{(name)|(data area)}
This corresponds to the deblocking algorithm field, ATTDBA, in an attach FMH.

For communication between two CICS/VS systems, no particular significance is attached by CICS/VS to the deblocking algorithm field in an attach FMH. For most CICS/VS applications, the option may be omitted when a value of "chain of RUs" will be assumed.

For communication between a CICS/VS system and another subsystem, refer to documentation supplied by the subsystem on how to use the deblocking algorithm field in an attach FMH.

When EIBATT is set during execution of a RECEIVE or CONVERSE command, the EXTRACT ATTACH command may be used to examine the deblocking algorithm field received in the attach FMH.

The value is halfword binary; only the low-order 8 bits being used. The bits in the binary value are used as follows:

```
0-7  reserved - must be set to zero
8-15 X'00' - reserved
     X'01' - variable length variable
             blocked
     X'02' - reserved
     X'03' - reserved
     X'04' - chain of RUs
     X'05' to X'FF' - reserved
```

If the option is omitted from the BUILD ATTACH command, a value of "chain of RUs" is assumed.

RESOURCE{(name)|(data-area)}
This corresponds to the resource name, ATTPRN, in an attach FMH.

For communication between two CICS/VS systems, no significance is attached by CICS/VS to the resource name in an attach FMH.

For communication between a CICS/VS system and another subsystem, refer to documentation supplied by the subsystem on how to use the resource name field in an attach FMH.

When EIBATT is set during execution of a RECEIVE or CONVERSE command, the EXTRACT ATTACH command may be used to examine the resource name received in the attach FMH.

RPROCESS{(name)|(data-area)}
This corresponds to the return process name, ATTRDPN, in an attach FMH.

For communication between two CICS/VS systems, no significance is attached by CICS/VS to the return process name in an attach FMH.

For communication between a CICS/VS system and another subsystem, refer to documentation supplied by the sybsystem on how to use the return process name field in an attach FMH.

When EIBATT is set during execution of a RECEIVE or CONVERSE command, the EXTRACT ATTACH command may be used to examine the return process name received in the attach FMH.

RRESOURCE{(name)|(data-area)}
This corresponds to the return resource name, ATTRPRN, in an attach FMH.

For communication between two CICS/VS systems, no significance is attached by CICS/VS to the return resource name in an attach FMH.

For communication between a CICS/VS system and another subsystem, refer to documentation supplied by the subsystem on how to use the return resource name field in an attach FMH.

When EIBATT is set during execution of a RECEIVE or CONVERSE command, the EXTRACT ATTACH command may be used to examine the return resource name received in the attach FMH.

SESSION(name)
specifies the symbolic identifier (maximum of four characters) of a session TCTTE. This option specifies the alternate session to be used. If this option is omitted, the principal facility for the task will be used.

SET(ptr-ref)
specifies the pointer reference that is to be set to the address of the data read from the terminal or logical unit.

STRFIELD
specifies that the data area specified in the FROM option contains structured fields. If this option is specified, the contents of all structured fields must be handled by the application program. The CONVERSE command, rather than a SEND command, must be used if the data area contains a read partition structured field. (Structured fields are described in the CICS/VS IBM 3270 Guide.) CTLCHAR and ERASE are mutually exclusive with STRFIELD, and their use with STRFIELD will generate an error message.

SYSID{(name)|(data-area)}
specifies the name (maximum of four characters) of a system TCTSE. This option specifies that one of the sessions to the named system is to be allocated.

When used with the EXTRACT TCT command, this option specifies the variable to be set to the equivalent local name of the system.

TERMID{(name)|(data-area)}
specifies the name (up to four characters in length) of the terminal whose buffer is to be copied. The terminal must have been defined in the TCT.

When used with the EXTRACT TCT command this option specifies the variable to be set to the equivalent local name of the terminal.

TOLENGTH(data-area)
See LENGTH(parameter). The TOLENGTH option of the CONVERSE command is equivalent to the LENGTH option of a RECEIVE command.

WAIT
specifies that processing of the command must be completed before any subsequent processing is attempted.

If the WAIT option is not specified, control is returned to the application program once processing of the command has started. A subsequent input or output request (terminal control, BMS, or batch data interchange) to the terminal associated with the task will cause the application program to wait until the previous request has been completed.

TERMINAL CONTROL EXCEPTIONAL CONDITIONS

Some of the following exceptional conditions may occur in combination with others. CICS/VS checks for these conditions in the following order: EODS, INBFMH, EOC. If more than one of these conditions occurs, only the first one found to be present is passed to the application program.

CBIDERR
occurs if the named set of terminal-control processing options

cannot be found.

Default action: terminate the task abnormally.

ENDINPT
occurs when an end-of-input indicator is received.

Default action: terminate the task abnormally.

EOC
occurs when a request/response unit (RU) is received with the end-of-chain indicator set. Field EIBEOC also contains this indicator.

Default action: ignore the condition.

EODS
occurs when an end-of-data-set indicator is received.

EOF
occurs when an end-of-file indicator is received.

Default action: terminate the task abnormally.

IGREQCD
occurs when an attempt is made to execute a SEND or CONVERSE command after a SIGNAL data-flow control command with an RCD (request change direction) code has been received from an LUTYPE4 logical unit.

Default action: terminate the task abnormally.

INBFMH
occurs if a request/response unit (RU) contains a function management header (FMH). Field EIBFMH contains this indicator and it should be used in preference to INBFMH. The IGNORE CONDITION command can be used to ignore the condition.

Default action: terminate the task abnormally.

INVREQ
occurs, for the EXTRACT TCT command, if the name specified in the NETNAME option cannot be found.

LENGERR
occurs if the length of data received in response to a command that specifies the INTO option, exceeds the value specified by the LENGTH or TOLENGTH option.

Default action: terminate the task abnormally.

NONVAL
occurs if a 3650 application program name is invalid.

Default action: terminate the task abnormally.

NOPASSBKRD
occurs if no passbook is present on an input operation.

NOPASSBKWR
occurs if no passbook is present on an output operation.

NOSTART
occurs if the 3651 is unable to initiate the requested 3650 application program.

Default action: terminate the task abnormally.

NOTALLOC
occurs if the facility specified in the command is not owned by the application.

Default action: terminate the task abnormally.

RDATT
occurs if a RECEIVE command is terminated by the attention (ATTN) key rather than the return key.

Default action: ignore the condition.

SESSBUSY
occurs if the request for a session cannot be serviced immediately.

Default action: queue the request until a session is available.

SESSIONERR
occurs if the name specified in the SESSION option is not that of a session TCTTE or if the session cannot be allocated because it is out of service.

Default action: terminate the task abnormally.

SIGNAL
occurs when an inbound SIGNAL data-flow control command is received from a logical unit or session. It is raised by execution of the next SEND, RECEIVE, or WAIT TERMINAL command that refers to the logical unit or session. It is raised also by execution of a WAIT SIGNAL command, in which case the data-flow control command has been received from the principal facility.

Default action: ignore the condition.

SYSBUSY
occurs if the request for a session cannot be serviced immediately.

Default action: queue the request until a session is available.

SYSIDERR
occurs if the name in the SYSID option is not that of a system TCTTE, or if all sessions are out of service.

Default action: terminate the task abnormally.

TERMIDERR
occurs if the specified terminal identifier cannot be found in the terminal control table (TCT).

Default action: terminate the task abnormally.

WRBRK
occurs if a SEND command is terminated by the attention key.

Default action: ignore the condition.

Chapter 3.3. Basic Mapping Support (BMS)

CICS/VS basic mapping support is an interface, between an application program and the terminal control program, that provides various formatting services for interpreting input data streams and for preparing output data streams for the terminal network.

The application program passes data to BMS and receives data from BMS in a standard device independent format. BMS commands are included in the application program to control formatting of the data and to initiate input from a terminal or output to one or more terminals.

BMS commands are provided to:

- Map data into a data area in the program (RECEIVE MAP).
- Map, and possibly transmit, output data in field or block data format (SEND MAP).
- Build, and possibly transmit, output data in text data format (SEND TEXT).
- Complete and transmit a logical message (SEND PAGE).
- Delete an incomplete logical message (PURGE MESSAGE).
- Initiate building a logical message for delivery to one or more terminals (ROUTE).

All of these commands, with their associated options and exceptional conditions, are described in the last part of this chapter. Other sections describe how combinations of the commands can be used to control output operations and discuss features shared by the commands.

BMS input and output commands result in terminal control commands. However, both terminal control and BMS commands can be included in an application program. An operation to map a data stream already in storage, rather than receiving and mapping, may be requested to cause BMS to map a device-dependent input data stream. If a map operation is requested for input from the non-formatted 3270 buffer, mapping is not performed; the non-formatted data stream is returned to the application program, and the MAPFAIL exceptional condition occurs.

The HANDLE CONDITION and IGNORE CONDITION commands, and the NOHANDLE option, can be used to deal with any exceptional conditions that occur during the execution of BMS commands. Refer to Chapter 1.5 for further information about exceptional conditions.

Two principal advantages are obtained by using BMS: device independence and format independence.

DEVICE INDEPENDENCE

Device independence allows the application program to send data to a terminal or to receive data from a terminal without regard to the physical characteristics of the terminal.

Under BMS, the terminal may be any of the following devices: 1050, 2740, 2741, 2770, 2780, 2980 Models 1 and 2, 2980-4 (keyboard and printer only), 3270, 3780, TWX, tape, disk, CRLP (a device declared to have card-reader-in/line-printer-out characteristics), or terminals specified by the system programmer in the terminal control table (TCT) as TRMTYPE=TCAM.

Certain BMS facilities can also be used with some 3270, 3600, 3650, 3767, 3770, and 3790 logical units; for information about these logical units, refer to the appropriate CICS/VS subsystem guide. These guides are listed in the bibliography.

With BMS, a CICS/VS installation with more than one type of terminal need provide only one application program for each transaction to support all terminals in the installation. BMS identifies which type of terminal is requesting use of the application program and provides for the conversion of the device-dependent data stream to and from the standard device independent data format used by the application program. A CICS/VS installation using only one type of terminal may wish to use the formatting services of BMS to facilitate the addition of other types or the conversion to another type in the future.

FORMAT INDEPENDENCE

Format independence allows the application program to provide data to one or more terminals or to receive data from a terminal without regard to the placement of fields within the data stream or on the terminal.

All references to data by the application program are through symbolic field names. Fields are placed within the data stream by BMS according to information stored in data format tables called **maps**. A CICS/VS installation in which BMS is used

may rearrange the fields to be included in the data by simply changing the values stored in the map that defines the format of the data. The application program that causes the data to be written need not be modified. The programming maintenance requirements should be considerably less than they might be if BMS were not used.

Format independence also allows information such as headings, field-identifying keywords, and 3270 screen formats to be stored in maps. This information can be modified simply by changing its value in the maps. Programs that refer to the maps benefit from the changes, but none of the programs themselves need be modified.

The format independence provided by BMS removes from the application program the requirement to know the location of fields within the data stream; fields may be rearranged, removed, or added without changing the application program.

DATA MAPPING

Data mapping is the technique used by BMS to convert the standard device-independent 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.

There are three standard formats in which the application program can provide or accept BMS data, as follows:

Field Data Format: data is passed to BMS as separate fields. Each field is given a symbolic name, which is used when passing data to, or retrieving data from, BMS. Each field consists of a two-byte length area (used by BMS on input), a one-byte attribute area (for 3270 output operations), and the data area. A map is used to describe the field position, data length, and other necessary information.

Block Data Format: data is passed to BMS as line segments. Fields positioned within the line segments may be given symbolic names to aid the application program in positioning the fields. Each field provides for a one-byte attribute and the field data area. A gap consisting of several blanks may separate consecutive fields in the line segment. A map is used to describe the number and lengths of line segments, the field position, data length, and other necessary information.

Text Data Format: output data is passed to BMS as a data stream which is divided into lines no longer than those defined for the terminal to which the data stream is related. Printable character strings or words which overlap lines are placed unbroken on the next available line. New-line (X'15') characters can be included in the data stream to further define line lengths. CICS/VS inserts the appropriate leading characters, carrier returns, and idle characters, and eliminates trailing blanks from each line. If tab control characters are contained in the data stream, the user should also supply all of the necessary new-line characters. No maps are used with text data format.

Field data format is the most common for both display and printer terminals.

Block data format may be used with both display and printer terminals, but it is more useful for input operations on printer terminals.

Text data format is used with both display and printer terminals and is especially convenient for handling data not divided into fields. When text data format is used with a 3270 device, an attribute byte appears on the 3270 as a blank at the beginning of each line and in front of each new piece of data. When the data is destined for a device with extended attributes, set attribute (SA) orders can be included also in the data stream. These orders enable characters in the data stream to be modified by the extended attributes. To aid this modification, symbolic names are available in DFHBMSCA (the standard attribute list). The standard attribute list is described in Chapter 3.2.

MAP DEFINITION

Most of the facilities of BMS (text data format is the exception) require two types of maps to be defined by CICS/VS macro instructions and to be assembled offline prior to running the application program. The two types are:

1. **Physical map** - used by BMS to convert data to or from the format required by the application program. The map is a table containing information about each field; it is stored in the CICS/VS program library and is loaded by BMS at execution time.

2. **Symbolic description map** - used by the application program to refer to the data in storage. This map is a set of source statements that are cataloged into the appropriate source library and copied into the application program when it is assembled or compiled.

All maps must be generated as members of a **map set**; a single map must be generated as the only member of such a map set. A map set is a collection of related maps

that are generated and stored together in the CICS/VS libraries. A reference to any map in a map set requires that the entire map set be loaded into storage for the duration of the task or until another map set is referred to by the task.

An alternative method of defining maps for use with BMS is by means of SDF/CICS (Screen Definition Facility (CICS)). The unload facility of SDF/CICS converts the stored form of the BMS map into a form acceptable to the Interactive Map Definition component of DPPX/DPS. For more information refer to the SDF/CICS Program Reference Manual.

The following macro instructions are used in the map-definition process.

DFHMSD macro

- defines a map set
- specifies that a set of macros is for a physical map or for a symbolic description map
- specifies that the map is for input, output, or both
- specifies that the data format is either field or block.

DFHMDI macro

- defines a map within a map set
- specifies the position of the map on the page, either absolutely or in relation to other maps
- specifies the size of the map
- specifies that the data format is either field or block.

DFHMDF macro

- defines a field within a map
- specifies the position of the field
- specifies the length of the field.

The formats of these macros and an example of their use and of the symbolic descriptions maps generated is given later in the chapter. The macros follow the normal coding conventions for assembler-language macros; in particular, note that if the comma preceding an operand is omitted, that operand and all succeeding operands for the associated macro will be treated as a comment, without notification.

An operand in a DFHMDF macro will always override the same operand in a DFHMDI macro. Similarly, an operand in a DFHMDI macro will always override the same operand in a DFHMSD macro.

If an operand is omitted from a DFHMDF macro, the same operand, if present, in the DFHMDI macro will be used. Similarly, if an operand is omitted from both the DFHMDF or DFHMDI macros, that in the DFHMSD macro will be used.

If an operand is omitted from all the macros used to define a map set, the default values for the DFHMDF macro will be assumed.

Some facilities, such as color, are available only on certain terminals, and a specification for such a facility will be ignored if the terminal does not support it. This obviates the requirement to define separate maps for different terminals.

The map definition macros are assembled twice, once to produce the physical map used by BMS, and once to produce the symbolic description map (or DSECT) that will be copied into the application program.

Examples of map definition are included in the sample programs in the appendixes.

INPUT MAPPING

For an input map, the starting position and the maximum data length of each field must be defined, as follows:

The TIOA symbolic storage definition contains an area for the length of each input data field, followed by a flag byte and an area for the data itself. Space is reserved for the maximum number of bytes defined for each field.

The program can access the length, flag, and data areas of any field by symbolic labels. The length area is a halfword binary field and is addressed by the name "fieldnameL" or "groupnameL". The flag is a one-byte field and is addressed by the name "fieldnameF" or "groupnameF". The data portion of each field (or group of fields) is contiguous with the length and flag areas. A group of fields, or a single field not within any group of fields, has one data portion addressed by the name "groupnameI" or "fieldnameI". For fields contained within a group, there are no intervening length or flag areas (only "groupnameL" exists) but each field is addressed by a name "fieldnameI".

In assembler-language programs, the first byte of the first occurrence of a field defined by the DFHMDF operand OCCURS=n (where n is greater than 1) is named "fieldnameD", and the first byte of the next occurrence of the field is named "fieldnameN". These names refer to the first byte of the length area if DATA=FIELD is specified, and to the first

byte of the attribute data if DATA=BLOCK is specified.

In COBOL and PL/I programs, "fieldnameD" is the name of the array of minor structures containing the length, flag, and data areas of the field.

The number of characters entered may differ from the length of the field at program execution time. If more data is keyed than specified in the map, the data is truncated on the right to the number of characters specified. The length that is returned to the application program is the truncated length. If less data is keyed than specified, the remaining character positions are filled with blanks or zeros and the length of the keyed data is returned in the length field.

The flag byte is normally set to X'00'. However, if the field has been modified but no data has been sent (as, for example, if it has been modified to nulls), the flag byte is set to X'80' and the length area is set to zeros.

Fields that are entered as input but are not defined in the map are discarded. The length and data areas of fields defined but not keyed are set to nulls (X'00').

For a light pen-detectable field, although no data is passed, a single data byte is reserved. This byte contains X'FF' if the field is selected or X'00' if the field is not selected. The length area of a light pen-detectable field contains a binary one if selected or a binary zero if not selected.

OUTPUT MAPPING

For an output map, the starting position, length, field characteristics, and default data (if desired) must be defined, as follows:

The fields of an output map are assigned names in the DFHMDF macro. The characteristic or attribute byte is named "fieldnameA" or "groupnameA". For a field contained within a group, the data area is given the name "fieldnameO", but there is no separate attribute byte for the field. (Only the group name has the attribute byte.) For a group name, or a field not contained within a group, the data area is given the name "groupnameO" or "fieldnameO."

In assembler-language programs, the first byte of the first occurrence of a field defined by OCCURS=n (where n is greater than 1) is named "fieldnameD", and the first byte of the next occurrence of the field is named "fieldnameN". These names refer to the first byte of the length area if DATA=FIELD is specified, and to the first byte of the attribute data if DATA=BLOCK is specified.

In COBOL and PL/I programs, "fieldnameD" is the name of the array of minor structures containing the attribute byte and data area of the field, together with the unused two-byte length field (described below). A field not contained within a group is treated as a group containing one field entry. An unused two-byte length field precedes each attribute byte and data field to provide a format similar to an input symbolic storage description TIOA.

The TIOAPFX=YES operand must be specified in the DFHMSD or DFHMDI macros that create the maps. Also, if the symbolic description maps are referred to by a PL/I program, the STORAGE=AUTO operand must be specified in the DFHMSD macro.

When defining fields, the user may provide a name for any field that he wishes to refer to at execution time. Such names are associated with the fields in the symbolic storage definition of the TIOA to allow symbolic references to be made to them. The user may specify not only the characteristics of the field but also the default data to be written as output for a field when no data is supplied for that field by an application program. This facility permits the specification of titles, headers, and so forth, for output maps. The user may temporarily override the field characteristics, the data, or both field characteristics and data of any field for which a name has been specified. The desired changes are simply inserted into the TIOA under the specified field name in the symbolic storage definition (symbolic description map) in the program.

Output field data supplied by the application program must not begin with a null character (X'00'), or the entire field will be ignored by BMS. A suitable character to use in the first position is blank (X'40').

Light pen-detectable fields should be "autoskip" to prevent data from being keyed into them. Because of the nature of these fields, in most instances, they should not be modified. If the data field is modified, the application program must ensure that the first character is a "?", ">", "&", or a blank character; otherwise, the field is no longer light pen-detectable.

Fields that can be keyed should be delimited by a stopper field to ensure that all the data keyed and transmitted can be mapped.

INPUT/OUTPUT MAPPING

Input/output maps combining all the functions of input and output maps can also be created using the DFHMSD, DFHMDI, and DFHMDF macros.

The number of fields which can be specified for a COBOL or PL/I input/output map is limited to 1023.

MAP RETRIEVAL

Map sets placed in the CICS/VS program library are accessed by BMS through program control LOAD commands. Each map set name must have been entered in the processing program table (PPT) by the system programmer. When device-dependent map sets are placed in the CICS/VS program library, they must be identified by the device-dependent suffixed name, and a corresponding entry of the same name must appear in the PPT. (Device-dependent suffixes are described below under the "mapset" name of the DFHMSD macro and under the SUFFIX and TERM operands of that macro.)

OUTBOARD FORMATTING

Outboard formatting is a facility that can be used in a telecommunication network to offload some of the display presentation work normally performed at the host to another node. DPS Version 2 supports outboard formatting on an 8100 running DPPX and communicating with a System/370 host running CICS/VS. Outboard formatting is also supported on a 3650 (for details refer to the CICS/VS 3650/3680 Guide).

The support is designed primarily for use by an application program running under CICS/VS basic mapping support (BMS) at the host, and interacting with a 3650 or a terminal at the 8100 node. Instead of input and output mapping being performed wholly by BMS at the host, it is performed in part by either the 3650 or by DPS at the 8100. Some of the resulting advantages are that:

1. The number of processing cycles at the host can be reduced.
2. The line traffic between the two systems can be reduced.
3. On an 8100 the BMS application program can take advantage of DPS/8100 facilities; it can make use of DPS exits, for example, and device features that are supported by DPS but not BMS.

For detailed information on DPS Version 2 outboard formatting support, including the procedure for setting up an outboard formatting system, see the DPPX/DPS Version 2 System Programming Guide.

DEFINE A MAP SET (DFHMSD MACRO)

The syntax of the DFHMSD macro used to define a map set is shown in Figure 14 on page 130. The macro specifies whether physical maps (TYPE=MAP) or symbolic description maps (TYPE=DSECT) are to be generated. The end of a map set is indicated always by a DFHMSD TYPE=FINAL macro.

Alternatively, both types of map can be assembled in the same job by job control language, as described in the CICS/VS System Programmer's Guide.

The operands are defined as follows:

mapset
is the name of the map set. The name (1 through 7 characters) must begin with an alphabetic character. A suffix specified by the SUFFIX operand, or based on the terminal type specified in the TERM operand, is added during assembly.

This suffixed name is the name that should be used in the NAME statement (OS) or the PHASE statement (DOS) in cataloging the map set (see the appropriate CICS/VS System Programmer's Guide for further details), and the name that should be specified in the PPT (see the CICS/VS System Programmer's Reference Manual). Valid suffixes are shown in the description of the TERM operand, below.

When a mapping operation is requested by a BMS command, CICS/VS adds a similar suffix to the map set name specified in the command, and attempts to load a map set with the suffixed name. If the suffixed map set name cannot be found in the library, CICS/VS will load a map set with the specified name (equivalent to being suffixed with a blank).

CICS/VS obtains the suffix from the TCTTE for the terminal (either the terminal associated with the transaction or, for routing, the destination terminal) depending on the terminal type specified in the TRMTYPE operand (together with the SESTYPE operand for VTAM terminals) of the DFHTCT TYPE=TERMINAL (or TYPE=LINE) system macro. If the alternate page size is being used, as specified by the ALTPGE operand of the DFHTCT TYPE=TERMINAL system macro, and the ALTSFX operand of that same system macro has also been specified, an attempt will be made to load the map set that has the alternate suffix specified in the

mapset	DFHMSD	TYPE={DSECT|MAP} [,BASE=name] [,COLOR={DEFAULT|BLUE|RED|PINK|GREEN|TURQUOISE|YELLOW|NEUTRAL}] [,CTRL=([PRINT][,{L40|L64|L80|HONEOM}][,FREEKB][,ALARM][,FRSET])] [,DATA={FIELD|BLOCK}] [,EXTATT={NO|MAPONLY|YES}] [,HILIGHT={OFF|BLINK|REVERSE|UNDERLINE}] [,HTAB=tab[,tab]...] [,LANG={ASM|COBOL|PLI|RPG}] [,LDC=mnemonic] [,MODE={IN|OUT|INOUT}] [,OBFMT={YES|NO}] [,PS={BASE|psid}] [,STORAGE=AUTO] [,SUFFIX=n] [,TERM=terminal-type] [,TIOAPFX={YES|NO}] [,VALIDN=([MUSTFILL][,MUSTENTER])] [,VTAB=tab[,tab]...]
mapset	DFHMSD	TYPE=FINAL

Figure 14. DFHMSD Macro (Define a Map Set)

ALTSFX operand of the DFHTCT TYPE=TERMINAL system macro. If this load is unsuccessful, normal map set selection will occur.

For example, if two map sets are assembled, one with TERM=CRLP and the other with TERM=ALL, the first map set name will be suffixed with A and the second with blank. The system programmer should use these suffixed names in the NAME/PHASE statements and in the PPT. If a CICS/VS transaction now routes a message to two terminals, one of which has TRMTYPE=CRLP and the other TRMTYPE=L3277,TRMMODL=2, CICS/VS will attempt to load "mapsetA" for the first and "mapsetM" for the second. The second of these will be unsuccessful, so BMS will then look for the unsuffixed map set name for routing to the 3277.

TYPE=
specifies the function of the macro.

DSECT
specifies that a symbolic description map is to be generated. If the same map set is to be used by application programs written in different languages, a separate DFHMSD TYPE=DSECT macro must be written for each language to put the symbolic description map into the copy library of the language.

MAP
specifies that a physical map is to be generated. This physical map is stored in the CICS/VS program library and loaded as required by BMS. The assembler-language application programmer can, alternatively, generate the map in his program and pass its address to BMS.

FINAL
must be coded to indicate the end of the map set. If other parameters are specified in this macro, they will be ignored.

BASE=name
specifies that the same storage base will be used for the symbolic description maps from more than one map set. The same name is specified 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, different maps within the same map set will also overlay one another.

This operand is not valid for assembler-language programs.

For example, assume that the following macros are used to generate symbolic description maps for two map sets.

```
MAPSET1 DFHMSD TYPE=DSECT,
               TERM=2780,LANG=COBOL,
               BASE=DATAREA1,
               MODE=IN

MAPSET2 DFHMSD TYPE=DSECT,
               TERM=3270,LANG=COBOL,
               BASE=DATAREA1,
               MODE=OUT
```

The symbolic description maps of this example might be referred to in a COBOL application program as follows:

```
LINKAGE SECTION.
01 DFHBLLDS COPY DFHBLLDS.
   .
   .
   02 TIOABAR PIC S9(8) COMP.
   02 MAPBASE1 PIC S9(8) COMP.
   .
   .

01 DFHTIOA COPY DFHTIOA.
01 DATAREA1 PIC X(1920).
01 name COPY MAPSET1.
01 name COPY MAPSET2.
```

MAPSET1 and MAPSET2 both redefine DATAREA1; only one 02 statement is needed to establish addressability. However, the program can only use the fields in one of the symbolic description maps at a time.

If BASE=DATAREA1 is deleted from this example, an additional 02 statement is needed to establish addressability for MAPSET2; the 01 DATAREA1 statement is not needed. The program could then refer to fields concurrently in both symbolic description maps.

In PL/I application programs, the name specified in the BASE operand is used as the name of the pointer variable on which the symbolic description map is based. If this operand is omitted, the default name (BMSMAPBR) is used for the pointer variable. The PL/I programmer is responsible for establishing addressability for the based structures.

COLOR=
specifies the default color for all fields in all maps in a map set unless overridden explicitly by the COLOR option of a DFHMDI or DFHMDF macro. If this option is specified when EXTATT=NO, a warning will be issued and the option ignored. If this option is specified, but EXTATT is not, EXTATT=MAPONLY will be assumed.

CTRL=
specifies device characteristics related to terminals of the 3270 Information Display System. CTRL=ALARM is valid for TCAM 3270 SDLC and VTAM-supported terminals (except interactive and batch logical units); all other parameters for CTRL are ignored. This operand must be specified on the last (or only) map of a page unless the options of a BMS command are being used to override the corresponding operand in the DFHMSD macro. If the CTRL operand is specified in the DFHMDI macro, it cannot be specified in the DFHMSD macro.

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 3270 displays without the Printer Adapter feature.

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 default printer line length to be used.

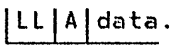
FREEKB
specifies that the keyboard should be unlocked after the map is written out. If omitted, the keyboard remains locked; further data entry from the keyboard is inhibited until this status is changed.

ALARM
activates the 3270 audible alarm feature. For other VTAM terminals it sets the alarm flag in the FMH; this feature is not supported by interactive and batch logical units.

FRSET
specifies 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 map data is written to the buffer. This allows the DFHMDF macro with the ATTRIB operand to control the final status of any fields written or rewritten in response to a BMS command.

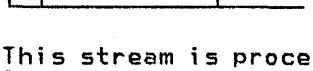
DATA=
specifies the format of the data.

FIELD
specifies that the data is passed as contiguous fields where each field has the following format:



"LL" is two bytes specifying the length of the data as input from the terminal (these two bytes are ignored in output processing). "A" is a byte into which the programmer may place an attribute to override that specified in the map used to process this data (see "Standard Attribute List and Printer Control Characters (DFHBMSCA), in "Chapter 3.2. Terminal Control" on page 85).

BLOCK
specifies that the data is passed as a continuous stream in the following format:



This stream is processed as line segments of the length specified in the map used to process the data set. The data is in the form that it appears on the terminal; that is, it contains data fields and interspersed blanks corresponding to any spaces that are to appear between the fields on output. The first byte of each line is the attribute byte; it is not available for data. EXTATT=YES cannot be used if DATA=BLOCK is specified.

EXTATT=
specifies whether the extended attributes (COLOR, HILIGHT, PS, and VALIDN) are supported.

NO
specifies that the extended attributes are not supported; the physical and symbolic description maps will be the same as those generated under Version 1 Release 4. "NO" is the default unless COLOR, HILIGHT, PS, or VALIDN is specified in the DFHMSD macro, in which case EXTATT=MAPONLY will be assumed. If the TERM operand is specified and is other than 3270, 3270-1, 3270-2, or ALL, EXTATT=MAPONLY or EXTATT=YES will be invalid, and the COLOR, HILIGHT, PS, and VALIDN operands on the DFHMSD, DFHMDI, or DFHMDF macros will be invalid.

MAPONLY
specifies that the extended attributes can be specified in a map, but that the resulting symbolic description map will contain no fields for them, and that it will be the same as one generated under Version 1, Release 4. This operand can be used to add the extended attributes to an existing map without recompiling.

YES
specifies that the extended attributes can be specified in a map, and that they can be modified dynamically. The symbolic description map (DSECT) will contain subfields for the attributes, identified by suffixes C (for COLOR), H (for HILIGHT), P (for PS), and V (for validation).

HILIGHT=
specifies the default highlighting attribute for all fields in all maps in a map set.

OFF
is the default and means that no highlighting is used.

BLINK
specifies that the field is to "blink" at a set frequency.

REVERSE
specifies that the character or field is displayed in "reverse video", for example, on a 3278, black characters on a green background.

UNDERLINE
specifies that a field is underlined.

If this option is specified when EXTATT=NO, a warning will be issued and the option ignored. If this option is specified, but EXTATT is not, EXTATT=MAPONLY will be assumed.

HTAB=tab[,tab]...
specifies one or more tab positions for use with interactive and batch logical units having horizontal forms control.

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
specifies that the symbolic description map is to be referred to by an assembler-language program.

COBOL
specifies that the symbolic description map is to be referred to by a COBOL program.

PLI
specifies that the symbolic description map is to be referred to by a PL/I program.

RPG
specifies that the symbolic description map is to be referred to by an RPG II program. This parameter is valid for CICS/DOS/VS only.

LDC=mnemonic
specifies the mnemonic to be used by CICS/VS to determine the logical device code that is to be used for a BMS output operation and transmitted in the function management header to the logical unit if no LDC operand has been specified on any previous BMS output in the logical message. This operand is used only for TCAM and VTAM-supported 3600 terminals, and batch logical units.

MODE=

IN
specifies an input map generation.

OUT
specifies an output map generation.

INOUT
specifies that the map definition is to be used for both input and output mapping operations.

Input mapping is not available for VTAM-supported 3600 terminals. However, INOUT may be specified for map generation. The map can then be used as a dummy input map for input operations using the RECEIVE MAP command.

OBFMT=
specifies whether outboard formatting is to be used. This operand is available only for 3650 logical units and for 8100 Information Systems using the DPPX/DPS Version 2 acting as a logical unit for a real display (for example, 3277, 3278, 3279, or 8775). If a logical unit does not support outboard formatting this operand will be ignored at execution time. Refer to the CICS/VS 3650/3680 Guide, or the DPPX/Distributed Presention Services Version 2 Systems Programming Guide for more details.

YES
specifies that all maps within this map set can be used in outboard formatting, except those for which OBFMT=NO is specified in the DFHMDI macro.

NO
specifies that no maps within this map set can be used in outboard formatting, except those for which OBFMT=YES is specified in the DFHMDI macro.

PS=
specifies that programmed symbols are to be used.

BASE
specifies that only the basic symbols are used.

psid
specifies a single EBCDIC character or a hexadecimal code of the form X'nn', that identifies the set of programmed symbols.

If this option is specified when EXTATT=NO, a warning will be issued and the option ignored. If this option is specified, but EXTATT is not, EXTATT=MAPONLY will be assumed.

STORAGE=AUTO
specifies, for assembler-language programs, that separate maps within a map set are to occupy separate storage, not to overlay one another.

specifies, for COBOL programs, that the symbolic description maps in the map set are to be in separate (that is, not redefined) areas. This operand is used when the symbolic description maps are copied into the WORKING-STORAGE section and the storage for the separate maps in the map set is to be used concurrently.

specifies, for PL/I programs, that the symbolic description maps are to be declared as having the AUTOMATIC storage class. If not specified, they are declared as BASED.

If STORAGE=AUTO is specified, BASE=name cannot be used. If STORAGE=AUTO is specified and TIOAPFX is not specified, TIOAPFX=YES is assumed.

SUFFIX=n
specifies a one-character map set suffix that overrides any suffix implied by the TERM operand. A message will indicate that the TERM operand has been ignored. The user should catalog the map set, with this suffixed name, in the program library, and ensure also that there is no conflict with a generated name of another version of the map set. The use of numeric suffixes would help prevent conflict.

TERM=terminal type
specifies the type of terminal or logical unit associated with the map set. If no terminal type is specified, 3270 is assumed.

TERM=	Suffix
CRLP	A
TAPE	B
DISK	C
TWX	D
1050	E
2740	F
2741	G
2770	I
2780	J
3780	K
3270-1 (40-col display)	L
3270-2 (80-col display)	M
INTLU|3767|3770I|SCS[1]	P
2980	Q
2980-4	R
3270	blank
3601	U
3653[2]	V
3650UP[3]	W
3650/3270[4]	X
BCHLU|3770B[5]	Y
ALL	blank

[1] Use also for all interactive LUs, 3790 full function LU, and SCS-printer LUs (3270 and 3790)

[2] Use for host conversational (3653) LU

[3] Use for interpreter LU

[4] Use for host conversational (3270) LU

[5] Use also for all batch and batch data interchange LUs.

For TCAM-connected terminals (other than 3270 or SNA devices), use either CRLP or ALL; for TCAM-connected 3270s or SNA devices, select the appropriate parameter in the normal way.

If ALL is specified, ensure that device-dependent characters are not included in the map set and that format characteristics such as page size are suitable for all input/output operations (and all terminals) in which the map set will be applied. For example, some terminals are limited to 480 bytes, others to 1920 bytes; the 3604 is limited to six lines of 40 characters each. Within these guidelines, use of ALL can offer important advantages. Since an assembly run is required for each map generation, the use of ALL, indicating that one map is to be used for more than one terminal, can result in significant time and storage savings.

However, better run-time performance for maps used by single terminal types will be achieved if the terminal type (rather than ALL) is specified. Alternatively, BMS support for device-dependent map sets can be bypassed by specifying BMSDDS=NO in the DFHSG PROGRAM=BMS system macro. (See the CICS/VS System Programmer's Reference Manual for further details.)

TIOAPFX=
specifies whether BMS should include a filler in the symbolic description maps to allow for the unused TIOA prefix.

YES
specifies that the filler should be included in the symbolic description maps. If TIOAPFX=YES is specified, all maps within the map set have the filler, except when TIOAPFX=NO is specified on the DFHMDI macro. TIOAPFX=YES should always be used for command-level application programs.

NO
is the default and specifies that the filler is not to be included. The filler may still be included for a map if TIOAPFX=YES is specified on the DFHMDI macro.

VALIDN=

MUSTFILL
specifies that the field must be filled completely with data. An attempt to move the cursor from the field before it has been filled, or to transmit data from an incomplete field, will raise the inhibit input condition.

MUSTENTER
specifies that data must be entered into the field. An attempt to move the cursor from an empty field will raise the inhibit input condition.

VTAB=tab[,tab]...
specifies one or more tab positions for use with interactive and batch logical units having vertical forms control.

DEFINE A MAP (DFHMDI MACRO)

The syntax of the DFHMDI macro to define a map is shown in Figure 15 on page 136. It defines the size of the data to be mapped and its position within the input or output. When defining more than one map, the corresponding number of DFHMDI macros must be used.

If the maps are for use in a COBOL program, and STORAGE=AUTO has been specified in the DFHMSD macro, they must be specified in descending size sequence (size refers to the generated 01 level data areas and not to the size of the map on the screen).

The operands are defined as follows:

map
is the name (1 through 7 characters) of the map.

COLOR=
specifies the default color for all fields in a map unless overridden explicitly by the COLOR option of a DFHMDF macro. If this option is specified when EXTATT=NO is specified in the associated DFHMSD macro, a warning will be issued and the option ignored.

COLUMN=
specifies the column in a line at which the map is to be placed, that is, it establishes the left or right map margin. The JUSTIFY operand controls whether map and page margin selection and column counting are to be from the left or right side of the page. The columns between the specified map margin and the page margin are not available for subsequent use on the page for any lines included in the map.

number
is the column from the left or right page margin where the left or right map margin is to be established.

NEXT
indicates that the left or right map margin is to be placed in the next available column from the left or right on the current line.

SAME
indicates that the left or right map margin is to be established in the same column as the last non-header or non-trailer map used that specified COLUMN=number and the same JUSTIFY parameters as this macro.

Refer to the section "Map Positioning," later in this chapter, for a more detailed discussion.

CTRL=
specifies device characteristics related to terminals of the 3270 Information Display System. CTRL=ALARM is valid for TCAM 3270 SDLC and VTAM-supported terminals (except interactive and batch logical units); all other parameters for CTRL are ignored. This operand must be specified on the last (or only) map of a page unless options of a BMS command are being used to override the corresponding operands in the DFHMSD macro. If the CTRL operand is specified in the DFHMDI macro, it cannot be specified in the DFHMSD macro.

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 3270 displays without the Printer Adapter feature.

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 default printer line length to be used.

FREEKB
specifies that the keyboard should be unlocked after the map is written out. If omitted, the keyboard remains locked; further data entry from the keyboard is inhibited until this status is changed.

ALARM
activates the 3270 audible alarm. For other VTAM terminals it sets the alarm flag in the FMH; this feature is not supported by interactive and batch logical units.

FRSET
specifies that the modified data tags (MDTs) of all fields currently in the 3270 buffer are to be reset to a not-modified condition (that

```
map     DFHMDI   [,COLOR={DEFAULT|BLUE|RED|PINK|GREEN|TURQUOISE|YELLOW|NEUTRAL}
                 [,COLUMN={number|NEXT|SAME}]
                 [,CTRL=([PRINT][,{L40|L64|L80|HONEOM}][,FREEKB][,ALARM][,FRSET])]
                 [,DATA={FIELD|BLOCK}]
                 [,HEADER=YES]
                 [,HILIGHT={OFF|BLINK|REVERSE|UNDERLINE}]
                 [,JUSTIFY=([{LEFT|RIGHT}][,{FIRST|LAST}])]
                 [,LINE={number|NEXT|SAME}]
                 [,OBFMT={YES|NO}]
                 [,PS={BASE|psid}]
                 [,SIZE=(line,column)]
                 [,TIOAPFX={YES|NO}]
                 [,TRAILER=YES]
                 [,VALIDN=([MUSTFILL][,MUSTENTER])]
```

Figure 15. DFHMDI Macro (Define a Map)

is, field reset) before map data is written to the buffer. This allows the DFHMDF macro with the ATTRIB operand to control the final status of any fields written or rewritten in response to a BMS command.

DATA=
specifies the format of the data.

FIELD
specifies that the data is passed as contiguous fields in the following format:

```
|LL|A|data...
```

"LL" is two bytes specifying the length of the data as input from the terminal (these two bytes are ignored in output processing). "A" is a byte into which the programmer may place an attribute to override that specified in the map used to process this data (see "Standard Attribute List and Printer Control Characters (DFHBMSCA), in "Chapter 3.2. Terminal Control" on page 85).

BLOCK
specifies that the data is passed as a continuous stream in the following format:

```
|A|data field|space...
```

This stream is processed as line segments of the length specified in the map used to process the data set. The data is in the form that it appears on the terminal; that is, it contains data fields and interspersed blanks corresponding to any spaces that are to appear between the fields on output. The first byte of each line is the attribute byte; it is not available for data.

HEADER=YES
allows the map to be used during page building without terminating the overflow condition (see "Overflow Processing," later in this chapter). This operand may be specified for more than one map in a map set.

HILIGHT=
specifies the default highlighting attribute for all fields in a map.

OFF
is the default and means that no highlighting is used.

BLINK
specifies that the field is to "blink" at a set frequency.

REVERSE
specifies that the field is displayed in "reverse video", for example, on a 3278, black characters on a green background.

UNDERLINE
specifies that a field is underlined.

If this option is specified when EXTATT=NO is specified in the associated DFHMSD macro, a warning will be issued and the option ignored. If this option is specified, but EXTATT is not, EXTATT=MAPONLY will be assumed.

JUSTIFY=
specifies the margins on a page in which a map is to be formatted.

LEFT
indicates that the map is to be positioned starting at the specified column from the left margin on the specified line.

RIGHT
indicates that the map is to be positioned starting at the specified column from the right margin on the specified line.

FIRST
indicates that the map is to be positioned as the first map on a new page. Any partially formatted page from preceding BMS commands is considered to be complete. This operand can be specified for only one map per page.

LAST
indicates that the map is to be positioned at the bottom of the current page. This operand can be specified for multiple maps to be placed on one page. However, maps other than the first map for which it is specified must be able to be positioned horizontally without requiring that more lines be used.

LEFT and RIGHT are mutually exclusive, as are FIRST and LAST. If neither FIRST nor LAST is specified, the data is mapped at the next available position as determined by other parameters of the map definition and the current mapping operation. FIRST and LAST are ignored unless PAGEBLD is specified, since otherwise only one map is placed on each page.

Refer to the section "Map Positioning," later in this chapter, for a more detailed discussion.

LINE=
specifies the starting line on a page in which data for a map is to be formatted.

number
is a value from 1 to 240, specifying a starting line number. A request to map data on a line and column that has been formatted in response to a preceding BMS command causes the current page to be treated as though complete. The new data is formatted at the requested line and column on a new page.

NEXT
specifies that formatting of data is to begin on the next available completely empty line. If LINE=NEXT is specified in the DFHMDI macro, it is ignored for input operations and LINE=1 is assumed.

SAME
specifies that formatting of data is to begin on the same line as that used for a preceding BMS command. If the data does not fit on the same line, it is placed on the next available completely-empty line.

Refer to the section "Map Positioning," later in this chapter, for a more detailed discussion.

OBFMT=
specifies whether outboard formatting is to be used. This operand is available only for 3650 logical units and for 8100 Information Systems using the DPPX operating system with DPPX/DPS Version 2 acting as a logical unit for a real display (for example, 3277, 3278, 3279, or 8775). If a logical unit does not support outboard formatting this operand will be ignored at execution time. Refer to the CICS/VS 3650/3680 Guide, or the DPPX/Distributed Presentation Services Version 2 System Programming Guide for more details.

If omitted, the OBFMT operand in the DFHMSD macro is used.

YES
specifies that this map is to be used with outboard formatting.

NO
specifies that this map is not to be used with outboard formatting.

PS=
specifies that programmed symbols are to be used.

BASE
specifies that only the basic symbols are used.

psid
specifies a single EBCDIC character or a hexadecimal code of the form X'nn' that identifies the set of programmed symbols.

If this option is specified when EXTATT=NO is specified in the associated DFHMSD macro, a warning will be issued and the option ignored. If this option is specified, but EXTATT is not, EXTATT=MAPONLY will be assumed.

SIZE=
specifies the size of a map.

line
is a value from 1 to 240, specifying the depth of a map as a number of lines.

column
is a value from 1 to 240, specifying the width of a map as a number of columns. Space for the attribute byte should be included.

This operand is required in the following cases:

- An associated DFHMDF macro with the POS operand is used.
- The map is to be referred to in a SEND MAP command with the ACCUM option.
- The map is to be used when referring to input data from other than a 3270 terminal in a RECEIVE MAP command.

TIOAPFX=
specifies whether BMS should include a filler in the symbolic description maps to allow for the unused TIOA prefix. If omitted, the TIOAPFX operand on the DFHMSD macro is used.

YES
specifies that the filler should be included in the symbolic description map. TIOAPFX=YES should always be used for command level application programs.

NO
specifies that the filler is not to be included for this map.

TRAILER=YES
allows the map to be used during page building without terminating the overflow condition (see "Overflow Processing," later in this chapter). This operand may be specified for more than one map in a map set. If a trailer map is used other than in the overflow environment, the space normally reserved for overflow trailer maps is not reserved while mapping the trailer map.

VALIDN=

MUSTFILL
specifies that the field must be filled completely with data. An attempt to move the cursor from the field before it has been filled, or to transmit data from an incomplete field, will raise the inhibit input conditions.

MUSTENTER
specifies that data must be entered into the field. An attempt to move the cursor from an empty field will raise the inhibit input condition.

DEFINE A FIELD (DFHMDF MACRO)

The syntax of the DFHMDF macro to define a field is shown in Figure 16 on page 139. This macro is used to define a field. One DFHMDF macro is required for each field in a map, giving information such as symbolic field name, field position, field length, attribute byte (for 3270 terminals), initial constant data, justification of input, and COBOL or PL/I data picture. Two or more DFHMDF macros must be arranged in numerical order of the POS operand, except for output mapping operations using DATA=FIELD.

The number of named fields that can be defined for a COBOL or PL/I input/output map must not exceed 1023.

The operands are defined as follows:

fld
is the name (1 through 7 characters) of the field. Although a name is not required for every field within a map, a name must be specified for at least one field of a map to be compiled under COBOL or PL/I. All fields within a group must have names.

If name is omitted, an application program cannot access the field to change its attributes or alter its contents. 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 mapping operation, data supplied by the user overlays data supplied by initialization (unless default data only is being written).

POS=
specifies the location of a field. This operand specifies the individually addressable character

```
[fld]   DFHMDF   [,POS={number|(line,column)}]
                 [,ATTRB=([{ASKIP|PROT|UNPROT[,NUM]}][,{BRT|NORM|DRK}][,DET][,IC]
                 [,FSET])]
                 [,COLOR={DEFAULT|BLUE|RED|PINK|GREEN|TURQUOISE|YELLOW|NEUTRAL}]
                 [,GRPNAME=group-name]
                 [,HILIGHT={OFF|BLINK|REVERSE|UNDERLINE}]
                 [,INITIAL='character data'|XINIT=hexadecimal data]
                 [,JUSTIFY=([{LEFT|RIGHT}][,{BLANK|ZERO}])]
                 [,LENGTH=number]
                 [,OCCURS=number]
                 [,PICIN='value']
                 [,PICOUT='value']
                 [,PS={BASE|psid}]
                 [,VALIDN=([MUSTFILL][,MUSTENTER])]
```

Figure 16. DFHMDF Macro (Define a Field)

location in a map at which the attribute byte that precedes the field is positioned.

number
specifies the displacement (relative to zero) from the beginning of the map being defined.

(line,column)
specify lines and columns (relative to one) within the map being defined.

The location of data on the output medium is dependent on DFHMDI macro parameters as well.

The first position of a field is reserved for an attribute byte. When supplying data for input mapping from non-3270 devices, the input data must allow space for this attribute byte. Input data must not start in column 1 but may start in column 2.

The POS operand always contains the location of the first position in a field, which is normally the attribute byte when communicating with the 3270. For the second and subsequent fields of a group, the POS operand points to an assumed attribute-byte position, ahead of the start of the data, even though no actual attribute byte is necessary. If the fields follow on immediately from one another, the POS operand should point to the last character position in the previous field in the group.

When a position number is specified which represents the last character position in the 3270, two special rules apply:

- The IC attribute should not be coded. The cursor may be set to location zero by using the cursor option of the SEND MAP or SENT TEXT command.
- If the field is to be used in an output mapping operation with the DATA=ONLY specification, an attribute byte for that field must be supplied in the TIOA by the application program.

ATTRB=
is applicable only to fields to be displayed on a 3270 and 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. A group of fields appears as one field to the 3270. Therefore, the ATTRB specification refers to all of the fields in a group as one field rather than as individual fields. Refer to the publication IBM 3270 Information Display System Component Description for further information.

This operand applies only to 3270 data stream devices; it will be ignored for other devices, including the SCS Printer Logical Unit. It will also be ignored if the NLEOM option is specified on the SEND MAP command for transmission to a 3270 printer. In particular, ATTRB=DRK should not be used as a method of protecting secure data on output. It could however, be used for making an input field nondisplay for secure entry of a password from a screen.

For input map fields, DET and NUM are the only valid options; all others are ignored.

ASKIP
specifies that data cannot be keyed into the field and causes the cursor (current location pointer) to skip over the field.

PROT
specifies that data cannot be keyed into the field.

If data is to be copied from one device to another attached to the same 3270 control unit, the first position (address 0) in the buffer of the device to be copied from must not contain an attribute byte for a protected field. When preparing maps for 3270s, ensure that the first map of any page does not contain a protected field starting at position 0.

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 a high-intensity display of the field is required. By virtue of the 3270 attribute character bit assignments, a field specified as BRT is also potentially detectable. However, for the field to be recognized as detectable by BMS, DET must also be specified.

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.

The first character of a 3270 detectable field must be a "?", ">", "&", or blank. If the first character is "&" or blank, the field is an attention field; if the first character is "?" or ">", the field is a selection field. (See the publication IBM 3270 Information Display System Component Description for further details of detectable fields.)

A field for which BRT is specified is potentially detectable to the 3270, by virtue of the 3270 attribute character bit assignments, but is not recognized as such by BMS unless DET is also specified.

DET and DRK are mutually exclusive options.

If DET is specified for an input field, only one data byte is reserved for each input field. This byte is set to X'00', and remains unchanged if the field is not selected. If the field is selected the byte is set to X'FF'.

No other data is supplied, even if the field is a selection field and the ENTER key has been pressed.

If the data in a detectable field is required, all of the following conditions must be fulfilled:

1. The field must begin with either a "?" ">", or "&" and DET must be specified in the output map.
2. The ENTER key (or some other attention key) must be pressed after the field has been selected, although for detectable fields beginning with "&" the ENTER key is not required.
3. DET must not be specified for the field in the input map. DET must, however, be specified in the output map.

IC
specifies that the cursor is to be placed in the first position of the 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 zero. 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 of the SEND MAP or SEND TEXT command that causes the write operation.

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 an output mapping request (for example, DFHMSD CTRL=FRSET or DFHBMS CTRL=FRSET) causes the MDT to be reset.

Either of two sets of defaults may apply when a field to be displayed on a 3270 is being defined but 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.

COLOR=
specifies the colors to be used. If this option is specified when EXTATT=NO is specified in the associated DFHMSD macro, a warning will be issued and the option ignored.

GRPNAME=group-name
is the name (1 through 7 characters) used to generate symbolic storage definitions and to combine specific fields under one group name. The same group name must be specified for each field that is to belong to the group.

The fields in a group must follow on; there can be intervening gaps between them, but not other fields from outside 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 the fields follow each other. 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).

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 sum of the lengths of the fields within the group must not exceed 256 bytes. If this operand is specified, the OCCURS operand cannot be specified.

Examples showing the effect of this operand are included later in the chapter.

HILIGHT=
specifies the type of highlighting to be used.

OFF
is the default and means that no highlighting is used.

BLINK
specifies that the field is to "blink" at a set frequency.

REVERSE
specifies that the field is displayed in "reverse video", for example, on a 3278, black characters on a green background.

UNDERLINE
specifies that a field is underlined.

If this operand is specified when EXTATT=NO is specified in the associated DFHMSD macro, a warning will be issued and the operand ignored.

INITIAL='character data'|XINIT=hexadecimal data
specifies 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.

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 '.

If hexadecimal data is specified that corresponds with line or format control characters, the results will be unpredictable. The XINIT operand should therefore be used with care.

JUSTIFY=
specifies the field justifications for input operations. This operand is ignored for TCAM-supported 3600 and 3790, and for VTAM-supported 3600, 3650, and 3790 terminals, as input mapping is not available.

LEFT
specifies that data in the input field is left-justified.

RIGHT
specifies that data in the input field is right-justified.

BLANK
specifies that blanks are to be inserted in any unfilled positions in an input field.

ZERO
specifies that zeros are to be inserted in any unfilled positions in an input field.

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.

If a field is initialized by an output map or contains data from any other source, data that is keyed as input may not be justified and the additional data may remain in the field.

LENGTH=number
specifies the length (1 through 256 bytes) of the field. This specified length should be the maximum length required for application-program data to be entered into the field; it should not include the one-byte attribute indicator appended to the field by CICS/VS for use in subsequent processing. 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 map dimensions specified in the SIZE operand of the DFHMDI macro instruction defining a map may be smaller than the actual page size or screen size as defined for the terminal. The LENGTH specification in a DFHMDF macro instruction cannot cause the map-defined boundary on the same line to be exceeded. That is, the length declared for a field cannot exceed the number of positions available from the starting position of the field to the final position of the map-defined line. For example, given an 80-position page line, the following map definition and field definition are valid:

```
DFHMDI  SIZE=(2,40),...
DFHMDF  POS=22,LENGTH=17,...
```

but the following definitions are not acceptable:

```
DFHMDI  SIZE=(2,40),...
DFHMDF  POS=22,LENGTH=30,...
```

OCCURS=number
specifies that the indicated number of entries for the field are to be generated in a map and that the map definition is to be generated in such a way that the fields are addressable as entries in a matrix or an array. This permits several data fields to be addressed by the same name (subscripted) without generating a unique name for each field. OCCURS and GRPNAME are mutually exclusive; that is, OCCURS cannot be used when fields have been defined under a group name. If this operand is omitted, a value of 1 is assumed.

Examples showing the effect of the OCCURS operand are included later in the chapter.

PICIN='value'
specifies a picture to be applied to an input field in an IN or INOUT map; this picture serves as an editing specification which is passed to the application program, thus permitting the user to exploit the editing capabilities of COBOL or PL/I. The PICIN operand is not valid for assembler-language programs. BMS checks 'value' to ascertain that the specified characters are valid picture specification characters for the language of the map.

However, no validity checking of the input data is performed by BMS or the high-level language when the map is used, so any desired checking must be performed by the application program. The length of the data associated with 'value' should be the same as that specified in the LENGTH operand if LENGTH is specified. If both PICIN and PICOUT (see below) are used, an error message is produced if their calculated lengths do not agree; the shorter of the two lengths is used. If PICIN or PICOUT is not coded for the field definition, a character definition of the field is automatically generated regardless of other operands that are coded, such as ATTRB=NUM.

As an example, assume the following map definition is created for reference by a COBOL application program:

```
MAPX    DFHMSD  TYPE=DSECT,
                LANG=COBOL,
                MODE=INOUT
MAP     DFHMDI  LINE=1,
                COLUMN=1,
                SIZE=(1,80)
F1      DFHMDF  POS=0,LENGTH=30
F2      DFHMDF  POS=40,LENGTH=10,
                PICOUT='$$$,$$0.00'
F3      DFHMDF  POS=60,LENGTH=6,
                PICIN='9999V99',
                PICOUT='ZZ9.99'
        DFHMSD  TYPE=FINAL
```

The following DSECT is generated:

```
01  MAPI.
    02  F1L     PIC S9(4) COMP.
    02  F1A     PIC X.
    02  FILLER REDEFINES F1A.
     03  F1F    PIC X.
    02  F1I     PIC X(30).
    02  FILLER  PIC X.
    02  F2L     PIC S9(4) COMP.
    02  F2A     PIC X.
    02  FILLER REDEFINES F2A.
     03  F2F    PIC X.
    02  F2I     PIC X(10).
    02  FILLER  PIC X.
    02  F3L     PIC S9(4) COMP.
    02  F3A     PIC X.
    02  FILLER REDEFINES F3A.
     03  F3F    PIC X.
    02  F3I     PIC 9999V99.
    02  FILLER  PIC X.
01  MAPO REDEFINES MAPI.
    02  FILLER  PIC X(3).
    02  F1O     PIC X(30).
    02  FILLER  PIC X.
    02  FILLER  PIC X(3).
    02  F2O     PIC $$$,$$0.00.
    02  FILLER  PIC X.
    02  FILLER  PIC X(3).
    02  F3O     PIC ZZ9.99.
    02  FILLER  PIC X.
```

PICOUT='value'
is similar to PICIN, except that a picture to be applied to an output field in the OUT or INOUT map is generated.

Like PICIN, PICOUT is not valid for assembler-language programs.

PS=
specifies the programmed symbol set to be used for the display of the field.

BASE
specifies that only the basic symbols are used.

psid
specifies a single EBCDIC character or a hexadecimal code of the form X'nn' that identifies the set of programmed symbols.

If this option is specified when EXTATT=NO is specified in the associated DFHMDS macro, a warning will be issued and the option ignored. If this option is specified, but EXTATT is not, EXTATT=MAPONLY will be assumed.

VALIDN=

MUSTFILL
specifies that the field must be filled completely with data. An attempt to move the cursor from the field before it has been filled, or to transmit data from an incomplete field, will raise the inhibit input conditions.

MUSTENTER
specifies that data must be entered into the field. An attempt to move the cursor from an empty field will raise the inhibit input condition.

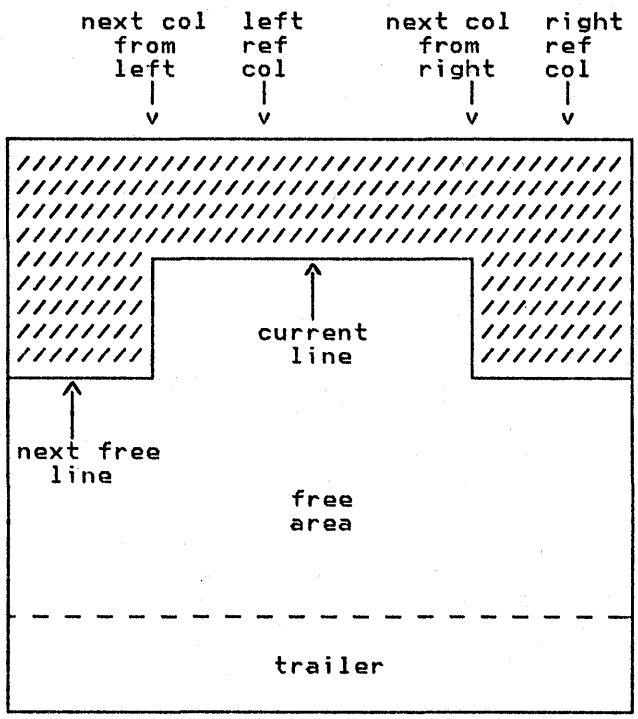
MAP POSITIONING

The position of a map on a screen is determined by two major factors: the current contents of the screen, and the values coded for the LINE, COLUMN, and JUSTIFY operands of the DFHMDI macro. Positioning is also affected if the DFHMDI macro specifies HEADER=YES or TRAILER=YES, and by the depth of the deepest trailer map in the map set.

THE SCREEN CONTENTS

At any instant, the part of the screen which is still available for maps is in the form of either an L, a reversed L, a

rectangle, or an inverted T, as shown by the unshaded area in the following diagram:



The shape and size of this area is represented internally by four variables: current line, next free line, next column from left, and next column from right.

Three other pointers are maintained that are relevant to map placement though they do not affect the area available: left reference column and right reference column, which are used when COLUMN=SAME is specified, and trailer size.

THE TRAILER AREA

The trailer size is equal to the number of lines that would be occupied by the deepest trailer map in the map set currently in use. It is determined when the map set is assembled, and is copied from the map set whenever one is loaded. The trailer size is assumed to be zero if there is no overflow routine.

The area defined by trailer size is not available for mapping unless no overflow routine has been specified or the map has TRAILER=YES specified in its DFHMDI macro.

JUSTIFY=FIRST AND JUSTIFY=LAST

If JUSTIFY=FIRST is specified, the map is placed on a new page, so that the only maps above it are the header maps used in overflow processing. The LINE operand may also be used with JUSTIFY=FIRST to place the map below the top of the page.

If JUSTIFY=LAST is specified, the map is placed as low as possible on the page. For a non-trailer map, this is immediately above the trailer area; for a trailer map, it is at the bottom of the page. In the absence of an overflow routine, the trailer area is null and JUSTIFY=LAST places the map at the bottom of the page .

A map defined with JUSTIFY=LAST cannot be used in input operations unless it was previously put out without the ACCUM option , in which case JUSTIFY=LAST is ignored and the map is positioned at the top of the page.

THE LINE OPERAND

The LINE operand specifies the line of the screen on which the first line of the map is to be placed. The initial determination of this line is made without regard to the specification of the COLUMN operand or the columns available on the screen on that particular line. If it transpires that the map will not fit on the chosen line, the first subsequent line that will satisfy the column requirements is selected.

If LINE=SAME or LINE=NEXT is specified, the initial line selected for the start of the map is the current line or the next free line , respectively. If a number is specified in the LINE operand, the line with that number is selected, provided the number is greater than or equal to the number of the current line; if not, the overflow condition is raised so that the map can be placed on the next page.

The line selected becomes the new current line and, if it is below the next free line, the next free line is reset to the same line; the next column from the left and right are also reset, to the left and right margins respectively.

If the line selected is such that the end of the map extends into the trailer area for a non-trailer map or beyond the end of the page for a trailer map, the overflow condition is raised and the map will be placed on the first available line of the next page when the request is reissued after handling the overflow.

THE COLUMN AND JUSTIFY OPERANDS

The COLUMN specification may be either NEXT, SAME, or a number and is processed in conjunction with the LEFT or RIGHT specification of the JUSTIFY operand. JUSTIFY=LEFT is the default and implies

that the column specification is related to the left-hand margin. Conversely, JUSTIFY=RIGHT implies that the column specification is related to the right-hand margin. For the purposes of this explanation, it is assumed hereafter that JUSTIFY=LEFT has been specified (or applied by default).

If COLUMN=NEXT is specified, the column chosen for the map is the next column from the left. If a numeric value is specified, the column with that number is chosen, counting from the left. If COLUMN=SAME is specified, the left reference column is chosen. (The left reference column is the one that was most recently specified by number with JUSTIFY=LEFT.)

The map is then checked to ensure that its right margin is not to the right of the next column from the right. If it is, the map will not fit into the remaining space, so a new line must be selected. This will be either the next full line or, if the map is too deep, the first available line on the next page.

Finally, the column pointers are updated by setting the next column from the left to the right margin of the map, and, if COLUMN=number was specified, by setting the left reference column to the specified column number.

PAGE BUILDING EXAMPLES

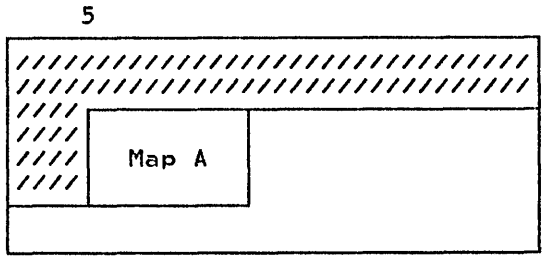
The effects of the mechanisms described above are illustrated by the following examples. The examples show the interactions between SIZE, LINE, COLUMN, and JUSTIFY=LEFT or RIGHT; header and trailer maps and JUSTIFY=FIRST or LAST are not brought into the examples.

In processing a BMS command, BMS determines whether the area of the page required by the map is wholly available or whether any part of it has been used by an earlier command. "Used" means actually filled by a map or rendered unavailable as described below.

1. When the LINE operand of the DFHMDI macro is coded, all lines above the specified line are unavailable.

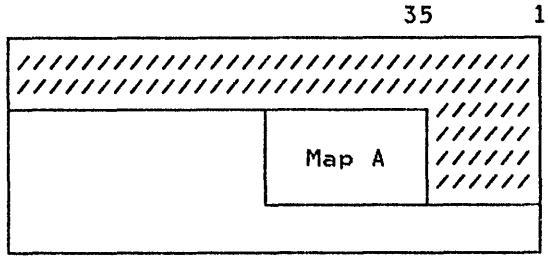
2. When JUSTIFY=LEFT is coded (or applied by default), all columns to the left of the leftmost map column, for the full depth of the map, are unavailable

```
MAPA DFHMDI  ...,LINE=3,COLUMN=5,
                    JUSTIFY=LEFT,...
```



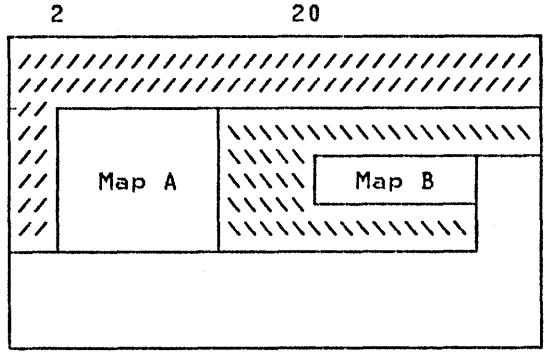
3. When JUSTIFY=RIGHT is coded, all columns to the right of the rightmost map column, for the full depth of the map, are unavailable.

```
MAPA DFHMDI  ...,LINE=3,COLUMN=35,
                    JUSTIFY=RIGHT,...
```

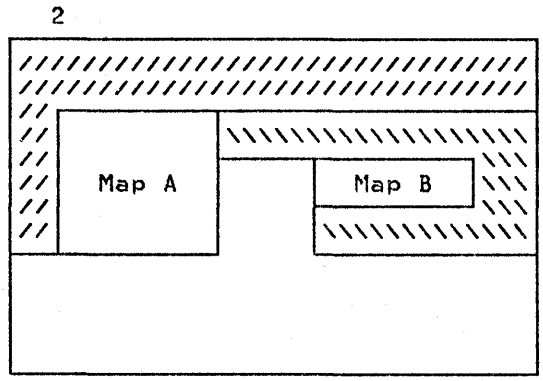


4. When two or more maps are placed so that they share certain lines, all columns beneath a map that ends higher are unavailable to the depth of the map that ends lowest. Similarly unavailable are all columns to the left (if the higher map is left justified) or to the right (if the higher map is right justified) of the 'used' area beneath the higher map.

```
MAPA DFHMDI  ...,LINE=3,COLUMN=2,
                    JUSTIFY=LEFT,...
MAPB DFHMDI  ...,LINE=4,COLUMN=20,
                    JUSTIFY=LEFT,...
```



```
MAPA DFHMDI   ...,LINE=3,COLUMN=2,
                  JUSTIFY=LEFT,...
MAPB DFHMDI   ...,LINE=4,COLUMN=20,
                  JUSTIFY=RIGHT,...
```




```
MAPA DFHMDI   ...,LINE=3,COLUMN=40,
                  JUSTIFY=RIGHT,...
MAPB DFHMDI   ...,LINE=3,COLUMN=1,
                  JUSTIFY=LEFT,...
```

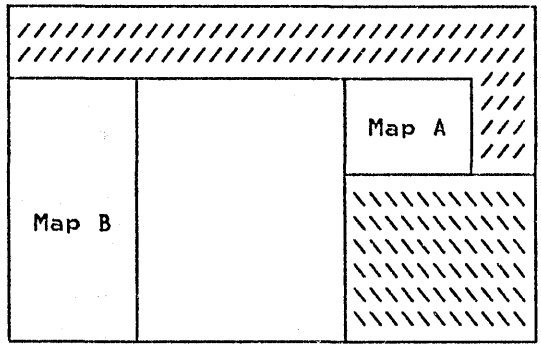


Figure 17 on page 147 shows the effect of several different maps on one page.

If an area of the page directly specified for a map has already been used by a previous map, the overflow condition is raised. This condition is handled as described later in the chapter under "Overflow Processing."

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. (Refer to "Copying Symbolic Description Maps" below.)

Data can then be passed to and from the application program under the field names in the symbolic description map. (The 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 new symbolic description map must then be copied into the application program and the program reassembled or recompiled. There are some map alterations that can be made without reassembly of the symbolic description map, in particular, COLOR, PS, HILIGHT, and VALDN can be added to existing maps if it is not required to change the attributes dynamically. It is only necessary to specify EXTATT=MAPONLY, define the new attributes, and reassemble the physical map.

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

By using BMS to construct and interpret data streams, application programs can be insulated from the device-dependent considerations required to handle the data streams. If necessary, the application program can modify temporarily the attributes or the initial data of any named field in an output map. A collection of named attribute combinations is supplied within BMS so that the application program remains essentially independent of the data stream format.

The ability to add to map definitions without obsoleting existing application programs permits the design and implementation of systems in a modular fashion 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.

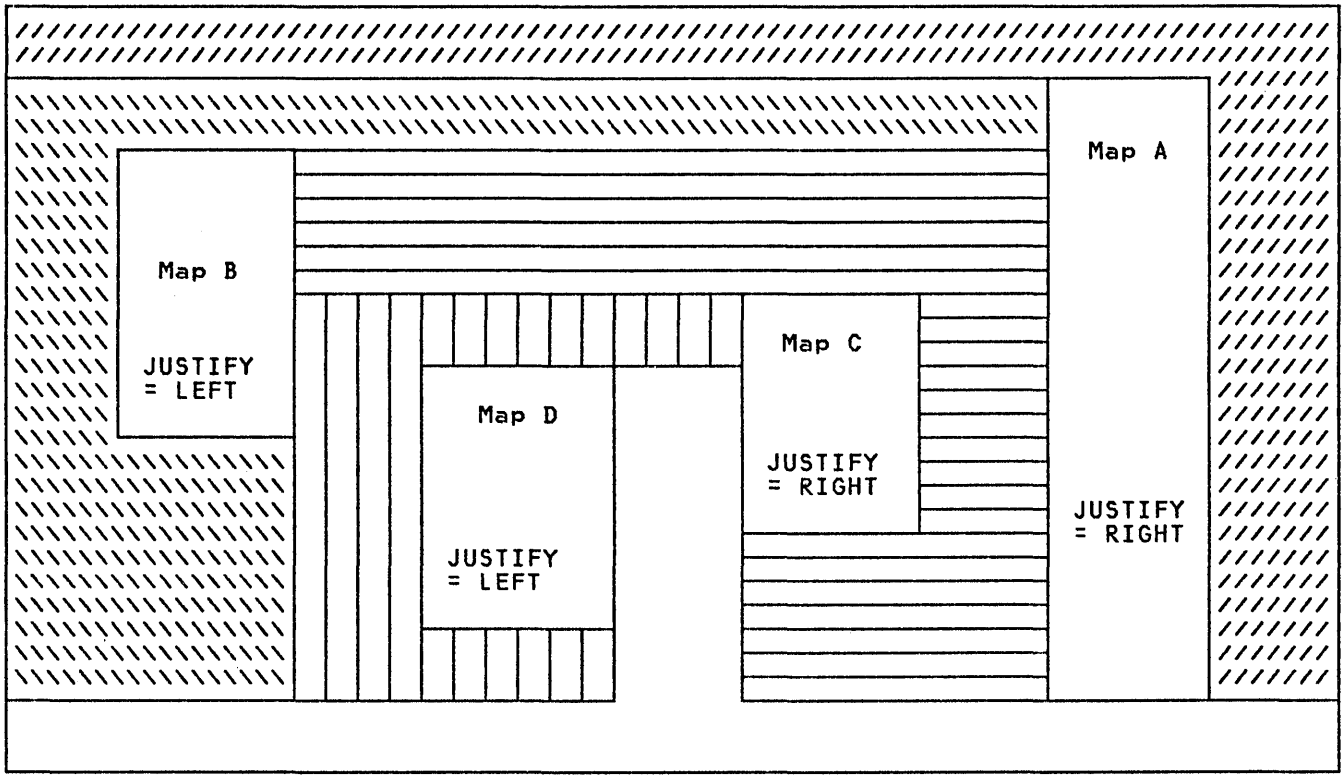


Figure 17. Map Positioning for More than One Map

COPYING SYMBOLIC DESCRIPTION MAPS

The symbolic description maps must be copied into the application program as shown in the following examples; "mapsetname1", "mapsetname2", and "mapsetname3" are the names of members that contain the assembly of a BMS symbolic storage definition. The TIOAPFX=YES operand must be specified in the DFHMSD macro instructions used to define the maps.

1. Assembler language COPY statements.

   ```
   COPY mapsetname1
   COPY mapsetname2
   COPY mapsetname3
   .
   .
   .
   ```

   The symbolic storage definitions can be copied into the DFHEISTG DSECT, in which case storage will be provided automatically. Alternatively, the application program can provide its own DSECT, storage, and addressability.

   While it is generally stated that TIOAPFX=YES must be specified in the map definition macros, it is possible to use maps created without the TIOA prefix if the following technique is used.

   The EXEC interface program assumes that the FROM or INTO option specifies an area which includes the 12-byte TIOA prefix. If the symbolic description maps do not include this, the COPY instruction in the DFHEISTG DSECT should be preceded by a filler, as follows:

   ```
   NEWNAME     DS 12C
               COPY MAP1
   ```

   and the command must specify the FROM or INTO option instead of using the default, for example as follows:

   ```
   EXEC CICS RECEIVE MAP('MAP1')
                          INTO(NEWNAME)
   ```

2. COBOL COPY statements. The names "mapname1", "mapname2", and "mapname3" in this example are the names of the first maps in the map sets. These names include the appropriate suffix to signify the type of map; that is, "I" for input (or input/output), and "O" for output.

   The symbolic storage definitions can be copied into either the linkage section or the working-storage section.

If the symbolic storage definition is copied into the linkage section, the required storage must be obtained by the application program and access to this storage made by the BLL (base locator for linkage) mechanism, as follows:

```
01  BLLCELLS.
    02  FILLER PIC S9(8) COMP.
    02  MAP1BLL PIC S9(8) COMP.
    02  MAP2BLL PIC S9(8) COMP.
    02  MAP3BLL PIC S9(8) COMP.
    .
    .
    .
01 mapname1 COPY mapsetname1.
01 mapname2 COPY mapsetname2.
01 mapname3 COPY mapsetname3.
.
.
.
```

If the symbolic storage definition is copied into the working-storage section, and there is more than one map in the map set, and separate storage is required for the data in each map, the STORAGE=AUTO operand must be specified in the DFHMSD macros.

If working storage is used as the origin or destination of data processed by BMS it should be initialized with low-values by a "MOVE LOW-VALUES TO..." statement.

3. PL/I %INCLUDE statements.

```
%INCLUDE mapsetname1;
%INCLUDE mapsetname2;
%INCLUDE mapsetname3;
.
.
.
```

The symbolic storage definitions may specify AUTOMATIC or BASED storage depending on the operands of the DFHMSD macro.

LOGICAL MESSAGE BUILDING

Logical message building allows the application program to:

- Combine several small mapped data areas into one or more pages of output, or
- Prepare more output than can be contained in one page of output.

A **page** is the area of a terminal on which data can be displayed or printed at one time. The size of the area (in numbers of lines and columns) for the terminal is specified in the TCT by the system programmer. A page of output may be constructed by BMS from several small maps, and these maps must be generated together to form a map set.

The SEND MAP command is used to map and position portions of a page. If all data to be mapped cannot be contained on one page, the overflow condition occurs and control is passed to an overflow routine within the application program. This routine normally causes any required trailer (footing) data to be placed at the foot of the page, the current page to be written to temporary storage, a new page to be started, a heading to be placed on the new page, and the data causing the overflow to be mapped on the new page.

As each page of output is completed, it is written to temporary storage to await completion of other pages. The result of building output data in this cumulative manner is known as a **logical message**. A SEND PAGE command signifies completion of the logical message. Alternatively, the logical message is completed upon termination of the application program unless CICS/VS has insufficient storage available, in which case the logical message is deleted.

An alternative way to build a logical message without the use of maps is by means of SEND TEXT commands. Data is passed in text data format, which BMS places on succeeding lines (and pages, if necessary) without reference to maps. A word is not split between lines; any word that cannot fit on the remaining portion of a line is placed on the next line. Formatting can be controlled by new-line characters (X'15') embedded within the text. A SEND PAGE command signifies completion of the logical message; alternatively, the logical message is completed upon termination of the application program unless CICS/VS has insufficient storage available, in which case the logical message is deleted.

OUTPUT OPERATIONS

The SEND MAP and SEND TEXT commands can be used individually to request BMS to map data and transmit it to a terminal or to a data area in the application program.

Alternatively, these commands can be used to build a logical message cumulatively. The logical message is built by successive SEND MAP or SEND TEXT commands, each of which must include the ACCUM option. Finally, a SEND PAGE command must be issued to complete the logical message and transmit it.

SEND MAP and SEND TEXT commands cannot be used to build portions of the same logical message. The process of building a logical message can be discontinued by means of a PURGE MESSAGE command, which

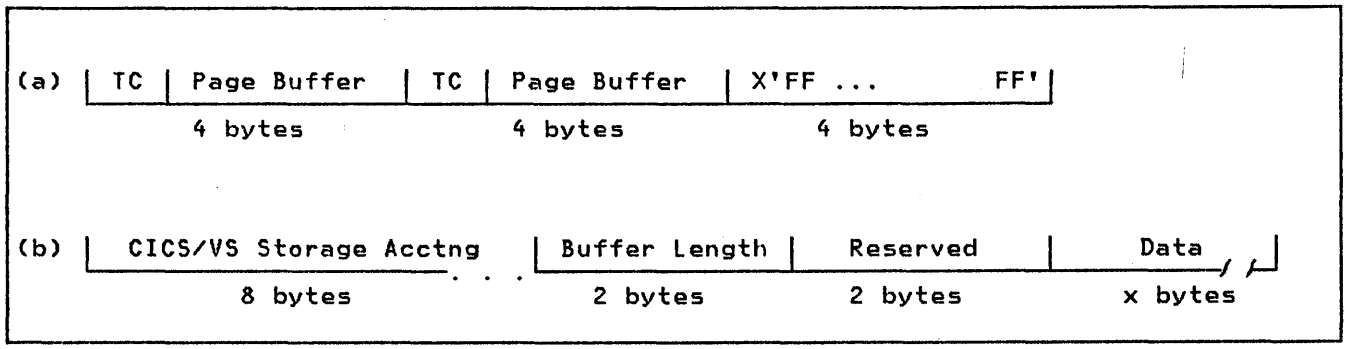


Figure 18. Page Address List (SET Option)

deletes the portions of the message already built.

OUTPUT COMMANDS WITH THE SET OPTION

The SET option of the SEND MAP and SEND TEXT commands causes completed pages to be returned to the application program and a pointer to be set to the address of a list of completed pages. Since more than one page of output may result from a single BMS output command, there may be more than one entry in the list for a given type of terminal. The entries for each type of terminal immediately follow one another in the list (TC is the terminal code as described in the next section). The list is laid out as shown in Figure 18.

The page buffer pointer in (a) of Figure 18 points to an area of storage which has an eight-byte storage accounting prefix, as shown in (b) of Figure 18.

At this point, page buffers are on the user's storage chain and are disassociated from BMS control blocks; when no longer needed, page buffers should be released by the FREEMAIN command. The data to be freed should not include the storage accounting prefix. The storage containing the list of buffers should not be freed; the list will be reused to reduce processing time. This list will be altered by the next BMS command; its contents must be saved before that command is executed.

TERMINAL CODE TABLE

A terminal code table is established within BMS for reference in servicing BMS-supported terminals. There is one entry in this table for each terminal supported under BMS. The terminal codes that appear in the table are given below. This code appears in the list of completed pages made available to the application program when the SET option is specified in a SEND MAP or SEND TEXT command. The code is available also in the EIBRCODE field of the EXEC interface block when the INVMPSZ condition occurs; for a description of this field, refer to "Appendix A. EXEC Interface Block" on page 239.

Code	Terminal or Logical Unit
A	CRLP or TRMTYPE=TCAM terminals
B	Magnetic Tape
C	Sequential Disk
D	TWX Model 33/35
E	1050
F	2740-1,-2 (no buffer receive)
G	2741
H	2740-2 (with buffer receive)
I	2770
J	2780
K	3780
L	3270 (40-character width)
M	3270 (80-character width)
N	Not used
O	Not used
P[1]	3767/70 Interpreter LU
Q	2980 Models 1 and 2
R	2980 Model 4
S	Not used
T	Not used
U	3600 (3601) LU
V	3650 Host Convers (3653) LU
W	3650 Interpreter LU
X	3650 Host Convers (3270) LU
Y[2]	3770 Batch LU
Z	Not used

[1] Used also for the 3790 full function logical unit, the SCS printer logical unit, and the SCS printer.

[2] Used also for the 3770 and 3790 batch data interchange logical units.

MESSAGE ROUTING

Message routing permits an application program to build and route a logical message to one or more terminals. The message is scheduled, for each designated terminal, to be delivered as soon as the terminal is available to receive messages, or at a specified time.

A ROUTE command initiates a message routing operation. It is followed by SEND MAP or SEND TEXT commands to build the logical message to be routed. A SEND PAGE command terminates the page building and causes the message to be routed. When individual logical messages are routed to a terminal, they are not necessarily delivered in the sequence in which they were issued. If a specific sequence is required, the pages must be output as one message.

The SEND MAP or SEND TEXT commands that build the message must include the ACCUM option. Other SEND MAP or SEND TEXT commands without the ACCUM option can be interleaved with these commands to send messages to the terminal that initiated the transaction while the message to be routed is being built.

Another consideration of routing to different types of terminal is the handling of overflow conditions. Since different types of terminal may have different page sizes, the overflow condition is apt to occur at different times in page building. BMS returns control to an overflow routine in the application program, indicating which type of terminal caused the overflow and how many pages have been created for that type.

The message routing facility of BMS is useful for developing message switching and broadcasting applications. CICS/VS provides a generalized message switching application program that uses the message routing facility of BMS (see the CICS/VS Operator's Guide for details).

BMS MESSAGE RECOVERY

BMS provides message recovery for routed and non-routed messages. Recoverable messages must satisfy the following requirements:

- The PAGING option must have been specified in the BMS output commands that built the logical message.
- The BMS default REQID (**) or the specified REQID for the logical message must have been identified to the temporary storage program (via the TST) as recoverable.
- The task that built the message must have reached its logical end of task.
- The temporary storage program and the interval control program must also support recovery.

DISPLAY DEVICE OPERATIONS (BMS)

The information in this section applies, in general, only to the IBM 3270 Information Display System. All the basic facilities described in the section "Display Device Operations" in "Chapter 3.2. Terminal Control" on page 85 can be requested in a BMS program. The following additional facilities apply only to BMS, and are described in the following sections:

- Symbolic Cursor Positioning
- Terminal Operator Paging Commands

SYMBOLIC CURSOR POSITIONING

The CURSOR option of the SEND MAP and SEND TEXT commands can be used to position the cursor on completion of an output operation. Alternatively, a method called symbolic cursor positioning can be used, which 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 symbolic cursor positioning are as follows:

- MODE=INOUT must be specified in the DFHMSD macro.
- CURSOR must be specified in the BMS command.
- The length field, suffix "L", associated with the field under which the cursor is to be placed must be initialized to -1.

The remainder of the data may be built as desired by the user. Symbolic cursor positioning is operable only for devices that allow cursor placement to be performed independently of data placement; for example, 3604 and 3270. Symbolic cursor positioning is ignored for other devices.

TERMINAL OPERATOR PAGING COMMANDS

The commands used by terminal operators to communicate with BMS are collectively known as **terminal paging commands**, or simply as paging commands. Their format and use are discussed in detail in the CICS/VS Operator's Guide.

Cursor placement is an important consideration in programming for paging commands. Any of the following can cause a paging command not to be the first data read by CICS/VS and therefore not to be interpreted as a paging command.

- After a print operation on a 3275 Display Station, the cursor is set to

position zero. A paging command entered at this location is not recognized unless the last position of the buffer contains an attribute byte or the buffer has been cleared.

- A field sent with the DATAONLY option of the SEND MAP command and without an attribute in the data (that is, with an attribute byte in the data having the value X'00') is written into the buffer without an attribute byte. If the application program places the cursor in this field and the operator keys a paging command beginning at the cursor location, the paging command is not recognized.

  Since the field has no attribute byte, the data is considered to be an extension of the previously defined field. When the operator keys into the middle of the hardware-recognized field and presses the enter key, the field is transmitted from the beginning of the previously defined field. The data at the beginning of the field is examined for a paging command and responded to accordingly.

- Cursor specification in the BMS commands can adversely affect operator action if the cursor is not set at the beginning of a field. Paging commands entered at a cursor location that is not the beginning of a field are not recognized by BMS because data transmission starts at the beginning of the field if the field is not set to nulls (X'00').

MAP INPUT DATA (RECEIVE MAP)

```
RECEIVE MAP(name)
[SET(ptr-ref)|INTO(data-value)]
[MAPSET(name)]
[FROM(data-area) LENGTH(data-value)|
 TERMINAL[ASIS]]

Conditions:
EOC, EODS, INVMPSZ, MAPFAIL, RDATT
```

This command is used to map data into a data area in the application program. The source of the data can be either a terminal (TERMINAL option) or another data area in the program (FROM option). If neither option is specified, TERMINAL is assumed. The ASIS option inhibits translation of lowercase characters to uppercase.

If the FROM and LENGTH options are used, the length specified must equal the value received by the corresponding terminal control RECEIVE command that includes the INTO and LENGTH options.

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 reference given in the SET option.

Data from certain logical units is not mapped, but is left unaltered. Refer to the appropriate CICS/VS subsystem guide for details.

If neither the INTO option nor the SET option is specified, it is assumed that the data is to be mapped into the data area defined by the symbolic description map copied into the program. This can be accomplished only if the map name provided is a literal constant. If it is a variable, INTO or SET must be specified. 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 PERSN must be referred to in the application program as PERSNI.)

The data area into which the data is mapped must include a 12-byte prefix for use by BMS. The application program must make provision for this prefix only if a data description other than the BMS-supplied symbolic description is used, or if TIOAPFX=YES is omitted from the DFHMDI macro defining the map.

If the symbolic description is included in the linkage section of a COBOL application program, the 12-byte prefix must not be overwritten.

If RECEIVE MAP commands are used to read data from a 3770 batch logical unit, the FMHs will be removed. However, if an FMH is required, a terminal control RECEIVE command should be included to deal with the FMH, followed by a RECEIVE MAP command with the FROM option to map the data.

MAP OUTPUT DATA (SEND MAP)

```
SEND MAP(name)
FROM(data-area)[DATAONLY]|MAPONLY
[LENGTH(data-value)]
[MAPSET(name)]
[FMHPARM]                          LUs only
[REQID(name)]
[LDC(name)]                        LUs only
[CURSOR[(data-value)]]
[SET(ptr-ref)|PAGING|
  TERMINAL[WAIT]]
[ACCUM]
[ERASE|ERASEAUP]
[PRINT]
[FREEKB]
[ALARM]
[FRSET]
[L40|L64|L80|HONEOM]
[NLEOM]
[LAST]                             LUs only

Conditions:
IGREQCD, IGREQID, INVLDC, INVMPSZ,
INVREQ, OVERFLOW, RETPAGE, TSIOERR,
WRBRK
```

This command is used to map output data. Several successive SEND MAP commands with the ACCUM option can be used to build a logical message, which must be completed by a SEND PAGE command.

If the FROM option is omitted, it is assumed that the data to be mapped is in the data area defined by the symbolic description map copied into the program. This assumption is valid only if the map name provided is a literal; if it is a variable, the FROM option must be specified. 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 area named.

The data area specified by the FROM option must include a 12-byte prefix for use by BMS. The application program must make provision for this prefix only if a data description other than the BMS-supplied symbolic description is used.

In the symbolic description map definition, the DFHMSD macro must have the TIOAPFX=YES operand specified either explicitly or implicitly by the appearance of the STORAGE=AUTO operand.

The mapped data can be transmitted to a terminal (specify the TERMINAL or PAGING option) or made available to the application program in its mapped form (specify the SET option). If none of these options is specified, TERMINAL is assumed. The WAIT option specifies that control is not to be returned to the program until the operation is completed.

The PAGING option causes the logical message to be placed in temporary storage until it is requested by paging commands entered by the terminal operator. The PAGING option conflicts with the LAST option and is ignored.

If the disposition specified by the PAGING, SET, or TERMINAL option is changed while a logical message is being built, the INVREQ condition occurs.

The DATAONLY and MAPONLY options are used to specify that application-program data only, or default data only, is to be written. If both these options are omitted, data placed in the data area named in the FROM option by the application program is merged with default data from the map. The user-supplied data and/or attribute character (3270 only) supplied for a given field replaces the corresponding default data and/or attribute character from the map. The MAPONLY and FROM options are mutually exclusive. If the user-supplied data for a field is X'00', the data from the map for that field is used. If the user-supplied attribute for a field is X'00', the attribute from the map for that field is used.

The mapped data is positioned by BMS within an area large enough to contain one page of output. The application program need not keep track of when a page is full: a HANDLE CONDITION OVERFLOW command will cause BMS to transfer control to an overflow routine.

The ERASE option should always be specified on the first SEND MAP command to select the correct screensize for the application.

If ACCUM is specified, the pointer-ref in the SET option will be updated with the address of the completed page only after the RETPAGE condition is raised. In an assembler-language program the SET option specifies a register instead of a pointer, and this register will not be set even though the RETPAGE condition has been raised. However, the register can be loaded from DFHEITP1.

OVERFLOW PROCESSING

Overflow occurs when the number of lines in the requested map plus the number of lines in the largest trailer map in the map set (if there are any trailer maps) is greater than the number of lines remaining in the page being built for the terminal involved in an output operation.

For logical units having LDC support, pages are accumulated individually by LDC mnemonic. Therefore, overflow may occur at end of page for each different LDC mnemonic used in different BMS commands.

The LDC mnemonic is accessible to the application program from LDCMNEM, and the LDC numeric value from LDCNUM. ASSIGN commands must be used to determine the values of LDCMNEM and LDCNUM.

Overflow can occur on a logical message being built for routing. If the route list contains more than one LDC mnemonic, the returned LDC mnemonic and numeric value is the first LDC mnemonic resolved in the route list. Refer to the section "Route a Logical Message (ROUTE)" later in this chapter for details of route lists.

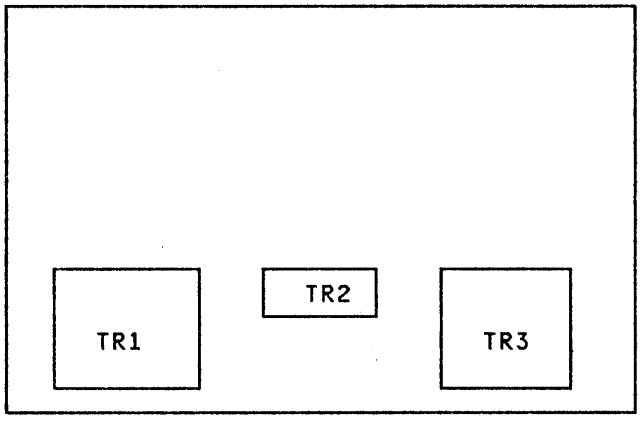
The routine to which control is transferred (specified in a HANDLE CONDITION OVERFLOW command) must be in the application program, but no special considerations apply. The data which was to have been mapped, but which caused the overflow, is not mapped by BMS and remains unaltered.

If a ROUTE command has not been issued previously, there is only one destination. If a ROUTE command has been issued, the logical message is probably being built for more than one destination. Since the application program can build pages concurrently for terminals that have different-sized output, overflow may occur at different times for different terminal groups. The overflow routine gets control every time any one of the destinations or groups of destinations encounters an overflow condition. The application program overflow routine must determine which destination or group of destinations has encountered the overflow.

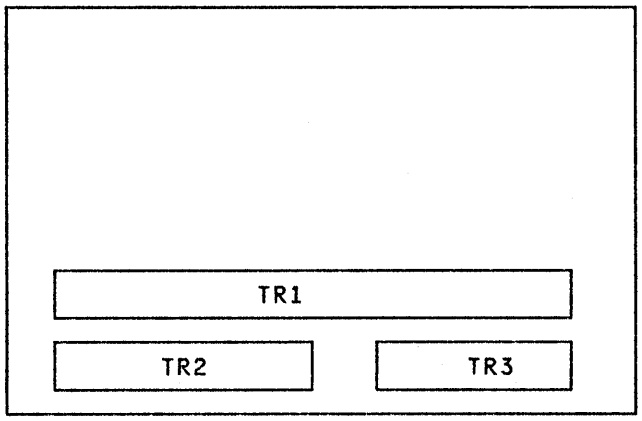
Upon return to the application program from a ROUTE command, a count of the number of destinations or groups of destinations can be determined by means of the DESTCOUNT option of the ASSIGN command. This count tells the application program how many overflow control areas (for example, accumulators) are required. Whenever the overflow routine gets control, DESTCOUNT indicates the relative overflow control number of the destination that has encountered the overflow. This number indicates which control area should be output, perhaps through one or more trailer maps.

In addition to the relative control count, BMS returns the current page number for the destination that has encountered the overflow. This page number can be determined by means of the PAGENUM option of the ASSIGN command.

The SEND MAP command is used to place trailer data on a page. The macros used to format the data must contain TRAILER=YES so that the amount of space on the page to reserve for overflow can be calculated. More than one trailer map may be placed on a page. There should be a dummy trailer map (not otherwise used) in the map set specifying the number of lines to be reserved for trailer data if no single trailer map extends over the total number of lines required for trailer data (see diagrams). Maps used to map trailer data may contain JUSTIFY=LAST to force their placement at the bottom of the page. An attempt to place more lines of trailer data on the page than are available causes the trailer data to be placed on a separate page by itself. Yet another page is built to continue mapping with or without a header map.



No dummy trailer required.



Dummy trailer required.

The SEND MAP command is used also to process header data and place it on a page. The maps used to map header data must specify JUSTIFY=FIRST to complete processing of the previous page if that has not been done, and to begin a new page. An attempt to place more header data on the page than the page can contain causes multiple pages to be created.

If a header map is not used, JUSTIFY=FIRST must be specified for the first map used after OVERFLOW is raised, if a line number is also specified to force out the previous page. Failure to

specify this will cause OVERFLOW to be raised again immediately.

When all trailer and/or header data has been processed, the command that caused the overflow must be reissued, since this data has not yet been mapped for all destinations.

It is important to recognize that BMS maintains the overflow environment for as long as the application program issues BMS commands using maps defined as headers or trailers. The first use of a map that is not defined as a header or trailer terminates overflow processing.

This coincides with reissuing the command that caused the overflow.

If an overflow routine has not been specified in a HANDLE OVERFLOW command, no overflow occurs and new pages will be forced automatically. If a header is to be placed on the first page and a trailer on the last, the OVERFLOW condition would not be used.

An overview of overflow processing is given in Figure 19.

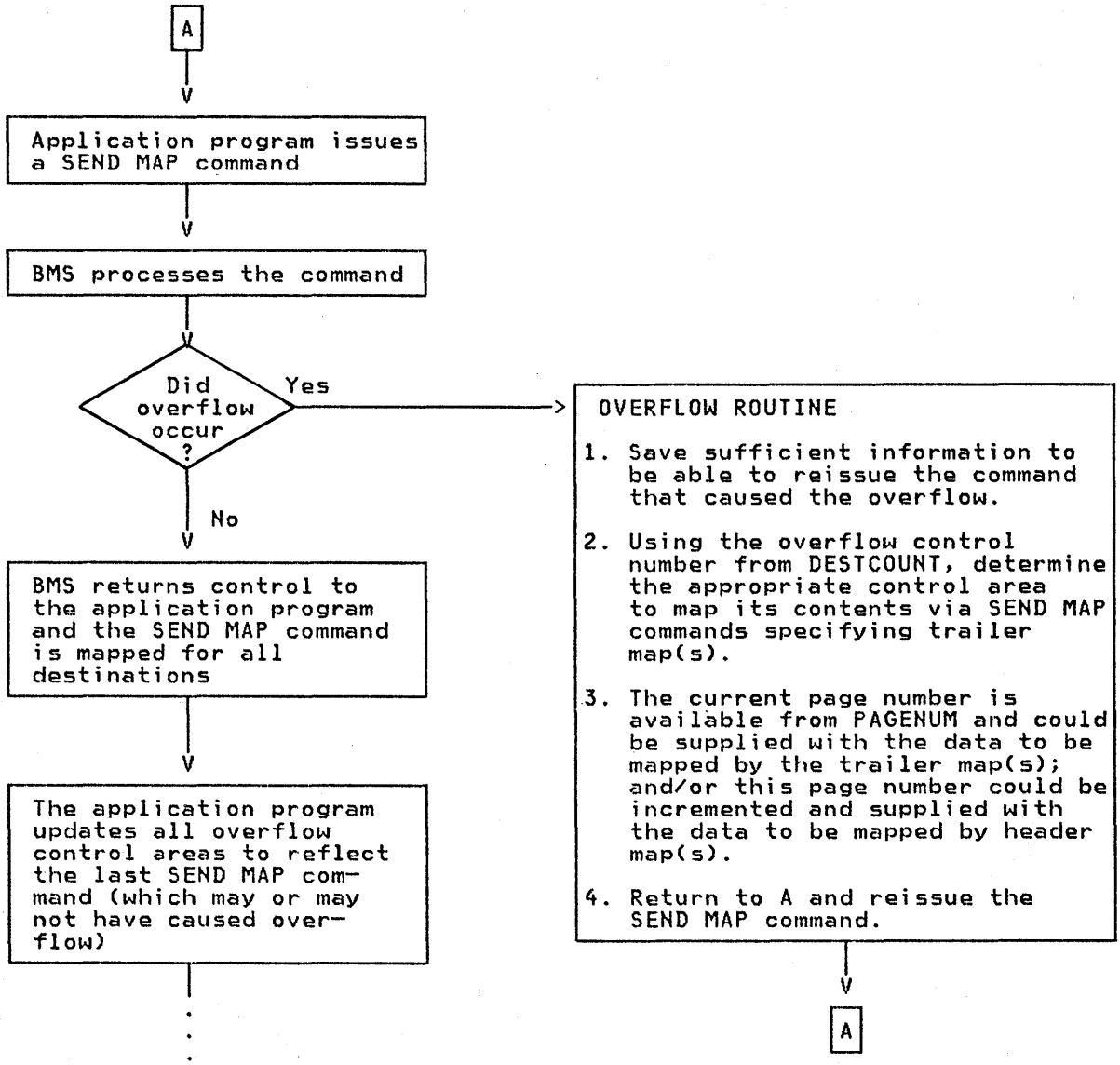


Figure 19. Overflow Processing

FORMAT OUTPUT DATA WITHOUT MAPPING (SEND TEXT)

```
SEND TEXT FROM(data-area)
LENGTH(data-value)
[FMHPARM]                      LUs only
[REQID(name)]
[LDC(name)]                    LUs only
[CURSOR(data-value)]
[SET(ptr-ref)|PAGING|TERMINAL[WAIT]
[HEADER(data-area)]
[TRAILER(data-area)]
[JUSTIFY(data-value)|JUSTFIRST|
 JUSTLAST]]
[ACCUM|NOEDIT]
[ERASE]
[PRINT]
[FREEKB]
[ALARM]
[L40|L64|L80|HONEOM]
[NLEOM]
[LAST]                         LUs only

Conditions:   IGREQCD, IGREQID,
INVLDC, INVREQ, RETPAGE,
TSIOERR, WRBRK
```

This command is used to format output data without mapping. Several successive SEND TEXT commands with the ACCUM option can be used to build a logical message, which must then be completed by a SEND PAGE command. The beginning and ending of pages is handled by BMS and does not affect the application program.

The data to be transmitted, specified by the FROM and LENGTH options, can be sent to a terminal (specify the TERMINAL or PAGING option) or made available to the application program in its formatted form (specify the SET option). If none of these options is specified, TERMINAL is assumed. The WAIT option specifies that control is not to be returned to the program until the operation is completed.

The PAGING option causes the logical message to be placed in temporary storage until it is requested by paging commands entered by the terminal operator.

If the disposition specified by the PAGING, SET, or TERMINAL option is changed while a logical message is being built, the INVREQ condition occurs.

The options HEADER, TRAILER, JUSTIFY, JUSFIRST, and JUSLAST can be used to edit the output pages. Any of these options imply the ACCUM option .

The NOEDIT option allows the application program to control the insertion of device-dependent control characters and the following notes apply when it is omitted:

1. SEND TEXT formats data for each terminal so that output lines are no longer than the line-length of the terminal as specified in the TCT, and wherever possible the output line is broken at a blank character. The user may force a new line at a particular point by inserting a new-line (X'15') character in the data stream presented to BMS via SEND TEXT.

2. For all terminal types, SEND TEXT interprets the data stream with regard to the line-size of the terminal found in the TCT and any embedded X'15' characters, and builds an internal representation of the final appearance of the data on the terminal. Control characters other than X'15' are treated as normal character data, and their presence may in certain cases disrupt the results of this internal formatting process.

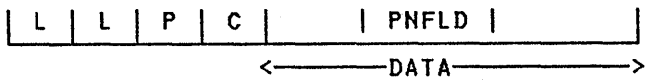
3. If the output terminal is a 3270 device and NLEOM is not specified, SEND TEXT uses the line-length specified in the TCT to position the data in the device buffer so that when displayed it will be in the correct format. Hardware new-line characters are not used: instead it is the position of the data in the buffer which determines the output format. Therefore if the actual line-length of the terminal differs from that specified in the TCT, the resulting output will not be correctly formatted on the terminal.

   For example, if the terminal is a 3270 printer with 132 print positions but a TCT line-length of 80, to get correct output format without specifying NLEOM it is necessary to specify L80 in the WCC.

The formats of the header and trailer data are described below.

HEADER AND TRAILER FORMAT

The data areas named in the HEADER and TRAILER options have the following format:



where:

LL
is a halfword binary field containing the length of the header or trailer data. (The value includes the two bytes for this field.)

P

is a one-byte field whose contents indicate whether page numbering is required or not. If the field contains a character[1] other than a blank (X'40'), page numbering is required. The character specified is the character that is embedded in the header or trailer data in the positions (a maximum of 5) where the page number is to appear. If the field contains a blank, page numbering is not required.

([1] X'0C', X'15', X'17', X'26', and X'FF' are reserved and cannot be used).

C

is a reserved one-byte field.

PNFLD

is the page number field. This field can be embedded anywhere in the header or trailer data in the required page number position. It can contain from one through five occurrences of the character specified by P. These characters will be replaced by the current page number, up to a maximum of 32,767, as a page is built. A SEND PAGE command will causes the page number to be reset to 1.

DATA

is the header or trailer data to be placed at the beginning or end of each page of output. Embedded new-line characters (X'15') may be used to provide multiple heading or footing lines.

OUTPUT DATA WITH EXTENDED ATTRIBUTES

When the data is destined for a device with extended attributes, set attribute (SA) orders can be included also in the data stream. These orders enable characters or words in the data stream to be modified by the extended attributes. These orders will be ignored during calculation of line lengths. Orders for extended attributes not supported by a terminal will be removed from the data stream. If a sequence of orders is less than three characters, or contains an invalid attribute type, the transaction will be terminated abnormally (ABMX).

Attributes will remain effective until overridden by subsequent orders. If output exceeds a page, the attributes will apply for the following page. However, in headers or trailers, the attributes will be reset to their default values until changed by a new sequence of orders within the header or trailer. On resumption of normal processing of text after the header or trailer, the previous attributes will be restored.

To aid the modification of characters or words, the following symbolic names are available in DFHBMSCA (the standard attribute list): DFHSA, DFHCOLOR, DFHPS, DFHHLT, and DFHALL. (The standard list DFHBMSCA is described in "Chapter 3.2. Terminal Control" on page 85.) The following example shows PL/I statements that will color a single word blue:

```
TEXTSTR='data '||DFHSA||DFHCOLOR||
         DFHBLUE||'blueword '
        ||DFHSA||DFHCOLOR||
         DFHDFCOL||'rest of data';

SEND TEXT FROM(TEXTSTR) LENGTH(100);
```

COMPLETE AND TRANSMIT A LOGICAL MESSAGE (SEND PAGE)

```
SEND PAGE [[TRANSID(name)]|RELEASE]|
  RETAIN]
[TRAILER(data-area)]
[FMHPARM(name)]                LUs only
[AUTOPAGE[CURRENT|ALL]|NOAUTOPAGE]
[OPERPURGE]
[LAST]                         LUs only

Conditions:  IGREQCD, IGREQID,
INVREQ, RETPAGE, TSIOERR, WRBRK
```

This command is used to complete and transmit a logical message built by one or more SEND MAP or SEND TEXT commands with the ACCUM option.

Options can be included to specify how much control the terminal operator should have over the disposition of the logical message (AUTOPAGE and OPERPURGE), to determine whether control should return to the application program after transmission of the logical message (RELEASE and RETAIN), and to add trailer data to the logical message (TRAILER). The format of the trailer data is described under "Format Output Data Without Mapping (SEND TEXT)" earlier in this chapter.

If neither AUTOPAGE nor NOAUTOPAGE is specified, the paging status specified for the terminal at system generation determines how pages are to be written to the terminal. For logical units with LDC support, paging status for each LDC is obtained from the system LDC table.

To ensure that a logical message appears at the receiving terminal before any messages that may have been routed to it, or before other transactions are initiated from the terminal, RELEASE should be specified. Control then returns to an application program at the next higher logical level or to CICS/VS; this action is as if a RETURN program control command had been issued. When

control returns to CICS/VS, the TRANSID option specifies the transaction identifier for the next application program to be associated with the terminal; the TRANSID option has the same function and restrictions on its use as the TRANSID option of the RETURN command. Refer to "Chapter 4.4. Program Control" on page 189, "Program Control," for information about application program logical levels, the way in which control returns through the levels, and the use of the TRANSID option.

RETAIN is intended to be used for a combination of page display from the page file (logical message built using PAGING) and operator data entry. BMS issues an input request to the terminal after writing the appropriate pages to the terminal. BMS issues the input request only if the logical message is built with PAGING. If the logical message is built without PAGING, BMS returns control to the application program after the last page is written to the terminal, and without issuing an input request to the terminal.

The operator may enter any page, purge, or copy commands that are valid for the particular message. Any other entered data is passed back to the application program after the current message is deleted.

If neither RETAIN nor RELEASE is specified and the logical message is to be retrieved by terminal-operator requests (PAGING specified in the SEND MAP or SEND TEXT command), a new task is scheduled for writing pages to the terminal. Control is returned to the application program immediately, rather than after the pages have been written. RETAIN and RELEASE are ignored for routed messages.

If an error occurs during the processing of a SEND PAGE command, control is returned to the application program, and the RETAIN or RELEASE specification is ignored. The logical message is not considered complete. The application program should either retry the SEND PAGE operation or delete the logical message.

Any logical message started but not completed when a SYNCPOINT command is executed is forced to completion by an implied SEND PAGE command.

DELETE A LOGICAL MESSAGE (PURGE MESSAGE)

```
PURGE MESSAGE

Condition:  TSIOERR
```

This command is used to discontinue the building of a logical message. The portions of the logical message already built in main storage or in temporary storage are deleted.

ROUTE A LOGICAL MESSAGE (ROUTE)

```
ROUTE [INTERVAL(hhmmss)|INTERVAL(0)|
  TIME(hhmmss)]
[ERRTERM[(name)]]
[TITLE(data-area)]
[LIST(data-area)]
[OPCLASS(data-area)]
[REQID(name)]
[LDC(name)]                    (LUs only)
[NLEOM]

Conditions:  INVERRTERM, INVLDC,
RTEFAIL, RTESOME
```

This command is used to initiate the building of a logical message that is to be scheduled for delivery to one or more terminals. It is followed by the SEND MAP or SEND TEXT commands that format the data.

The options LIST and OPCLASS allow the designation of those terminals or logical units, or particular operators, to which the logical message is to be scheduled for delivery. Whether or not the logical message will actually be delivered (that is, received at the terminal) depends on many factors, such as availability of the terminal, or of a specific operator, within a certain time after the logical message is ready to be delivered.

The LIST option specifies a list of terminals and/or operators to receive the routed logical message. If no list is provided, the logical message will be scheduled for delivery to all terminals supported by BMS (unless the OPCLASS option is specified and has some effect).

The OPCLASS option specifies the classes of operators to receive the routed logical message. OPCLASS can be used alone, or in conjunction with LIST.

The uses and format of the route list and of the information to be provided in the OPCLASS option are described in the section "Route List and Operator Class Codes (LIST and OPCLASS Options)" later in this chapter.

The logical message can be delivered at a specified time (TIME option) or after a certain interval has elapsed (INTERVAL option); if neither option is specified, or if INTERVAL(0) is specified, the logical message will be delivered as soon as possible.

If a logical message is to be routed to more than one type of terminal, BMS builds the message for each type. Each message is stored on temporary storage until all terminals of the related terminal type have received the message. If a terminal is scheduled to receive a message but is not eligible, the message is stored until one of the following conditions occurs:

- A change in terminal status allows the message to be sent.
- A period (specified at system generation) has elapsed, causing the message to be deleted by BMS.
- The message is deleted by the destination terminal.

If a logical message is to be routed to terminals with alternate screensize capabilities (for example, the 3278), the choice of alternate or default screensize is made depending on the SCRNSZE operand of the DFHPCT TYPE=ENTRY system macro for the transaction issuing the ROUTE command. (See the CICS/VS System Programmer's Reference Manual.)

If a ROUTE command followed by one or more BMS output commands is not terminated by a SEND PAGE command before a subsequent ROUTE command is issued, the INVREQ exceptional condition occurs. A ROUTE command may be issued immediately following another ROUTE command. In this case, the first ROUTE command is nullified, and the second determines the routing environment.

If a message cannot be delivered within a certain time, it will be deleted (purged); the time is specified in the PRGDLAY (purge delay) operand of the DFHSIT system macro. If the PRGDLAY operand is omitted, undelivered messages await delivery indefinitely. If PRGDLAY is specified, an error message is generated by CICS/VS whenever a message becomes undeliverable. The error message will be sent to the terminal associated with the task that is sending the message; alternatively, the application program can specify a different terminal to receive such error messages by using the ERRTERM option. In addition to sending an error message, CICS/VS lets the master terminal operator know how many undeliverable messages have been deleted for a destination.

In CICS/DOS/VS, there is a DL/I restriction that a single ROUTE command cannot route a message to more than 40 terminals. The restriction applies when:

- DL/I logging to the CICS/VS system log (tape only) is being used
- BMS message recovery is required (that is, the route request specifies a recoverable temporary storage prefix in the REQID option, or the default prefix (**) is defined as recoverable.

To route a message to more than 40 terminals, more than one ROUTE command must be used, each with a LIST option of no more than 40 entries.

The restriction arises because under CICS/DOS DL/I, the log buffer size cannot exceed 1K bytes for tape files, and the limit of 40 terminals in a route list corresponds to a size of 1K bytes for the BMS message control record which will be put on temporary storage and logged to the same file if the temporary storage is recoverable .

DISPOSITION AND MESSAGE ROUTING

A logical message can be built using either of two dispositions: PAGING or SET. The first BMS output command following the ROUTE command (with some exceptions noted below) determines the disposition of the logical message. Once established, the disposition remains unchanged until the logical message is completed by a SEND PAGE command. An output request specifying a disposition that is not in effect results in the INVREQ condition.

PAGING is the normal disposition and results in the logical message either being delivered or deleted. SET causes the logical message to be returned to the application program which is then responsible for its delivery.

INTERLEAVING CONVERSATION WITH MESSAGE ROUTING

A task can converse with the terminal to which it is currently attached while it is building the logical message. The attached terminal is known as the direct terminal; a terminal to which the message is to be routed is known as a routing terminal. If any RECEIVE MAP (or RECEIVE) commands are encountered while the message is being built, they are processed as usual.

The following rules apply to a direct terminal:

- TERMINAL must be specified or implied in any output command that is to go to the direct terminal.
- ACCUM options with a disposition of TERMINAL are invalid and result in the INVREQ condition.
- The direct terminal may be included in the routing environment without impairing the ability to converse with it while under ROUTE. Data

routed to the direct terminal will be delivered as though the ROUTE command had been issued from another terminal.

The following is a list of abridged commands, in order of execution. For each command, the action taken by BMS is shown.

SEND TEXT TERMINAL - Transmit to direct terminal.

ROUTE - Establish routing environment.

SEND TEXT TERMINAL - Transmit to direct terminal.

RECEIVE MAP - Receive from direct terminal.

SEND TEXT PAGING ACCUM - First output command eligible for routing establishes disposition of PAGING.

SEND TEXT TERMINAL - Transmit to direct terminal.

SEND TEXT SET(A) - Invalid request, routed logical message has already established a disposition of PAGING.

SEND TEXT PAGING ACCUM - Continue building routed logical message.

SEND MAP(Y) PAGING ACCUM - Invalid request routed logical message cannot be built with both SEND TEXT and SEND MAP commands.

SEND MAP(Y) TERMINAL ACCUM - Invalid request, cannot issue SEND MAP ACCUM or SEND TEXT ACCUM command to direct terminal while building a routed logical message.

SEND TEXT PAGING ACCUM - Continue building routed logical message.

SEND PAGE - Complete and transmit logical message and terminate routing operation.

SEND TEXT TERMINAL - Transmit to direct terminal.

MESSAGE TITLE

The title named in the TITLE option is displayed with the logical message identifier when the terminal paging query command is entered (see the CICS/VS Operator's Guide). This title serves as an additional message identifier, displayed upon request with the message identifier, not on the logical message. The value in the two byte length field preceding the title includes the bytes used for the length field. The length field and title, in total, may be up to 64 bytes long. For example:

2-byte length field

24-byte title field

ROUTE LIST AND OPERATOR CLASS CODES (LIST AND OPCLASS OPTIONS)

The system programmer specifies the terminal or logical unit identifiers for all the terminals of the CICS/VS system in the terminal control table (TCT). (For logical units with LDC support, LDC mnemonics are specified in the LDC table.) Also, an operator identifier must be specified for each operator, and up to 24 operator class codes (in the range 1 through 24) can be specified for particular operators, using the OPIDENT and OPCLASS operands, respectively, of the sign-on-table system macro (DFHSNT TYPE=ENTRY). When an operator signs on at a terminal, CICS/VS associates the operator and the optional class codes with that terminal until the operator signs off again.

The application program can provide a route list in the LIST option to specify which terminals, or logical units, or operators are to receive the logical message; alternatively, or in addition, up to 24 operator class codes can be specified for use with a ROUTE operation, by using the OPCLASS option.

Before a logical message is delivered, all of the following conditions must be fulfilled:

- The terminal or logical unit must be supported by BMS and be operational.
- The logical message must be ready for delivery (TIME or INTERVAL options satisfied).
- The purge delay must not have expired.

Whether or not a logical message will be delivered at a specific terminal then depends on the use of the LIST and OPCLASS options, as follows:

- LIST and OPCLASS are omitted. All terminals will receive the message.
- LIST is specified but OPCLASS is omitted. The route list can contain three types of entry, each type having a different effect. All three types of entry can be included in the same list. The types of entry are:
  - Entries specifying a particular terminal (or logical unit)

identifier but no operator identifier. Each specified terminal will receive the message.

- Entries specifying a particular terminal (or logical unit) identifier and an operator identifier. Each specified terminal will receive the message if or when the specified operator is signed on at the terminal.

- Entries specifying only an operator identifier. Each specified operator must be signed on at a terminal supported by BMS when the ROUTE command is issued; otherwise the route list entry for that operator is ignored (skipped). CICS/VS will then schedule the message for delivery to each terminal at which a specified operator is signed on. If a particular operator is signed on at more than one terminal, CICS/VS will schedule the message for delivery to the one whose entry appears first in the terminal control table. Each terminal for which the message is scheduled will then receive the message (when it is ready for delivery) if the specified operator is still signed on at the terminal or when the operator signs on again.

- LIST is omitted but OPCLASS is specified. CICS/VS will schedule the message for delivery to all terminals at which an operator having at least one of the specified operator class codes is signed on when the ROUTE command is issued. Each terminal for which the message is scheduled will then receive the message (when it is ready for delivery) if or when an operator (not necessarily the same one as before) having at least one of the specified operator class codes is signed on at the terminal.

- LIST and OPCLASS are both specified. The effect of the OPCLASS specification for the different types of route list entries is as follows:

  - Entries specifying no operator identifier. The effect is the same as if only the OPCLASS option were specified, but is restricted to those terminals (or logical units) specified in the route list.

  - Entries specifying an operator identifier (and possibly a terminal or logical unit identifier). The OPCLASS specification is ignored for these route list entries, and the effect is the same as if only the LIST option were specified.

Route List Format. The route list specified in the LIST option must conform to a fixed format. The list consists of 16-byte entries (with contents as shown in the following table). The end of the list is designated by a binary halfword initialized to -1.

Bytes	Contents
0-3	Terminal or logical unit identifier (four-characters, including trailing blanks), or blanks
4,5	LDC mnemonic (two-characters) for logical units with LDC support, or blanks
6-8	Operator identifier, or blanks
9	Status flag for the route entry
10-15	Reserved; must contain blanks

The status flag byte indicates to the application program the status of the destination when the ROUTE command is issued. Upon return, the application program can investigate the status flag byte for each entry and take appropriate action. The status flag byte settings and their meanings are as follows:

ENTRY SKIPPED
A route list entry was excluded. If an entry has been excluded, another flag indicating why the entry was skipped may be on in the status byte. This second flag could be any of the other flags shown in the table. If the OPERATOR NOT SIGNED ON flag is on, only an operator identifier was specified in the route list entry and the specified operator was not signed on at any terminal. If only the ENTRY SKIPPED flag is on, neither a terminal identifier nor an operator identifier was specified in the route list entry. The settings are X'80' for ASM, 12-0-1-8 for COBOL, and 10000000 for PL/I.

INVALID TERMINAL IDENTIFIER
indicates that the terminal identifier specified in the route list entry does not have a corresponding TCTTE in the terminal control table. This entry is also flagged as ENTRY SKIPPED. The settings are X'40' for ASM, no punches for COBOL, and 01000000 for PL/I.

TERMINAL NOT SUPPORTED UNDER BMS
indicates that the terminal identifier specified in the route list entry is for a type of terminal that is not supported under BMS; or the terminal table entry indicated that the terminal was not eligible for routing. This entry is also flagged as ENTRY SKIPPED. The settings are X'20' for ASM 11-0-1-8-9 for COBOL, and 00100000 for PL/I.

OPERATOR NOT SIGNED ON
indicates that the specified operator is not signed on. Any one of the following conditions causes this flag to be set:

- Both an operator identifier and a terminal identifier were specified, and the specified operator was not signed on at the terminal. This entry is not skipped.
- An operator identifier was specified without a terminal identifier, and the operator was not signed on at any terminal. This entry is also flagged as ENTRY SKIPPED.
- The OPCLASS option was specified with the ROUTE command and a terminal identifier was specified in the route list entry, but the operator signed on at the terminal did not have any of the specified operator classes. This entry is not skipped.

The settings are X'10' for ASM, 12-11-1-8-9 for COBOL, and 00010000 for PL/I.

OPERATOR SIGNED ON AT UNSUPPORTED TERMINAL
indicates that only an operator identifier was specified in the route list entry, and that operator was signed on a terminal not supported by BMS. This entry is also flagged as ENTRY SKIPPED. The unsupported terminal identifier is returned in that route list entry at URLTRMID, defined in DFHURLDS (described below). The settings are X'08' for ASM, 12-8-9 for COBOL, and 00001000 for PL/I.

INVALID LDC MNEMONIC
indicates that one of the following situations exists:

- The LDC mnemonic specified in the route list does not appear in the LDC list associated with the TCT.
- The device type generated in the system LDC table for the specified or implied LDC mnemonic is not the same as the device type for the first LDC specified in the route environment.

The settings are X'04' for ASM, 12-4-9 for COBOL, and 00000100 for PL/I.

A symbolic storage definition of the user-supplied route list is available in the source library (or libraries) under the member name DFHURLDS. This definition can be used as an aid in building the route list, and if necessary, in testing the status flag byte for each entry upon return from a ROUTE command that refers to a list.

The list can be supplied in noncontiguous areas called segments, in which case every segment except the last is terminated with (at least) an eight-byte entry with contents as shown in the following table. The last segment ends with a binary halfword initialized to -1.

Bytes	Contents
0,1	ASM: binary halfword initialized to -2 COBOL: PIC S9(4) COMP VALUE -2 PL/I: DCL FIXED BIN(15) INIT(-2)
2,3	Reserved
4-7	Chain address to the first entry of the next segment

BASIC MAPPING SUPPORT OPTIONS

ACCUM
specifies that this command is one of a number of commands that are used to build a logical message. The logical message is completed by a SEND PAGE command. This option is mutually exclusive with NOEDIT.

ALARM
specifies that the 3270 audible alarm feature is to be activated. For logical units supporting FMHs (except interactive and batch logical units), ALARM signals BMS to set the alarm flag in the FMH.

ALL
specifies that if the ATTN key on a 2741 is pressed while data is being sent to the terminal and the WRBRK condition is not active, transmission of the current page is to cease and no additional pages are to be transmitted. The logical message is deleted.

ASIS
specifies that the specification FEATURE=UCTRAN in the terminal control table for the terminal is to be overridden. Lowercase characters in the data stream are not translated to uppercase.

AUTOPAGE
specifies that each page of the logical message is to be sent to the terminal as soon as it is available. If paging upon request is specified for the terminal at system generation, AUTOPAGE overrides it for this logical message.

AUTOPAGE is assumed for 3270 printers; it does not apply to 3270 display terminals. If neither AUTOPAGE nor NOAUTOPAGE is specified, the terminal has the paging status specified for it at CICS/VS system generation.

CURRENT
specifies that if the ATTN key on a 2741 is pressed while data is being sent to the terminal and the WRBRK condition is not active, transmission of the current page is to cease and transmission of the next page (if any) is to begin.

CURSOR[(data-value)]
specifies the position to which the 3270 or 3604 cursor is to be returned upon completion of a send 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 (described earlier in the chapter) is assumed.

This option overrides the IC option of the ATTRB operand of the DFHMDF macro instruction, if it is specified in a command that completes a page-building operation and thus causes a send operation. Previous specifications of the IC option and of the CURSOR option for the other maps making up the page are ignored.

DATAONLY
specifies that only application-program data is to be written. The attribute characters (3270 only) must be specified for each field in the supplied data. If the attribute byte in the user-supplied data is set to X'00', the attribute byte on the screen will be unchanged. Any default data or attributes from the map are ignored.

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. (This option applies only to the 3270 and to the 3604 Keyboard Display.) The first output operation in any transaction, or in a series of pseudo-conversational transactions, should always specify ERASE. For transactions attached to 3278 screens, this will also ensure that the correct screen size is selected, as defined for the transaction in the PCT.

ERASEAUP
specifies that before this page of output is displayed, all unprotected character locations are to be erased. (This option applies only to the 3270.)

ERRTERM[(name)]
specifies the name of the terminal to be notified if the message is deleted because it is undeliverable. The message number, title identification, and destination are indicated. If no name is specified, the originating terminal is assumed.

This option is operative only if the PRGDLAY operand has been specified in the DFHSG PROGRAM=BMS system macro.

FMHPARM(name)
specifies the name (1 through 8 characters) of the outboard map to be used. (This option applies only to 3650 logical units with outboard formatting).

FREEKB
specifies that the 3270 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 the data area provided in a SEND MAP command has not been generated by the BMS map definition process, it must start with a 12-byte TIOA prefix. If FROM is specified, the MAPONLY option must not be specified. If FROM 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".

The data area provided in a RECEIVE MAP command should not include a TIOA prefix.

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.

This allows the ATTRB operand of the DFHMDF macro for the requested map to control the final status of fields written or rewritten in response to a BMS command.

HEADER(data-value)
specifies the header data to be placed at the beginning of each page. The format of the header is described under "Format Output Data without Mapping (SEND TEXT)" earlier in this chapter.

HONEOM
specifies that the default printer line length is to be used. This length should be the same as that specified in the PGESIZE operand of the DFHTCT TYPE=TERMINAL system macro, otherwise the data may not format correctly.

INTERVAL(hhmmss)
specifies the interval of time after which the data is to be transmitted to the terminals specified in the ROUTE command.

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.

JUSTIFY(data-value)
specifies the line of the page at which the data is to be positioned. The data value must be a halfword binary value in the range 1 through 240. Although they may not be specified as constants, the special values -1 and -2 can be supplied dynamically to signify JUSFIRST or JUSLAST, respectively.

JUSFIRST
specifies that the data is to be placed at the top of the page. Any partially formatted page from previous requests is considered to be complete. If the HEADER option is specified, the header precedes the data. See also the description of the JUSTIFY option.

JUSLAST
specifies that the data is to be positioned at the bottom of the page. The page is considered to be complete after the request has been processed. If the TRAILER option is specified, the trailer follows the data. See also the description of the JUSTIFY option.

LAST
specifies that this is the last output operation for a transaction and, therefore, the end of a bracket. If the RELEASE option is specified, LAST is assumed unless the SEND PAGE command is terminating a routing operation. (This option applies to logical units only.)

LDC(name)
specifies a two-character mnemonic to be used to determine the logical device code (LDC) to be transmitted in the FMH to the logical unit. The mnemonic represents an LDC entry specified in the DFHTCT TYPE=LDC system macro.

When an LDC is specified, BMS uses the device type, the page size, and the page status associated with the LDC mnemonic to format the message. These values are taken from the extended local LDC table for the LU, if it has one. If the LU has only a local (unextended) LDC table, the values are taken from the system LDC table. The numeric value of the LDC is obtained from the local LDC table, unless this is an unextended table and the value is not specified, in which case it is taken from the system table.

If the LDC option of a SEND MAP or ROUTE command is omitted, the LDC mnemonic specified in the DFHMSD macro is used. If the LDC option has also been omitted from the DFHMSD macro, the action depends on the type of logical unit, as follows:

3601 LU - the first entry in the local or extended local LDC table is used, if there is one. If a default cannot be obtained in this way, a null LDC numeric value (X'00') is used. The page size used is the value that is specified in the DFHTCT TYPE=TERMINAL system macro, or (1,40) if such a value is not specified.

LUTYPE4 LU, batch LU, or batch data interchange LU - the local LDC table is not used to supply a default LDC; instead, the message is directed to the LU console (that is, to any

medium that the LU elects to receive such messages. For a batch data interchange LU, this does not imply sending an LDC in an FMH). The page size is obtained in the manner described for the 3601 LU.

For message routing, the LDC option of the ROUTE command takes precedence over all other sources. If this option is omitted and a route list is specified (LIST option), the LDC mnemonic in the route list is used; if the route list contains no LDC mnemonic, or no route list is specified, a default LDC is chosen as described above.

LENGTH(data-value)
specifies the length of the data to be formatted as a halfword binary value.

LIST(data-area)
specifies the data area that contains a list of terminals and/or operators to which data is to be directed. If this option is omitted, all terminals supported by BMS receive the data (unless the OPCLASS option has some effect). The format of the list is described under "Route List and Operator Class Codes" earlier in this chapter.

L40
specifies the line length for a 3270 printer; a carrier return and line feed are forced after 40 characters have been printed on a line.

L64
specifies the line length for a 3270 printer; a carrier return and line feed are forced after 64 characters have been printed on a line.

L80
specifies the line length for a 3270 printer; a carrier return and line feed are forced after 80 characters have been printed on a line.

MAP(name)
specifies the name (1 through 7 characters) of the map to be used.

MAPONLY
specifies that only default data from the map is to be written. If this option is specified, the FROM option must not be specified.

MAPSET(name)
specifies the name (1 through 7 characters) 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.

This option should be used always unless a reference is made to pre-VS BMS maps, which were loaded one at a time, rather than as a set, and whose names were not extended by a terminal-type suffix.

NLEOM
specifies that data for a 3270 printer or a 3275 display with the printer adapter feature should be built with new-line (NL) characters, and that an end-of-message (EM) character should be placed at the end of the data. As the data is printed, each NL character causes printing to continue on the next line, and the EM character terminates printing.

This option must be specified in the first SEND MAP or SEND TEXT command used to build a logical message, and in the ROUTE command if the message is to be routed. The option is ignored if the device receiving the message (direct or routed) is not one of those noted above.

If this option is used, buffer updating and attribute modification of fields previously written into the buffer are not allowed. CICS/VS includes the ERASE option with every write to the terminal.

The NL character occupies a buffer position. A number of buffer positions, equivalent to the value of the PGESIZE operand of the DFHTCT system macro for that terminal, is unavailable for data. This may cause data to wrap around in the buffer; if this occurs, the PGESIZE value must be reduced.

NOAUTOPAGE
specifies that pages are to be sent one at a time to the terminal. BMS sends the first page to the terminal when the terminal becomes available or upon request of the operator. Subsequent pages are sent to the terminal in response to requests from the terminal operator. (Refer to the CICS/VS Operator's Guide.)

If automatic paging is specified for the terminal at system generation, NOAUTOPAGE overrides it for this logical message. For logical units, NOAUTOPAGE applies to all LDC page sets accumulated within the logical message.

NOAUTOPAGE does not apply to 3270 printers.

NOEDIT
specifies that the application program, as opposed to CICS/VS,

controls the insertion of device-dependent control characters (carrier return, line feed, idle, and so on) into the output data stream. This option is mutually exclusive with ACCUM. This option cannot be used with 3601 devices.

OPCLASS(data-area)
specifies the data area that contains a list of operator classes to which the data is to be routed. The classes are supplied in a three-byte field, each bit position corresponding to one of the codes in the range 1 through 24 but in reverse order, that is, the first byte corresponds to codes 24 through 17, the second byte to codes 16 through 9, and the third byte to codes 8 through 1.

OPERPURGE
specifies that CICS/VS is to delete the message only when the terminal operator requests deletion. If the option is omitted, CICS/VS deletes the message if the operator enters a transaction that is not a paging command.

PAGING
specifies that the output data is not to be sent immediately to the terminal, but is to be placed in temporary storage and displayed in response to paging commands entered by the terminal operator.

If PAGING is specified with a REQID that is defined in the temporary storage table (TST), CICS/VS provides message recovery for logical messages if the task has reached logical end.

PRINT
specifies that a print operation is to be started at a 3270 printer or at a 3275 with the printer adapter feature, or that data on an LUTYPE2 (3274/76 or 3790) is to be printed on a printer allocated by the controller. If this option is omitted, the data is sent to the printer buffer but is not printed.

RELEASE
specifies that control is to be returned to the program at the next higher logical level, or to CICS/VS (if the issuing program is at the highest logical level), after the pages have been written to the terminal. For more details of the effect of this option, refer to the description of the SEND PAGE command earlier in the chapter.

REQID(name)
specifies a two-character prefix to be used as part of a temporary storage identifier for CICS/VS message recovery. Only one prefix can be specified for each logical message. The default prefix is **.

BMS message recovery is provided for a logical message only if the PAGING option is specified in the BMS output command and if the logical end of task has been reached.

RETAIN
specifies that control is to be returned to the application program after the pages have been written to the terminal. For more details of the effect of this option, refer to the description of the SEND PAGE command earlier in the chapter.

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

For input, the pointer is set to the address of the mapped data.

For output, the SET option specifies that the completed pages are to be returned to the application program. The pointer is set to the address of a list of completed pages. For the format of the list, refer to "Output Requests with the SET Option" earlier in this chapter.

The application program regains control either immediately following the BMS command (if the current page is not yet completed), or at an alternative entry point specified through a HANDLE CONDITION RETPAGE command (if one or more pages have been completed).

TERMINAL
specifies that input data is to be read from the terminal that originated the transaction, or that output data is to be sent to that terminal when the page is completed.

TIME(hhmmss)
specifies the time of day at which data is to be transmitted to the terminals specified in the ROUTE command.

TITLE(data-area)
specifies the data area that contains the title to be used with the logical message. For the format of the title, refer to "Routing Messages (ROUTE)" earlier in this chapter.

TRAILER(data-area)
specifies the data area that contains trailer data to be placed

at the bottom of each output page (with a SEND TEXT command) or at the bottom of the last page only (with a SEND PAGE command). For the format of the trailer data, refer to "Formatting Output Data Without Mapping (SEND TEXT)" earlier in this chapter.

TRANSID(name)
specifies the transaction identifier to be used with the next input message from the terminal to which the task is attached. The identifier can consist of up to four alphameric characters; it must have been defined in the program control table (PCT). TRANSID is valid only if RELEASE is specified.

If this option is specified in a program that is not at the highest logical level, the specified transaction identifier will be used only if a new transaction identifier is not provided in another SEND PAGE command (or in a RETURN program control command) issued in a program at a higher logical level.

WAIT
specifies that control should not be returned to the application program until the output operation has been completed.

If WAIT is not specified, control will return to the application program once the output operation has started. A subsequent input or output command (terminal control, BMS, or batch data interchange) will cause the application program to wait until the previous command has been completed.

BASIC MAPPING SUPPORT EXCEPTIONAL CONDITIONS

Some of the following exceptional conditions may occur in combination with others. CICS/VS checks for these conditions in the following order: TSIOERR, INVREQ, RETPAGE, MAPFAIL, RTEFAIL, INVERRTERM, INVMPSZ. If more than one of these conditions occurs, only the first one found to be present is passed to the application program.

EOC
occurs if the request/response unit (RU) is received with the end-of-chain (EOC) indicator set. It applies only to logical units.

Default action: ignore the condition.

EODS
occurs if no data is received (only an FMH). It applies only to 3770 batch logical units and to 3770 and 3790 batch data interchange logical units.

Default action: terminate the task abnormally.

IGREQCD
occurs when an attempt is made to execute a SEND MAP, SEND PAGE, or SEND TEXT command after a SIGNAL data-flow control command with an RCD (request change direction) code has been received from an LUTYPE4 logical unit.

Default action: terminate the task abnormally.

IGREQID
occurs if the prefix specified in the REQID option is different from that established by a previous REQID option or by default for this logical message.

Default action: terminate the task abnormally.

INVERRTERM
occurs if the terminal identifier specified in the ERRTERM option of a ROUTE command is invalid or is assigned to a type of terminal not supported by BMS.

Default action: terminate the task abnormally.

INVLDC
occurs if the specified LDC mnemonic is not included in the LDC list for the logical unit.

Default action: terminate the task abnormally.

INVMPSZ
occurs if the specified map is too wide for the terminal, or if a HANDLE CONDITION OVERFLOW command is active and the specified map is too long for the terminal.

Default action: terminate the task abnormally.

INVREQ
occurs if a request for BMS services is invalid for any of the following reasons:

- The disposition of a routed message is changed prior to its completion by a SEND PAGE command.
- A separate SEND TEXT ACCUM or SEND MAP ACCUM command is issued to the terminal that originated the transaction while a routed logical message is being built.

- The TRAILER option is specified in a SEND PAGE command when terminating a logical message built with SEND MAP commands.
- An output mapping command is issued for a map without field specifications by specifying the FROM option without the DATAONLY option.

Default action: terminate the task abnormally.

MAPFAIL
occurs if the data to be mapped has a length of zero or does not contain a set-buffer-address (SBA) sequence. It applies only to 3270 devices. The receiving data area will contain the unmapped input data stream. The amount of unmapped data moved to the user's area will be limited to the length specified in the LENGTH option of the RECEIVE MAP command.

Default action: terminate the task abnormally.

OVERFLOW
occurs if the mapped data does not fit on the current page.

Default action: ignore the condition.

RDATT
occurs if a RECEIVE MAP command is terminated by the operator using the ATTN key rather than the RETURN key. It applies only to the 2741 Communications Terminal, and only if 2741 read attention support has been generated for CICS/VS.

Default action: ignore the condition.

RETPAGE
occurs if the SET option is specified and one or more completed pages are ready for return to the application program.

Default action: return to the application program at the point immediately following the BMS command.

RTEFAIL
occurs if a ROUTE command would result in the message being sent only to the terminal that initiated the transaction.

Default action: return to the application program at the point immediately following the ROUTE command.

RTESOME
occurs if any of the terminals specified by options of a ROUTE command will not receive the message.

Default action: return control to the application program at the point immediately following the ROUTE command.

TSIOERR
occurs if there is an unrecoverable temporary storage input/output error.

Default action: terminate the task abnormally.

WRBRK
occurs if a SEND command is interrupted by the terminal operator pressing the ATTN key. It applies only to the 2741 Communication Terminal under OS/VS, and only if write break support has been generated for CICS/VS.

Default action: ignore the condition.

Chapter 3.4. Batch Data Interchange

The CICS/VS batch data interchange program provides for communication between an application program and a named data set (or destination) that is part of a batch data interchange logical unit in an outboard controller, or with a selected medium on a batch logical unit or an LUTYPE4 logical unit.

The term "outboard controller" is a generalized reference to a programmable subsystem, such as the IBM 3770 Data Communication System or the IBM 3790 Data Communication System, which uses SNA protocols. (Details of SNA protocols and the data sets that can be used are given in the publications CICS/VS IBM 3767, 3770, and 6670 Guide and CICS/VS IBM 3790 Guide.)

Batch data interchange commands are provided to:

- Interrogate a data set (ISSUE QUERY).
- Read a record from a data set or read data from an input medium (ISSUE RECEIVE).
- Add a record to a data set (ISSUE ADD).
- Update (replace) a record in a data set (ISSUE REPLACE).
- Delete a record in a data set (ISSUE ERASE).
- Terminate processing of a data set (ISSUE END).
- Terminate processing of a data set abnormally (ISSUE ABORT).
- Request the next record number in a data set (ISSUE NOTE).
- Wait for an operation to be completed (ISSUE WAIT).
- Transmit data to a named data set or to a selected medium (ISSUE SEND).

Where the controller is an LUTYPE4 logical unit, only the ISSUE ABORT, ISSUE END, ISSUE RECEIVE, ISSUE SEND, and ISSUE WAIT commands can be used.

The HANDLE CONDITION command is used to deal with any exceptional conditions that occur during execution of a batch data interchange command. Refer to "Chapter 1.5. Exceptional Conditions" on page 25 for further information about exceptional conditions.

DESTINATION SELECTION AND IDENTIFICATION

All batch data interchange commands except ISSUE RECEIVE include options that specify the destination. This is either a named data set in a batch data interchange logical unit, or a selected medium in a batch logical unit or LUTYPE4 logical unit.

Selection by Named Data Set: The DESTID and DESTIDLENG options must always be specified, to supply the data set name and its length (up to a maximum of eight characters). For destinations having diskettes, the VOLUME and VOLUMELENG options may be specified, to supply a volume name and its length (up to a maximum of six characters); the volume name identifies the diskette that contains the data set to be used in the operation. If the VOLUME option is not specified for a multi-diskette destination, all diskettes are searched until the required data set is found.

Selection by Medium: As an alternative to naming a data set as the destination, various media can be specified by means of the CONSOLE, PRINT, CARD, or WPMEDIA1-4 options. These media can be specified only in an ISSUE ABORT, ISSUE END, ISSUE SEND, or ISSUE WAIT command.

DEFINITE-RESPONSE

CICS/VS uses terminal control commands to carry out the functions specified in batch data interchange commands. For those commands that cause terminal control output requests to be made, the DEFRESP option can be specified. This option has the same effect as the DEFRESP option of the SEND terminal control command; that is, to request a definite response from the outboard controller, irrespective of the specification of message integrity for the CICS/VS task (by the system programmer). The DEFRESP option can be specified for the ISSUE ADD, ISSUE ERASE, ISSUE REPLACE, and ISSUE SEND commands.

WAITING FOR FUNCTION COMPLETION

For those batch data interchange commands that cause terminal control output requests to be made, the NOWAIT option can be specified also. This option has the effect of allowing CICS/VS task processing to continue; unless the option is specified, task activity is suspended until the batch data interchange command is completed. The NOWAIT option can be

specified for the ISSUE ADD, ISSUE ERASE, ISSUE REPLACE, and ISSUE SEND commands.

After a batch data interchange command with the NOWAIT option has been issued, task activity can be suspended, by the ISSUE WAIT command, at a suitable point in the program to wait for the command to be completed.

INTERROGATE A DATA SET (ISSUE QUERY)

```
ISSUE QUERY DESTID(data-value)
DESTIDLENG(data-value)
[VOLUME(data-value)
 VOLUMELENG(data-value)]

Conditions:  FUNCERR, SELNERR,
UNEXPIN
```

This command is used to request that a sequential data set in an outboard controller be transmitted to the host system. The application program should either follow this command with ISSUE RECEIVE commands to obtain the resulting inbound data, or terminate the transaction to allow CICS/VS to start a new transaction to process the data.

READ A RECORD FROM A DATA SET (ISSUE RECEIVE)

```
ISSUE RECEIVE {SET(ptr-ref)|
INTO(data-area)}
LENGTH(data-area)

Conditions:  DSSTAT, EODS, LENGERR,
NODATARECD, UNEXPIN
```

This command is used to read a record from an outboard controller. The INTO option specifies the area into which the data is to be placed. The LENGTH option must include 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 that area. After the retrieval operation, the data area specified in the LENGTH option is set to the record length.

The outboard controller might not send the data from the data set specified in the ISSUE QUERY command. An ASSIGN command must be used to obtain the value of DESTID, which identifies the data set that has actually been transmitted; also the value of DESTIDLENG, which is the length of the identifier in DESTID.

ADD A RECORD TO A DATA SET (ISSUE ADD)

```
ISSUE ADD DESTID(data-value)
DESTIDLENG(data-value)
[VOLUME(data-value)|
VOLUMELENG(data-value)]
FROM(data-area)
LENGTH(data-value)
[NUMREC(data-value)]
[DEFRESP]
[NOWAIT]

Conditions:  FUNCERR, SELNERR,
UNEXPIN
```

This command is used to add records to a sequential or keyed direct data set in an outboard controller. The FROM option is used to specify the data to be written, and the LENGTH option specifies its length.

The RIDFLD option is not needed with this command; the key is embedded in the data.

UPDATE A RECORD IN A DATA SET (ISSUE REPLACE)

```
ISSUE REPLACE DESTID(data-value)
DESTIDLENG(data-value)
[VOLUME(data-value)|
 VOLUMELENG(data-value)]
FROM(data-area)
LENGTH(data-value)
RIDFLD(data-area)
[DEFRESP]
[NOWAIT]
[KEYLENGTH(data-value)|RRN]
[NUMREC(data-value)]

Conditions: FUNCERR, SELNERR,
UNEXPIN
```

This command is used to replace (update) a record in either a relative (addressed direct) or an indexed (keyed direct) data set in an outboard controller.

The FROM option is used to specify the data to be written to the data set and the LENGTH option specifies the length of the data.

The RIDFLD option specifies the relative record number of the first record to be replaced for a relative data set, or the

embedded key in the data specified by the FROM option for an indexed data set.

For a relative data set, the RRN option must be specified since the RIDFLD option contains a relative record number. In addition, the NUMREC option must specify the number of records to be replaced consecutively, starting with the one specified in RIDFLD.

For an indexed data set, the RIDFLD option specifies the key embedded in the data specified in the FROM option. In addition, the KEYLENGTH option must specify the length of the key. The NUMREC option cannot be specified since only one record is replaced.

DELETE A RECORD FROM A DATA SET (ISSUE ERASE)

```
ISSUE ERASE DESTID(data-value)
DESTIDLENG(data-value)
[VOLUME(data-value)
 VOLUMELENG(data-value)]
RIDFLD(data-area)
[KEYLENGTH(data-value)|RRN]
[NUMREC(data-value)]
[DEFRESP]
[NOWAIT]

Conditions:  FUNCERR, SELNERR,
UNEXPIN
```

This command is used to delete a record from a keyed direct data set in an outboard controller. The RIDFLD option specifies the key of the record to be deleted; the length of the key must be specified in the KEYLENGTH option.

TERMINATE PROCESSING OF A DATA SET (ISSUE END)

```
ISSUE END [DESTID(data-value)
 DESTIDLENG(data-value)]
[VOLUME(data-value)
 VOLUMELENG(data-value)]
[SUBADDR(data-value)]
[CONSOLE|PRINT|CARD|WPMEDIA1|
  WPMEDIA2|WPMEDIA3|WPMEDIA4]

Conditions:  FUNCERR, SELNERR,
UNEXPIN
```

This command is used to terminate communication with a data set in an outboard controller or with the selected medium. The data set specified in the DESTID option, or the selected medium, is de-selected normally. The options CONSOLE, PRINT, CARD, WPMEDIA1-4 are alternatives to DESTID and DESTIDLENG.

TERMINATE PROCESSING OF A DATA SET ABNORMALLY (ISSUE ABORT)

```
ISSUE ABORT [DESTID(data-value)
 DESTIDLENG(data-value)]
[VOLUME(data-value)
 VOLUMELENG(data-value)]
[SUBADDR(data-value)]
[CONSOLE|PRINT|CARD|WPMEDIA1|
  WPMEDIA2|WPMEDIA3|WPMEDIA4]

Conditions:  FUNCERR, SELNERR,
UNEXPIN
```

This command is used to terminate communication with a data set in an outboard controller, or with the selected medium, abnormally. The data set specified in the DESTID option is de-selected abnormally. The options CONSOLE, PRINT, CARD, WPMEDIA1-4 are alternatives to DESTID and DESTIDLENG.

TRANSMIT DATA TO AN OUTPUT DEVICE (ISSUE SEND)

```
ISSUE SEND  [DESTID(data-value)
 DESTIDLENG(data-value)]
[VOLUME(data-value)
 VOLUMELENG(data-value)]
FROM(data-area)
LENGTH(data-value)
[SUBADDR(data-value)]
[CONSOLE|PRINT|CARD|WPMEDIA1|
  WPMEDIA2|WPMEDIA3|WPMEDIA4]
[NOWAIT]
[DEFRESP]

Conditions:  FUNCERR, IGREQCD,
SELNERR, UNEXPIN
```

This command is used to transmit data to a named data set in an outboard controller, or to a selected medium in a batch logical unit or an LUTYPE4 logical unit. The options CONSOLE, PRINT, CARD, WPMEDIA1-4 are alternatives to DESTID and DESTIDLENG.

REQUEST NEXT RECORD NUMBER (ISSUE NOTE)

```
ISSUE NOTE DESTID(data-value)
 DESTIDLENG(data-value)
[VOLUME(data-value)
  VOLUMELENG(data-value)]
RIDFLD(data-area)
[RRN]

Conditions:  FUNCERR, SELNERR,
UNEXPIN
```

This command is used to find the relative record number of the next record in an addressed direct data set. The number is returned in the data area specified in the RIDFLD option. The RRN option must be specified, because a relative record number is involved.

WAIT FOR AN OPERATION TO BE COMPLETED (ISSUE WAIT)

```
ISSUE WAIT [DESTID(data-value)
 DESTIDLENG(data-value) ]
[VOLUME(data-value)
  VOLUMELENG(data-value)]
[SUBADDR(data-value)]
[CONSOLE|PRINT|CARD|WPMEDIA1|
  WPMEDIA2|WPMEDIA3|WPMEDIA4]

Conditions:  FUNCERR, SELNERR,
UNEXPIN
```

This command is used to cause task activity to be suspended until the previous batch data interchange command is completed. This command is meaningful only when it follows an ISSUE ADD, ISSUE ERASE, ISSUE REPLACE, or ISSUE SEND command. The options CONSOLE, PRINT, CARD, WPMEDIA1-4 are alternatives to DESTID and DESTIDLENG.

BATCH DATA INTERCHANGE OPTIONS

CARD
specifies that the output medium is a card reader/punch device. This option is not valid with DESTID and DESTIDLENG.

CONSOLE
specifies that the output medium is that provided for messages to the operator. This option is not valid with DESTID and DESTIDLENG.

DEFRESP
specifies that all terminal control commands issued as a result of the batch data interchange command will request a definite response from the outboard batch program, irrespective of the specification of message integrity for the CICS/VS task (by the system programmer).

DESTID(data-value)
specifies the name of the data set in the outboard destination. The data value must be a character string of up to eight characters. This option is not valid with CONSOLE, CARD, PRINT, or WPMEDIA1-4.

DESTIDLENG(data-value)
specifies the length of the name specified in the DESTID option as a halfword binary value. This option is not valid with CONSOLE, CARD, PRINT, or WPMEDIA1-4.

DFTPROF
specifies that the default data stream profile has been specified.

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

INTO(data-area)
specifies the receiving field for the data read from the data set. The INTO option implies move-mode access.

KEYLENGTH(data-value)
specifies the length of the key specified in the RIDFLD option as a halfword binary value.

LENGTH(parameter)
specifies a halfword binary value to be used with ISSUE ADD, ISSUE RECEIVE, ISSUE REPLACE, and ISSUE SEND commands.

For an ISSUE ADD, ISSUE REPLACE, or ISSUE SEND1. command, the parameter must be a data value that is the length of the data that is to be written.

For an ISSUE RECEIVE 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 an ISSUE RECEIVE 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.

NOWAIT
specifies that the CICS/VS task will continue processing without waiting for the batch data interchange command to complete. If this option is not specified, the task activity will be suspended until the command is completed.

NUMREC(data-value)
for a relative data set, specifies as a halfword binary value the

number of logical records affected by one ISSUE REPLACE command. Records are replaced sequentially starting with the one identified by the RIDFLD option.

For an indexed data set, NUMREC cannot be specified since only one record is replaced.

PRINT
specifies that the output is to the print medium.

RIDFLD(data-area)
specifies the record identification field for use with ISSUE REPLACE and ISSUE ERASE commands; it also specifies a data area in which the relative record number of the next record is returned in an ISSUE NOTE command.

For ISSUE REPLACE commands for a relative data set, the RIDFLD option must specify a fullword binary integer being the relative record number (starting from zero) of the record. The RRN option is also required.

For ISSUE REPLACE commands for an indexed data set, the RIDFLD option specifies the key which is embedded in the data specified by the FROM option. The KEYLENGTH option is also required.

For ISSUE ERASE commands, the RIDFLD option must specify the key of the record.

RRN
specifies that the record identification field specified in the RIDFLD option is a relative record number. If the option is not specified, RIDFLD is assumed to specify a key.

SET(ptr-ref)
specifies the pointer reference that is to be set to the address location of the data read from the data set. The SET option implies locate-mode access.

SUBADDR(data-value)
specifies the medium subaddress as a decimal value (in the range 0 through 15) which allows media of the same type, for example, "printer 1" or "printer 2", to be defined. Value 15 means a medium of any type. The default is 00.

VOLUME(data-value)
specifies the name of a diskette in an outboard destination that contains the data set specified in the DESTID option. The data value must be a character string of up to six characters.

VOLUMELENG(data-value)
specifies the length of the name specified in the VOLUME option as a halfword binary value.

WPMEDIA1 through WPMEDIA4
specifies that for each specific LUTYPE4 device, a word processing medium is defined to relate to a specific input/output device.

BATCH DATA INTERCHANGE EXCEPTIONAL CONDITIONS

DSSTAT
occurs when the destination status changes in one of the following ways:

- The data stream is aborted.
- The data stream is suspended.

Default action: terminate the task abnormally.

EODS
occurs when the end of the data set is encountered.

Default action: terminate the task abnormally.

IGREQCD
occurs when an attempt is made to execute an ISSUE SEND command after a SIGNAL RCD data-flow control code has been received from an LUTYPE4 logical unit.

Default action: terminate the task abnormally.

FUNCERR
occurs when an error occurs during execution of the command. Destination selection is unaffected and other commands for the same destination may be successful.

Default action: terminate the task abnormally.

LENGERR
occurs if the length of the retrieved data is greater than the value specified by the LENGTH option for a move-mode ISSUE RECEIVE command.

Default action: terminate the task abnormally.

NODATARECD
occurs if an ISSUE RECEIVE command is issued to an LUTYPE4 logical unit and the destination currently has no data to send.

Default action: terminate the task abnormally.

SELNERR
occurs when an error occurs during destination selection. The destination is not selected and other commands for the same destination are unlikely to be successful.

Default action: terminate the task abnormally.

UNEXPIN
occurs when some unexpected or unrecognized information is received from the outboard controller.

Default action: terminate the task abnormally.

More detailed information about the cause of an exceptional condition is given in field EIBRCODE in the EIB which is shown in "Appendix A. EXEC Interface Block" on page 239. (Refer also to the CICS/VS Problem Determination Guide.)

Part 4. Control Operations

Chapter 4.1. Introduction to Control Operations

This part of the manual collects together several groups of operations that are not specifically data base or data communication operations, but that control the execution of tasks within a CICS/VS system. These groups of operations are as follows:

- Interval control - comprising functions whose execution is dependent on time.
- Task control - comprising functions to synchronize task activity or resource usage.
- Program control - comprising functions affecting the flow of control between application programs.
- Storage control - comprising functions to obtain and release areas of main storage.
- Transient data control - comprising functions for the transfer of data between CICS/VS tasks and between the CICS/VS region or partition and other regions or partitions.
- Temporary storage control - comprising functions for the temporary storage of data.

Each group of operations is described in a separate chapter within this part, as listed on the previous page.

Chapter 4.2. Interval Control

The CICS/VS interval control program, in conjunction with a time-of-day clock maintained by CICS/VS, provides functions that 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 task mix and the frequency of task switching operations.

Interval control commands are provided to:

- Request the current date and time of day (ASKTIME).
- Delay the processing of a task (DELAY).
- Request notification when a specified time has expired (POST).
- Wait for an event to occur (WAIT EVENT).
- Start a task and store data for the task (START).
- Retrieve data stored (by a start command) for a task (RETRIEVE).
- Cancel the effect of previous interval control commands (CANCEL).

Exceptional conditions that occur during execution of an interval control command are handled as described in "Chapter 1.5. Exceptional Conditions" on page 25.

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.

An interval is measured relative to the current time and so the expiry time will always be after the current time (assuming a nonzero interval is specified). An absolute time is measured relative to midnight prior to the current time and may therefore be prior to the current time.

CICS/VS treats as expired a request for an absolute time that is equal to the current time or that precedes the current time by up to six hours. If the specified absolute time precedes the current time by more than six hours, CICS/VS adds 24 hours, that is, the requested function is performed at the time specified but on the next day.

Examples of the START command specifying absolute time-of-day requests, are as follows:

EXEC CICS START TIME (123000) issued at 1000 hours on Monday will expire at 1230 hours on the same Monday.

EXEC CICS START TIME (090000) issued at 1000 hours on Monday will expire immediately because the specified time is within the preceding six hours.

EXEC CICS START TIME (020000) issued at 1000 hours on Monday will expire at 0200 hours on Tuesday because the specified time is more than six hours before the current time.

EXEC CICS START TIME (330000) issued at 1000 hours on Monday will expire at 0900 hours on Tuesday.

Since each end of an intersystem link may be in a different time zone, it is recommended that the INTERVAL form of expiration time be used when the transaction to be started is in a remote system.

Request Identifiers

As a means of symbolically identifying the request and any data associated with it, a unique request identifier is assigned by CICS/VS to each DELAY, POST, or 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 POST and 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 by a CANCEL command.

REQUEST CURRENT TIME OF DAY (ASKTIME)

```
ASKTIME
```

This command is used to update the CICS/VS time-of-day clock, and the fields

EIBDATE and EIBTIME in the EIB. The two fields contain initially the date and time when the task started. Refer to "Appendix A. EXEC Interface Block" on page 239 for details of the EIB.

DELAY PROCESSING OF A TASK (DELAY)

```
DELAY [INTERVAL(hhmmss)|INTERVAL(0)|
  TIME(hhmmss)]
[REQID(name)]

Conditions:  EXPIRED, INVREQ
```

This command is used to request CICS/VS to suspend the processing of the issuing task for a specified interval of time or until a specified time of day. It supersedes any previously initiated POST command for the task. (The POST command is described in the following section.)

The following example shows how to suspend the processing of a task for a specified period of time:

```
EXEC CICS DELAY
     INTERVAL(500)
     REQID('GXLBZQMR')
```

The following example shows how to suspend the processing of a task until a specified time of day:

```
EXEC CICS DELAY
     TIME(124500)
     REQID('UNIQCODE')
```

REQUEST NOTIFICATION WHEN SPECIFIED TIME HAS EXPIRED (POST)

```
POST [INTERVAL(hhmmss)|INTERVAL(0)|
 TIME(hhmmss)]
SET(ptr-ref)
[REQID(name)]

Conditions:  INVREQ, EXPIRED
```

This command is used to request notification that a specified time has expired. In response to this command, CICS/VS makes a timer event control area available for testing. This four-byte area is initialized to binary zeros, and the pointer reference specified in the SET option is set to its address. This area is available for the duration of the task issuing the POST command.

When the time specified has expired, the timer event control area is posted; that is, its first byte is set to X'40' and its third byte to X'80'. Posting can be tested in either of the following ways:

- By checking the timer event control area at intervals. CICS/VS must be given the opportunity to post the area; that is, the task must relinquish control of CICS/VS before testing the area. Normally, this condition is satisfied as a result of other commands being issued; if a task is performing a long internal function, control can be relinquished by issuing a SUSPEND command, described in "Chapter 4.3. Task Control" on page 187.
- By suspending task activity by a WAIT EVENT command until the timer event control area is posted. This action is similar to issuing a DELAY command, the difference being that with a POST, WAIT EVENT sequence, it is possible to perform some processing after issuing the POST command, whereas a DELAY command suspends task activity at once. No other task should attempt to wait on the event set up by a POST command. The timer event control area can be released for a variety of reasons (see below). If this happens, the result of any other task issuing a WAIT on the event set up by the POST is unpredictable.

  However, other tasks can CANCEL the event if they have access to the REQID associated with the POST command. (See CANCEL command and description of REQID option.)

A timer event control area provided for a task is not released or altered (except as described above) until one of the following events occurs:

- The task issues a subsequent DELAY, POST, or START command.
- The task issues a CANCEL command to cancel the POST command.
- The task is terminated, normally or abnormally.
- Any other task issues a CANCEL command for the event set up by the POST command.

A task can have only one POST command active at any given time. Any DELAY, POST, or START command supersedes a previously issued POST command by the task.

The following example shows how to request a timer event control area for a task, to be posted after 30 seconds:

```
EXEC CICS POST
     INTERVAL(30)
     REQID('RBL3D')
     SET(PREF)
```

The following example shows how to provide a timer event control area for the task, to be posted when the specified time of day is reached. Since no request identifier is specified in the command, CICS/VS automatically assigns one and returns it to the application program in the EIBREQID field in the EIB.

```
EXEC CICS POST
     TIME(PACKTIME)
     SET(PREF)
```

WAIT FOR AN EVENT TO OCCUR (WAIT EVENT)

```
WAIT EVENT ECADDR(ptr-value)
```

This command is used to synchronize a task with the completion of an event initiated by the same task or by another task. The event would normally be the posting, at the expiration time, of a timer event control area provided in response to a POST command, as described in the preceding section. The WAIT EVENT command provides a method of directly relinquishing control to some other task until the event being waited on is completed.

A pointer value giving the address of an event control area must be specified in the ECADDR option. The event control area must conform to the format and standard posting conventions for ECBs; it will normally be the timer event control area created by a POST command.

The following example shows how to suspend processing of a task until the specified event control area is posted:

```
EXEC CICS WAIT EVENT
     ECADDR(PVALUE)
```

START A TASK (START)

```
START [INTERVAL(hhmmss)|INTERVAL(0)|
 TIME(hhmmss)]
TRANSID(name)
[REQID(name)]
[FROM(data-area)
  LENGTH(data-value)[FMH]]
[TERMID(name)]
[SYSID(name)]
[RTRANSID(name)]
[RTERMID(name)]
[QUEUE(name)]
[NOCHECK]
[PROTECT]

Conditions:  IOERR, INVREQ,
ISCINVREQ, SYSIDERR, TERMIDERR,
TRANSIDERR
```

This command is used to start a task, on a local or remote system, at a specified time. The starting task may pass data to the started task and may also specify a terminal to be used by the started task as its principal facility. The TRANSID, TERMID, and FROM options specify the transaction to be executed, the terminal to be used, and the data to be used, respectively.

The FMH option may be specified if the FROM option is specified. It indicates that the data, to be passed to the started task, contains function management headers.

Further data may be passed to the started task in the RTRANSID, RTERMID, and QUEUE options. For example, one task can start a second task passing it a transaction name and a terminal name to be used when the second task starts a third task; the first task may also pass the name of a queue to be accessed by the second task.

If data is to be passed, it will be queued using the request identifier specified in the REQID option, if one is provided. This identifier should be recoverable (in temporary-storage terms) if the PROTECT option is also specified, or nonrecoverable if PROTECT is not specified, otherwise unpredictable results can occur. Such problems cannot occur if REQID is not used .

The NOCHECK option specifies that no response (to execution of the START command) is expected by the starting transaction. For START commands naming tasks to be started on a local system, error conditions will be returned, whereas those for tasks to be started on a remote system will not be returned. The NOCHECK option allows CICS/VS to improve performance when the START command has to be shipped to a remote system; it is also a prerequisite if the shipping of the START command is queued pending the establishing of links to the remote system.

One or more constraints have to be satisfied before the transaction to be executed can be started, as follows:

1. The specified interval must have elapsed or the specified expiration time must have been reached. See the section "Expiration Times" earlier in the chapter. It is recommended that the INTERVAL option be specified when a transaction is to be executed on a remote system; this avoids complications arising when the local and remote systems are in different time zones.

2. If the TERMID option is specified, the named terminal must be available.

3. If the PROTECT option is specified, the starting task must have taken a successful syncpoint. This option, coupled to extensions to system tables, reduces the exposure to lost or duplicated data caused by failure of a starting task.

4. If the transaction to be executed is on a remote system the format of the data must be declared to be the same as that at the local system. This is done by the DATASTR and RECFM operands of the DFHTCT TYPE=SYSTEM system macro. For CICS/VS-CICS/VS these are always the default values. For CICS/VS-IMS/VS care should be taken to specify the correct values.

Execution of a START command naming a transaction in the local system will supersede any outstanding POST command executed by the starting task.

STARTING TASKS WITHOUT TERMINALS

If the task to be started is not associated with a terminal, each START command results in a separate task being started. This happens regardless of whether or not data is passed to the started task.

The following example shows how to start a specified task, not associated with a terminal, in one hour:

```
EXEC CICS START
     TRANSID('TRNL')
     INTERVAL(10000)
     REQID('NONGL')
```

STARTING TASKS WITH TERMINALS BUT WITHOUT DATA

Only one task is started if several START commands, each specifying the same transaction and terminal, expire at the same time or prior to terminal availability.

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

```
EXEC CICS START
     TRANSID('TRN1')
     TIME(185000)
     TERMID('STA5')
```

STARTING TASKS WITH TERMINALS AND DATA

Data is passed to a started task if one or more of the FROM, RTRANSID, RTERMID, and QUEUE options is specified. Such data is accessed by the started task through execution of a RETRIEVE command as described later in the chapter.

It is possible to pass many data records to a new task by issuing several START commands, each specifying the same transaction and terminal.

Execution of the first START command will ultimately cause the new task to be started and allow it to retrieve the data specified on the command. The new task will also be able to retrieve data specified on subsequently executed START commands that expire before the new task is terminated. If such data has not been retrieved before the new task is terminated, another new task will be started and will be able to retrieve the outstanding data.

The following example shows how to start a task associated with a terminal and pass data to it:

```
EXEC CICS START
     TRANSID('TRN2')
     TIME(173000)
     TERMID('STA3')
     REQID(DATAREC)
     FROM(DATAFLD)
     LENGTH(100)
```

RETRIEVE DATA STORED FOR A TASK (RETRIEVE)

```
RETRIEVE {INTO(data-area)|
 SET(ptr-ref)}
LENGTH(data-area)
[RTRANSID(data-area)]
[RTERMID(data-area)]
[QUEUE(data-area)]
[WAIT]

Conditions:  ENDDATA, ENVDEFERR,
INVREQ, INVTSREQ, IOERR, LENGERR,
NOTFND,
```

This command is used to retrieve data stored by expired START commands (the START command is described in the previous section). It is the only method available for accessing such data.

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 task 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 task is released upon execution of the RETRIEVE command, or upon termination of the task if no RETRIEVE command is executed prior to termination.

A task 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 started the task; it does so by issuing consecutive RETRIEVE commands. Expired data records are presented to the task upon request in expiration time sequence, starting with any data stored by the command that started the task, and including data from any commands that have expired since the task started. Each data record is retrieved from temporary storage using the REQID of the original START command as the identification of the record in temporary storage.

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 WAIT option specifies that, if all expired data records have already been retrieved, the task is suspended until further expired data records become available. The ENDDATA exceptional condition will be raised only if CICS/VS is shut down before any expired data records become available.

If a value has been specified in the DTIMOUT operand of the DFHPCT TYPE=ENTRY system macro, the ENDDATA condition will be raised if no data is available after the specified length of time. This condition will be raised also if the terminal, on which the transaction has been suspended, receives a request for a transaction other than the one that has been suspended. This condition will be raised also if CICS/VS enters shut down and the transaction is still suspended. An attempt to reissue the RETRIEVE command with the WAIT option after this event (that is, system shut down) will cause an abend with a code of AICB.

If the retrieved data contains FMHs, as specified by the FMH option on the associated START command, field EIBFMH in the EIB will be set to X'FF'. If no FMH is present, EIBFMH will be set to X'00'.

If an input/output error occurs during a retrieval operation, the IOERR exceptional condition occurs. The operation can be retried by reissuing the RETRIEVE command.

The following example shows how to retrieve data stored by a START command for the task, and store it in a specified area:

```
EXEC CICS RETRIEVE
     INTO(DATAFLD)
     LENGTH(LENG)
```

The following example shows how to request retrieval of a data record stored for a task 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 RETRIEVE
     SET(PREF)
     LENGTH(LENG)
```

CANCEL INTERVAL CONTROL REQUESTS (CANCEL)

```
CANCEL [REQID(name)[TRANSID(name)]
[SYSID(name)]]

Conditions:  INVREQ, ISCINVREQ,
NOTFND, SYSIDERR
```

This command is used to cancel a previously issued DELAY, POST, or START command. The presence of SYSID will cause the command to be shipped to a remote system. If SYSID is not present, TRANSID (if present) will determine where the command is to be executed. The effect of the cancellation varies depending on the type of command being canceled, as follows:

- A DELAY command can be canceled only prior to its expiration, and only by a task other than the task that issued the DELAY command (which is suspended for the duration of the request). The REQID used by the suspended task must be specified. The effect of the cancellation is the same as an early expiration of the original DELAY. That is, the suspended task becomes dispatchable as though the original expiration time had been reached.

- When a POST command issued by the same task is to be canceled, no REQID should be specified; if it is, it will be ignored. Cancellation can be requested either before or after expiration of the original request. The effect of the cancellation is as if the original request had never been made.

- When a POST command issued by another task is to be canceled, the REQID of that command must be specified. The effect of the cancellation is the same as an early expiration of the original POST request. That is, the timer event control area for the other task is posted as though the original expiration time had been reached.

- When a START command is to be canceled, the REQID of the original command must be specified. The effect of the cancellation is as if the original command had never been made. The cancellation is effective only prior to expiration of the original command.

INTERVAL CONTROL OPTIONS

ECADDR(ptr-value)
specifies the address of the timer event control area that must be posted before task activity can be resumed.

FMH
specifies that the user data to be passed to the started task contains FMHs.

FROM(data-area)
specifies the data that is to be stored for a task that is to be started at some future time.

INTERVAL(hhmmss)
specifies the expiration time for an interval control function as an interval of time that is to elapse from the time at which the interval control command is issued. The time specified is added to the current clock time by CICS/VS when the command is executed to calculate the expiration time.

This option is used in DELAY commands (to specify the time for which the task should be suspended), POST commands (to specify when the posting of the timer event control area should occur), and START commands (to specify when a new task should be started).

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 must be a data value that is the length of the data that is to be stored for the new task.

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.

NOCHECK
specifies that, for a remote system, CICS/VS should optimize the execution of the START command to improve performance by providing less error checking and slightly less function.

PROTECT
specifies that, in addition to the constraints described earlier in the chapter, the new task will not be started until the starting task has taken a sync point. If the starting task abends before the sync point is taken, the request to start the new task will be canceled. If the REQID option is specified as well, the request identifier should be a name defined as recoverable to temporary storage.

QUEUE{(name)|(data area)}
when used in a START command, "name" specifies the name of the queue that may be used by the transaction specified also in the START command. The name must be up to eight characters in length.

When used in a RETRIEVE command, "data area" specifies the name of the queue that may be accessed by

the transaction issuing the RETRIEVE command. The data area must be eight characters in length.

REQID(name)
specifies a unique name (up to eight characters) to identify a command.

This option can be used in a DELAY, POST, or START command when another task is to be provided with the capability of canceling an unexpired command; and in CANCEL commands, except those canceling a POST command issued by the same task (for which, the REQID option is ignored if it is specified).

If this option is omitted from a POST command, CICS/VS generates a unique request identifier in the EIBREQID field of the EXEC interface block. This applies also to a START command unless the NOCHECK option is specified, in which case field EIBREQID is set to blanks and cannot be used subsequently to cancel the START command.

RTERMID{(name)|(data area)}
When used in a START command, "name" specifies a value, for example a terminal name, that may be retrieved when the transaction, specified in the TRANSID option in the START command, is started. The name must be up to four characters in length.

When used in a RETRIEVE command, "data area" specifies an area which may be used in the TERMID option of a START command that may be executed subsequently. The data area must be four characters in length.

RTRANSID{(name)|(data area)}
When used in a START command, "name" specifies a value, for example a transaction name, that may be retrieved when the transaction, specified in the TRANSID option in the START command, is started. The name must be up to four characters in length.

When used in a RETRIEVE command, "data area" specifies an area which may be used in the TRANSID option of a START command that may be executed subsequently. The data area must be four characters in length.

SET(ptr-ref)
When used with a POST command, SET specifies the pointer reference to be set to the address of the 4-byte timer event control area generated by CICS/VS. This area is initialized to binary zeros; on expiration of the specified time, the first byte is set to X'40', and the third byte to X'80'.

When used with a RETRIEVE command, SET specifies the pointer reference to be set to the address of the retrieved data. If this option is specified, locate-mode access is implied.

SYSID(name) **remote systems only**
specifies the name of the system whose resources are to be used for intercommunication facilities. The name may be up to four characters in length.

TIME(hhmmss)
specifies the expiration time for an interval control function. See the section "Expiration Times" earlier in the chapter.

This option is used in DELAY commands (to specify the time for which the task should be suspended), POST commands (to specify when the posting of the timer event control area should occur), and START commands (to specify when a new task 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.

TERMID(name)
specifies the symbolic identifier of the terminal associated with a transaction to be started as a result of a START command. This option is required when the transaction to be started 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.

If the transaction to be started is on a remote system, the terminal identifier will be assumed to be defined in the TCT on the remote system.

TRANSID(name)
specifies the symbolic identifier of the transaction to be executed by a task started as the result of a START command, or to be canceled by a CANCEL command. The name may be up to four characters in length and must have been defined in the program control table (PCT) by the system programmer.

If SYSID is specified, the transaction is assumed to be on a remote system irrespective of whether or not the name is defined in the PCT. Otherwise the entry in the PCT will be used to determine if

the transaction is on a local or remote system.

WAIT
specifies that, if all expired data records have already been retrieved, the task is to be put into a wait state until further expired data records become available. The ENDDATA condition will be raised only if CICS/VS is shut down before any expired data records become available.

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.

ENVDEFERR
occurs when a RETRIEVE command specifies an option not specified by the corresponding START command.

Default action: terminate the task abnormally.

EXPIRED
occurs if the time specified in a POST or DELAY command has already expired when the command is issued.

Default action: ignore the condition.

INVREQ
occurs if an invalid type of interval control command is received for processing by CICS/VS.

Default action: terminate the task abnormally.

INVTSREQ
occurs if there is no support for a temporary storage read request issued by CICS/VS during execution of a RETRIEVE command. This situation can occur when a dummy Temporary Storage Program is included in the system by the system programmer in place of a functional Temporary Storage Program.

Default action: terminate the task abnormally.

IOERR
occurs if an input/output error occurs during a RETRIEVE or START operation. The operation can be retried by reissuing the RETRIEVE command.

Default action: terminate the task abnormally.

ISCINVREQ
occurs when the remote system indicates a failure which does not correspond to a known condition.

Default action: terminate the task abnormally.

LENGERR
occurs in move-mode retrieval 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 interval control command.
- The RETRIEVE command is issued by a task that is started by a START command which did not specify the FROM option.
- The request identifier associated with a START command fails to remain unique; when a RETRIEVE command is issued, CICS/VS cannot find the data.

Default action: terminate the task abnormally.

SYSIDERR
occurs when the SYSID option specifies either a name which is not defined in the intersystem table or a system to which the link is closed.

Default action: terminate the task abnormally.

TERMIDERR
occurs if the terminal identifier in a START command cannot be found in the terminal control table.

Default action: terminate the task abnormally.

TRANSIDERR
occurs if the transaction identifier specified in a START command cannot be found in the program control table.

Default action: terminate the task abnormally.

Chapter 4.3. Task Control

The CICS/VS task control program provides functions that synchronize task activity, or that control the use of resources.

CICS/VS processes tasks concurrently according to priorities assigned by the system programmer. Control of the processor is given to the highest priority task that is ready to be processed and is returned to the operating system when no further work can be done by CICS/VS or by user-written application programs.

Task control commands are provided to:

- Suspend a task (SUSPEND).
- Schedule the use of a resource by a task (ENQ and DEQ).

A task can issue the SUSPEND command to relinquish control and allow tasks with a higher priority to proceed. This facility can be used to prevent processor-intensive tasks from monopolizing the processor. If no higher-priority task is waiting to be processed, control is returned to the issuing task; that is, the task remains dispatchable.

Scheduling the use of a resource by a task is sometimes useful in order to protect the resource from concurrent use by more than one task, that is, to make the resource serially reusable. Each task that is to use the resource issues an ENQ (enqueue) command. The first task to do so has the use of the resource immediately, but subsequent ENQ commands for the resource, issued by other tasks, result in those tasks being suspended until the resource is available. Each task using the resource should issue a DEQ (dequeue) command when it has finished with it. The resource then becomes available and the next task to have issued an ENQ command is resumed and given use of the resource. The other tasks obtain the resource in turn, in the order in which they enqueued upon it.

Exceptional conditions that occur during execution of a task control command are handled as described in "Chapter 1.5. Exceptional Conditions" on page 25.

SUSPEND A TASK (SUSPEND)

```
SUSPEND
```

This command is used to relinquish control to a task of higher dispatching priority. Control is returned to the task issuing the command if no other task of a higher priority is ready to be processed.

SCHEDULE USE OF A RESOURCE BY A TASK (ENQ AND DEQ)

```
{ENQ|DEQ} RESOURCE(data-area)
[LENGTH(data-value)]

Condition:  ENQBUSY (ENQ only)
```

The ENQ and DEQ commands can be used to enqueue upon and dequeue from a resource that is to be protected from concurrent use by more than one task.

The ENQ command causes further execution of the task issuing the ENQ command to be synchronized with the availability of the specified resource; control is returned to the task when the resource is available.

The ENQBUSY condition allows a conditional ENQ to be used. If a resource is not available when enqueued, the ENQBUSY condition is raised. The execution of a HANDLE CONDITION ENQBUSY command will return control to the task at the ENQBUSY label, without waiting for the resource to become available.

The DEQ command causes a resource currently enqueued upon by the task to be released for use by other tasks. If a task enqueues upon a resource but does not dequeue from it, CICS/VS automatically releases the resource when the task is terminated.

When issuing the ENQ command, the resource to be enqueued upon must be identified by one of the following methods:

- Specifying a data area that is the resource.
- Specifying a data variable that contains a unique character-string argument (for example, an employee name) that represents the resource. The character string may be up to 255 bytes in length. The length of the string must be supplied in the LENGTH option.

When issuing the DEQ command, the resource to be dequeued from must be identified by the method used when enqueuing upon the resource.

The following examples show how to enqueue upon a resource using the two methods shown above. Substituting "DEQ" for "ENQ" in these examples illustrates the ways in which a resource can be released.

```
EXEC CICS ENQ
     RESOURCE(RESNAME)

        or

EXEC CICS ENQ
     RESOURCE(SOCSECNO)
     LENGTH(9)
```

TASK CONTROL OPTIONS

LENGTH(data-value)
specifies that the resource to be enqueued upon (or dequeued from) is a data variable of length given by the data value. The data value is a halfword binary value in the range 1 through 255. If the LENGTH option is specified in an ENQ command, it must also be specified in the DEQ command for that resource, and the values of these options must be the same. This option is required if the resource is specified as a character string; it should not be specified otherwise.

RESOURCE(data-area)
specifies either the resource to be enqueued upon (or dequeued from) or a data variable that contains a character string (for example an employee name) that represents the resource. In the latter case, the length of the string must be specified by the LENGTH option.

TASK CONTROL EXCEPTIONAL CONDITIONS

ENQBUSY
occurs when an ENQ command specifies a resource that is unavailable.

Default action: wait for the resource to become available.

Chapter 4.4. Program Control

The CICS/VS program control program 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.

Program control commands are provided to:

- Link one user-written application program to another, anticipating subsequent return to the requesting program (LINK). The COMMAREA option allows data to be passed to the requested application program.
- Transfer control from one user-written application program to another, with no return to the requesting program (XCTL). The COMMAREA option allows data to be passed to the requested application program.
- Return control from one user-written application program to another or to CICS/VS (RETURN). The COMMAREA option allows data to be passed to a newly-initiated transaction.
- Load a designated application program, table, or map into main storage and return control to the requesting program (LOAD).
- Delete a previously loaded application program, table, or map from main storage (RELEASE).

Exceptional conditions that occur during execution of a program control command are handled as described in "Chapter 1.5. Exceptional Conditions" on page 25.

The HANDLE ABEND command can be used to deal with abnormal terminations. Refer to "Chapter 5.2. Abnormal Termination Recovery" on page 215 for further information about this command.

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.

LINK TO ANOTHER PROGRAM ANTICIPATING RETURN (LINK)

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

Condition:  PGMIDERR
```

This 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
     PROGRAM('PROG1')
```

The COMMAREA option can be used to pass data to the linked-to program. For further details, see the section "Passing Data to Other Programs" later in the chapter. 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 with regard to handling exceptional conditions, attention identifiers, and abends. For example, the effects of HANDLE commands in the linking program are not inherited by the linked-to program, but the original HANDLE commands are restored on return to the linking program. Figure 20 on page 190 illustrates the concept of logical levels.

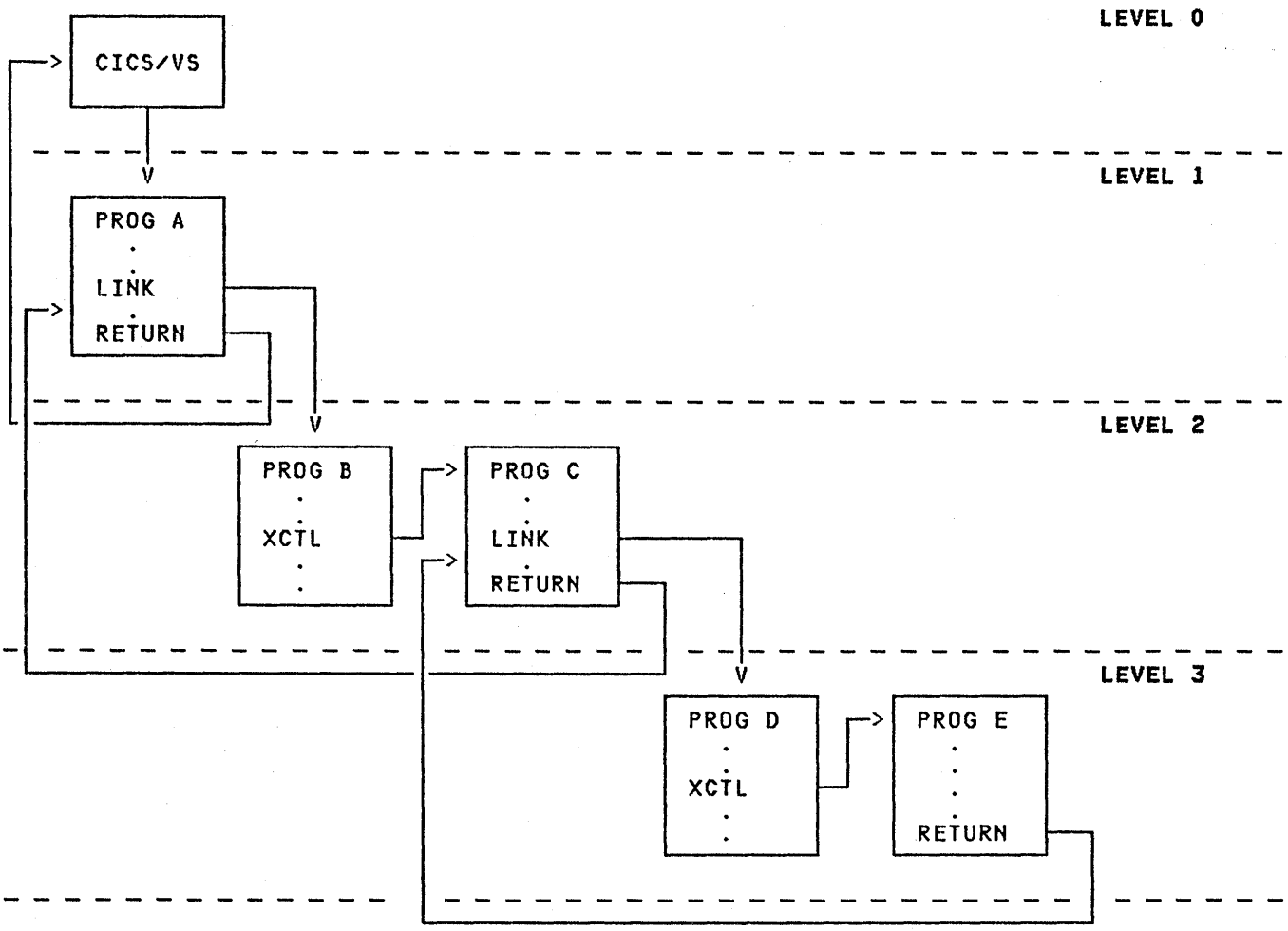


Figure 20. Application Program Logical Levels

TRANSFER PROGRAM CONTROL (XCTL)

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

Condition:  PGMIDERR
```

This 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 not already in main storage, it will be loaded.

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

```
EXEC CICS XCTL
     PROGRAM('PROG2')
```

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

RETURN PROGRAM CONTROL (RETURN)

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

Condition:  INVREQ
```

This 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. In addition, the COMMAREA option can be used to pass data to the new task that will be started. For further details, see the section "Passing Data to Other Programs" later in the chapter. 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; the INVREQ exceptional condition will occur otherwise.

LOAD A PROGRAM (LOAD)

```
LOAD PROGRAM(name)
[SET(ptr-ref)]
[LENGTH(data-area)]
[ENTRY(ptr-ref)]
[HOLD]

Condition:  PGMIDERR
```

This command is used to fetch application programs, tables, or maps from the library where they reside and load them into main storage. This facility is used to load an application program that will be used repeatedly, thereby reducing system overhead through a single load, to load a table to which control is not to be passed, or to load a map to be used in a mapping operation. (See "Chapter 3.3. Basic Mapping Support (BMS)" on page 125 for further details about maps.) CICS/VS sets the pointer reference specified in the SET option to the address of the loaded program, table, or map; if the LENGTH option is specified, the data area provided will be set to the length involved. (See also the RELOAD operand of the DFHPPT TYPE=ENTRY macro as described in the CICS/VS System Programmer's Reference Manual.)

If the HOLD option is specified, the loaded program, table, or map remains in main storage until a RELEASE command is issued; if HOLD is not specified, the program, table, or map remains in main storage until a RELEASE command is issued or until the task that issued the LOAD command is terminated normally or abnormally.

The following example shows how to load a user-prepared table called TAB1:

```
EXEC CICS LOAD
     PROGRAM('TAB1')
     SET(PTR)
```

DELETE A LOADED PROGRAM (RELEASE)

```
RELEASE PROGRAM(name)

Condition:  PGMIDERR
```

This command is used to delete from main storage a program, table, or map previously loaded in response to a LOAD command. If the HOLD option is specified in the LOAD command, the loaded program is deleted only in response to a RELEASE command. If the HOLD option is not specified, the loaded program can be deleted by a RELEASE, or it will be deleted automatically when the task that issued the LOAD is terminated.

The following example shows how to delete an application program, called PROG4, loaded in response to a LOAD command:

```
EXEC CICS RELEASE
     PROGRAM('PROG4')
```

PASSING DATA TO OTHER PROGRAMS

This section describes how data can be passed between programs when control is passed to another program by means of a program control command. (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 "Chapter 1.6. Access to System Information" on page 29 for details. Another way is to store the data in temporary storage; see "Chapter 4.7. Temporary Storage Control" on page 207 for details.)

The COMMAREA option of the LINK and XCTL commands specifies the name of a data area (known as a **communication area**) in which data can be passed to the program being invoked.

In a similar manner, the COMMAREA option of the RETURN command specifies the name of a communication area in which data can be passed to the transaction identified in the TRANSID option. (The TRANSID option specifies a transaction that will be initiated when input is received from the terminal associated with the task.)

The length of the communication area is specified in the LENGTH option; PL/I programs need not specify the length.

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.

In an assembler-language program, the data area should be a DSECT. The register used to address this DSECT must be loaded from DFHEICAP, which is in the DFHEISTG DSECT.

In a COBOL program, the data area must be called DFHCOMMAREA.

In a PL/I program, the data area can have any name, but it must be declared as a based variable, based on the parameter passed to the program. The pointer to this based variable should be declared explicitly as a pointer rather than contextually by its appearance in the declaration for the area. This will prevent the generation of a PL/I error message.

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. It must not be longer than the length being passed, because the results in this situation are unpredictable.

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 an XCTL command, a copy of that area is made unless the area to be passed has the same address and length as the area that was passed to the program issuing the command. For example, if program A issues a LINK command to program B which, in turn, issues an XCTL command to program C, and if B passes to C the same communication area that A passed to B, program C will be passed addressability to the communication area that belongs to A (not a copy of it) and any changes made by C will be available to A when control returns to it.

A communication area can be passed by means of a RETURN command issued at the highest logical level when control returns to CICS/VS; in this case, a copy of the communication area is made, and addressability to the copy is passed to the first program of the next transaction.

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

The following examples show how a LINK command causes data to be passed to the program being linked to; the XCTL command is coded in a similar way. Further examples show how the RETURN command is used to pass data to a new transaction.

LINK or XCTL, for assembler language

```
DFHEISTG DSECT                                          (Invoking program)
COMREG   DS 0CL20
FIELD    DS CL3
         .
         .
PROG1    CSECT
         .
         .
         MVC FIELD,=C'ABC'
         EXEC CICS LINK PROGRAM('PROG2') COMMAREA(COMREG)
         .
         .
         END
COMREG   DSECT                                          (Invoked program)
FIELD    DS CL3
         .
         .
PROG2    CSECT
         .
         .
         L COMPTR,DFHEICAP   ADDRESS COMMAREA
         USING COMREG,COMPTR
         CLC FIELD,=C'ABC'
         .
         .
         END
```

LINK or XCTL, for COBOL

```
IDENTIFICATION DIVISION.                                (Invoking program)
PROGRAM ID. 'PROG1'.
         .
         .
WORKING-STORAGE SECTION.
01  COMMUNICATIONS-REGION.
    02  FIELD PICTURE X(3).
         .
         .
PROCEDURE DIVISION.
    MOVE 'ABC' TO FIELD.
    EXEC CICS LINK PROGRAM('PROG2')
         COMMAREA(COMMUNICATIONS-REGION) LENGTH(3) END-EXEC.
         .
         .

IDENTIFICATION DIVISION.                                (Invoked program)
PROGRAM-ID. 'PROG2'.
         .
         .
LINKAGE SECTION.
01  DFHCOMMAREA.
    02  FIELD PICTURE X(3).
         .
         .
PROCEDURE DIVISION.
    IF EIBCALEN GREATER ZERO THEN IF FIELD EQUALS 'ABC' ....
```

LINK or XCTL, for PL/I

```
PROG1: PROC OPTIONS(MAIN);                              (Invoking program)
DCL 1 COMMUNICATIONS_REGION AUTOMATIC,
      2 FIELD CHAR(3),
          .
          .
FIELD='ABC';
EXEC CICS LINK PROGRAM('PROG2')
     COMMAREA(COMMUNICATIONS_REGION) LENGTH(3);
END;

PROG2: PROC(COMM_REG_PTR) OPTIONS(MAIN);                (Invoked program)
DCL COMM_REG_PTR PTR;
DCL 1 COMMUNICATIONS_REGION BASED(COMM_REG_PTR);
      2 FIELD CHAR(3),
          .
          .
IF EIBCALEN>0 THEN DO;
    IF FIELD='ABC' THEN ...
          .
          .
    END;
          .
          .
END;
```

RETURN, for assembler language

```
DFHEISTG DSECT                                          (Invoking program)
TERMSTG  DS 0CL20
FIELD    DS CL3
DATAFLD  DS CL17
         .
         .
PROG1    CSECT
         .
         .
         MVC FIELD,=C'ABC'
         EXEC CICS RETURN TRANSID('TRN2') COMMAREA(TERMSTG)
         .
         .
         END

TERMSTG  DSECT                                          (Invoked program)
FIELD    DS CL3
DATAFLD  DS CL17
         .
         .
PROG2    CSECT
         .
         .
         CLC EIBCALEN,=H'0'
         BNH LABEL2
         L COMPTR,DFHEICAP
         USING TERMSTG,COMPTR
         CLC FIELD,=C'XYZ'
         BNE LABEL1
         MVC FIELD,=C'ABC'
LABEL1   DS 0H
         .
         .
LABEL2   DS 0H
         .
         .
         END
```

RETURN, for COBOL

```
IDENTIFICATION DIVISION.                              (Invoking program)
PROGRAM-ID. 'PROG1'.
          .
          .
WORKING-STORAGE SECTION.
01  TERMINAL-STORAGE.
    02  FIELD PICTURE X(3).
    02  DATAFLD PICTURE X(17).
          .
          .
PROCEDURE DIVISION.
    MOVE 'ABC' TO FIELD.
    EXEC CICS RETURN TRANSID('TRN2')
         COMMAREA(TERMINAL-STORAGE) LENGTH(20) END-EXEC.
          .
          .
IDENTIFICATION DIVISION.                              (Invoked program)
PROGRAM-ID.  'PROG2'
          .
          .
          .
LINKAGE SECTION.
01  DFHCOMMAREA.
    02  FIELD PICTURE X(3).
    02  DATAFLD PICTURE X(17).
          .
          .
PROCEDURE DIVISION.
    IF EIBCALEN GREATER ZERO THEN
    IF FIELD EQUALS 'XYZ' MOVE 'ABC' TO FIELD.
    EXEC CICS RETURN END-EXEC.
```

RETURN, for PL/I

```
PROG1: PROC OPTIONS(MAIN);                            (Invoking Program)
DCL 1 TERMINAL_STORAGE,
      2 FIELD CHAR(3),
          .
          .
FIELD='XYZ';
EXEC CICS RETURN TRNID('TRN2')
     COMMAREA(TERMINAL_STORAGE);
END;

PROG2: PROC(TERM_STG_PTR) OPTIONS(MAIN);              (Invoking Program)
DCL   TERM_STG_PTR PTR;
DCL 1 TERMINAL_STORAGE BASED (TERM_STG_PTR),
      2 FIELD CHAR(3),
          .
          .
IF EIBCALEN>0 THEN DO;
    IF FIELD='XYZ' THEN FIELD='ABC';
    END;
EXEC CICS RETURN;
END;
```

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 commands, a pointer to the data area is passed or a copy of it (see "Passing Data to Other Programs" earlier in this chapter); and for 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.

ENTRY(ptr-ref)
specifies the pointer reference that is to be set to the address of the entry point in the program, table, or map that has been loaded.

HOLD
specifies that the loaded program, table, or map is not to be deleted (if still resident) when the task issuing the LOAD command is terminated; deletion is to occur only in response to a RELEASE command, from this task or from another task.

LENGTH(parameter)
specifies a halfword binary value to be used with LINK, XCTL, RETURN, and LOAD commands.

For a LINK, XCTL, or RETURN command, the parameter must be a data value that is the length in bytes of the communication area. If a negative value is supplied, zero is assumed.

For a LOAD command, the parameter must be a data area. On completion of the LOAD operation, the data area is set to the length of the loaded program, table, or map.

PROGRAM(name)
specifies the identifier of the program to which control is to be passed unconditionally (for a LINK or XCTL command); or the identifier of a program, table, or map to be loaded (for a LOAD command) or deleted (for a RELEASE command). The specified name must consist of up to eight alphameric characters and must have been defined in the processing program table (PPT).

SET(ptr-ref)
specifies the pointer reference that is to be set to the address at which a program, table, or map is loaded.

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

INVREQ
occurs if either of the following situations exists:

- A RETURN command with the COMMAREA option is issued in a program that is not at the highest logical level.
- A RETURN command with the TRANSID option is issued in a task that is not associated with a terminal.

PGMIDERR
occurs if a program, table, or map cannot be found in the PPT or is disabled.

Default action: terminate the task abnormally.

Chapter 4.5. Storage Control

The CICS/VS storage control program controls requests for main storage to provide intermediate work areas and any other main storage not provided automatically by CICS/VS but needed to process a transaction. The acquired main storage can be initialized to any bit configuration; for example, binary zeros or EBCDIC blanks.

Storage control commands are provided to:

- Obtain and initialize main storage (GETMAIN).
- Release main storage (FREEMAIN).

CICS/VS releases all main storage associated with a task when the task is terminated normally or abnormally. This includes any storage acquired, and not subsequently released, by the application program.

If there is insufficient main storage to satisfy a GETMAIN command, the NOSTG exceptional condition occurs and all activity within the task is suspended until sufficient storage becomes available, when task activity will be resumed and the requested storage obtained.

Exceptional conditions that occur during execution of a storage control command are handled as described in "Chapter 1.5. Exceptional Conditions" on page 25.

OBTAIN AND INITIALIZE MAIN STORAGE (GETMAIN)

```
GETMAIN SET(ptr-ref)
LENGTH(data-value)
[INITIMG(data-value)]

Condition:  NOSTG
```

This command is used to obtain a specified amount of main storage and, optionally, to initialize that storage to a specified bit configuration. The pointer reference specified in the SET option is set to the address of the acquired storage. The acquired storage is doubleword-aligned.

Storage should be released when no longer needed; it will then be available to other tasks. Other storage not released will be released by CICS/VS when the task is terminated.

The following example shows how to obtain a 1024-byte area of main storage:

```
EXEC CICS GETMAIN
     SET(PTR)
     LENGTH(1024)
     INITIMG(BLANK)
```

RELEASE MAIN STORAGE (FREEMAIN)

```
FREEMAIN DATA(data-area)
```

This command is used to release main storage previously acquired by a GETMAIN command. If the task itself does not release the acquired storage, it is released by CICS/VS when the task is terminated.

The following example shows how to release main storage:

```
EXEC CICS FREEMAIN
     DATA(RECORD)
```

STORAGE CONTROL OPTIONS

DATA(data-area)
specifies that the main storage associated with the data area is to be released. This storage must have been acquired previously by a GETMAIN command and the length of data released will be the length obtained by the GETMAIN and not necessarily the length of the data area.

INITIMG(data-value)
specifies the one-byte hexadecimal initialization value for the acquired main storage. A data area must be provided in COBOL programs.

LENGTH(data-value)
specifies the length of main storage required as a halfword binary value. The maximum length that can be specified is 32767 bytes.

SET(ptr-ref)
specifies the pointer reference to be set to the address of the acquired main storage. The pointer reference addresses the user data, and not the CICS/VS control information that precedes the acquired main storage.

STORAGE CONTROL EXCEPTIONAL CONDITIONS

NOSTG
occurs if the requested main storage cannot be obtained.

Default action: suspend task activity until the required main storage can be provided.

Chapter 4.6. Transient Data Control

The CICS/VS transient data control program provides a generalized queuing facility. Data can be queued (stored) for subsequent internal or external processing. Selected data, specified in the application program, 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 or region, and extrapartition if the data is directed to a destination that is external to the CICS/VS partition or region. The destinations must be defined in the destination control table (DCT) by the system programmer when the CICS/VS system is generated.

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).

If TD is omitted, the command is assumed to be for temporary storage (see "Chapter 4.7. Temporary Storage Control" on page 207).

Exceptional conditions that occur during execution of a transient data control command are handled as described in "Chapter 1.5. Exceptional Conditions" on page 25.

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 must consist of variable-length records. Intrapartition destinations can be associated with either a terminal or an output data set. Intrapartition data may ultimately be transmitted upon request to the destination terminal or retrieved sequentially from the output data set.

Typical uses of intrapartition data include message switching, broadcasting, data base access and routing of output to several terminals (for example, for order distribution), queuing of data (for example, for assignment of order numbers or priority by arrival), and data collection (for example, for batched input from 2780 Data Transmission Terminals). If generated within the system, the CICS/VS Asynchronous Transaction Processing (ATP) facility can be used to transfer data to or from an intrapartition destination. (Refer to the section "Asynchronous Transaction Processing" later in this chapter for further information.)

The storage associated with an intrapartition queue can be reused. The system programmer can specify, for each symbolic destination, whether or not storage tracks are to be reused as the data on them is read. If the storage is specified to be non-reusable, an intrapartition queue continues to grow, irrespective of whether the data has been read, until a DELETEQ TD command is issued when the whole of an intrapartition queue is deleted and the storage associated with it is released.

Extrapartition Destinations

Extrapartition destinations are queues (data sets) residing on any sequential device (DASD, tape, printer, and so on), which are accessible by programs outside (or within) the CICS/VS partition or region. In general, sequential extrapartition destinations are used for storing and retrieving data outside the CICS/VS partition. For example, one task may read data from a remote terminal, edit the data, and write the results to a data set for subsequent processing in another partition or region. Logging 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 an extrapartition destination must be defined in the DCT by the system programmer. (Refer to the CICS/VS System Programmer's Reference Manual for details.)

Indirect Destinations

Intrapartition and extrapartition destinations can be used as indirect destinations, which are symbolic references to other destinations. This facility provides some flexibility in program maintenance in that data can be routed to a destination known by a different symbolic name, without the necessity for recompiling existing programs that use the original name; only the destination control table (DCT) need be changed. When the DCT has been changed, the application programs can route data to the destination using the previous symbolic name; however, the previous name is now an indirect destination that refers to the new symbolic name. Since indirect destinations are established by means of destination control table entries, the application programmer need not usually be concerned with how this is done. Further information is available in the CICS/VS System Programmer's Reference Manual.

Automatic Task Initiation (ATI)

For intrapartition destinations, CICS/VS provides the option of automatic task initiation. A basis for automatic task initiation is established by the system programmer by specifying a non-zero trigger level for a particular intrapartition destination in the DCT. (See the discussion of the DFHDCT TYPE=INTRA macro instruction in the CICS/VS System Programmer's Reference Manual.) When the number of entries (created by WRITEQ TD commands issued by one or more programs) in the queue (destination) reaches the specified trigger level, a task specified in the definition of the destination 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 task initiation cycle begins. That is, a new task is scheduled for initiation when the specified trigger level is again reached, whether or not execution of the prior task has terminated.

If an automatically initiated task does not deplete the queue, access to the queue is not inhibited. The task may be normally or abnormally terminated before the queue is emptied (that is, before a QZERO exceptional condition occurs in response to a READQ TD command). If the destination is a terminal, the same task is reinitiated regardless of the trigger level. If the destination is a data set, the task is not reinitiated until the specified trigger level is reached. If the trigger level of a queue is zero, no task is automatically initiated. To ensure that termination of an automatically initiated task 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.

Asynchronous Transaction Processing (ATP)

Typically, a task to be run under CICS/VS is initiated from a terminal and processed at regular intervals until completion, according to system service patterns established for CICS/VS. This mode of operation is sometimes referred to as **synchronous** transaction processing, because the task has complete control of the terminal which initiated it.

Support for asynchronous transaction processing can also be generated into a CICS/VS system. This capability is designed primarily to permit a type of batch processing within CICS/VS. A task is initiated from a terminal as described above, but the specified transaction identification code causes a CICS/VS-provided asynchronous transaction processing program to read the data to an intrapartition data set. In effect, data collection from a device such as the 2780 Data Transmission Terminal is possible. When the data has been read, the device is freed for other activity. An application program processes the data, and, upon operator request, output is queued for subsequent transmission to a specified terminal. If the automatic task initiation feature is generated into CICS/VS, that application program can be initiated automatically when a specified trigger level is reached (that is, when a specified number of inputs have been entered in the intrapartition data set).

The asynchronous transaction processing (ATP) facility is designed specifically for handling input from batch terminals like the 2770 and 2780. Generally, ATP can also be used for other, interactive terminals like the 2741. However, ATP is not intended for, and will not support, input from the 2980, 3270, or 3735; ATP is not available for VTAM logical units. Application programs intended to execute under control of ATP must not contain Basic Mapping Support (BMS) commands requesting BMS terminal paging facilities.

Additional information concerning the creation of user exits for asynchronous transaction processing and the coding of the exit routines is given in the CICS/VS

System Programmer's Reference Manual. The initiation of ATP by means of terminal commands is described in the CICS/VS Operator's Guide.

WRITE DATA TO TRANSIENT DATA QUEUE (WRITEQ TD)

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

Conditions:  IOERR, ISCINVREQ,
LENGERR, NOSPACE, NOTOPEN, QIDERR,
SYSIDERR
```

This command is used to write transient data to a predefined symbolic destination. The destination (queue) 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 length is known and a data area of the correct size is available. If SYSID is specified, LENGTH must be specified as well.

For CICS/DOS/VS, the LENGTH option must be specified for a destination other than disk; length is not checked. If the LENGTH option is omitted, the LENGERR condition will occur.

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
     QUEUE('CSML')
     FROM(MESSAGE)
     LENGTH(LENG)
```

READ DATA FROM TRANSIENT DATA QUEUE (READQ TD)

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

Conditions:  IOERR, ISCINVREQ,
LENGERR, NOTOPEN,
(CICS/OS/VS only), QIDERR, QZERO,
SYSIDERR
```

This command is used to read transient data from a predefined symbolic source. The source (queue) is identified in the QUEUE option.

The INTO option specifies 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. If SYSID is specified, LENGTH must be specified as well.

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. The area is retained until another transient data command is executed. After the retrieval operation, the data area specified in the LENGTH option is set to the record length.

If automatic task initiation is being used (see earlier in the chapter under "Automatic Task Initiation (ATI)"), the HANDLE CONDITION QZERO command should be included to ensure that termination of an automatically initiated task only occurs when the queue is empty.

For CICS/DOS/VS, the LENGTH option must be specified for a destination other than disk, when the INTO option is specified. If the LENGTH option is omitted, the LENGERR condition will occur.

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

```
EXEC CICS READQ TD
     QUEUE('CSML')
     INTO(DATA)
     LENGTH(LENG)
```

The following example shows how to read a record from an extrapartition data set (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
     QUEUE(EX1)
     SET(PREF)
```

DELETE AN INTRAPARTITION TRANSIENT DATA QUEUE (DELETEQ TD)

```
DELETEQ TD QUEUE(name)
[SYSID(name)]

Conditions:  ISCINVREQ, QIDERR,
SYSIDERR
```

This command is used to delete all of the transient data associated with a particular intrapartition destination (queue). All storage associated with the destination is released (deallocated).

This command must be used to release the storage associated with a destination specified as non-reusable in the destination control table. Otherwise, the storage remains allocated to the destination; the data and the amount of storage associated with the destination continue to grow whenever a WRITEQ TD command refers 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 placed. 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 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 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 to be written to, read from, or deleted. The name must be alphameric, up to four characters in length, and must have been defined in the destination control table (DCT) by the system programmer.

When used with the READQ TD command, the name used should not be that of the system spool file otherwise unpredictable results or an abnormal termination will occur.

If SYSID is specified, the data set is assumed to be on a remote system irrespective of whether or not the name is defined in the DCT. Otherwise the entry in the DCT will be used to determine if the data set is on a local or remote system.

SET(ptr-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.

SYSID(name) **remote systems only**
specifies the name of the system whose resources are to be used for intercommunication facilities. The name may be up to four characters in length.

TRANSIENT DATA CONTROL EXCEPTIONAL CONDITIONS

IOERR
occurs when an input/output error occurs and the data record in error is skipped. The IOERR condition occurs so long as the queue can be read; a QZERO condition occurs when the queue cannot be read, in which case a restart may be attempted.

Default action: terminate the task abnormally.

ISCINVREQ
occurs when the remote system indicates a failure which does not correspond to a known condition.

Default action: terminate the task abnormally.

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.

- The LENGTH option is not coded for an input operation (without the SET option) from, or an output operation to, a destination other than disk, involving fixed-length records.

Default action: terminate the task abnormally.

NOSPACE
occurs if no more space exists on the intrapartition queue. If the NOSPACE condition occurs, no more data should be written to the queue because it may be lost.

Default action: terminate the task abnormally.

NOTOPEN
occurs if the destination is closed.

Default action: terminate the task abnormally.

QBUSY (CICS/OS/VS only)
occurs if a READQ TD command attempts to access a record in an intrapartition queue that is being written to or is being deleted by another task. This exceptional condition applies only to input; output requests are always queued until the intrapartition queue is no longer busy.

Default action: the task issuing the READQ TD command waits until the queue is no longer being used for output.

QIDERR
occurs if the symbolic destination to be used with a transient data control command cannot be found.

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.

SYSIDERR
occurs when the SYSID option specifies either a name which is not defined in the intersystem table, or a system to which the link is closed.

Default action: terminate the task abnormally.

Chapter 4.7. Temporary Storage Control

The CICS/VS temporary storage control program provides the application programmer with the ability to store data in temporary storage queues, either in main storage, or in auxiliary storage on a direct-access storage device. Data stored in a temporary storage queue is known as temporary data.

Temporary storage control commands are provided to:

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

If TS is omitted, the command is assumed to be for temporary storage, not for transient data which has similar commands.

Exceptional conditions that occur during execution of a temporary storage control command are handled as described in "Chapter 1.5. Exceptional Conditions" on page 25.

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 (logical records) within a queue are referred to by relative position numbers. To avoid conflicts caused by duplicate names, a naming convention should be established, for example, the operator identifier, terminal identifier, or transaction identifier could be used as a prefix or suffix to each programmer-supplied symbolic name.

Temporary storage queues remain intact until they are deleted by the originating task or by any other task; prior to deletion, they can be accessed any number of times. 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.

Temporary data can be stored either in main storage or in auxiliary storage. Generally, main storage should be used if the data is needed for short periods of time; auxiliary storage should be used if the data is to be kept for long periods of time. Data stored in auxiliary storage is retained after CICS/VS termination and can be recovered in a subsequent restart, but data in main storage cannot be recovered. Main storage might be used to pass data from task to task, or for unique storage that allows programs to meet the requirement of CICS/VS that they be quasi-reentrant (that is, serially reusable between entry and exit points of the program).

Typical Uses of Temporary Storage Control

A temporary storage queue having only one record can be treated as a single unit of data that can be accessed using its symbolic name. Using temporary storage control in this way provides a typical "scratch pad" facility. This type of storage should be accessed using the READQ TS command with the ITEM(1) option; failure to do so may cause the ITEMERR condition to be raised.

In general, temporary storage queues of more than one record should be used only when direct access or repeated access to records is necessary; transient data control provides facilities for efficient handling of sequential data sets.

Some uses of temporary storage queues follow:

- Terminal paging. A task could retrieve a large master record from a direct-access data set, format it into several screen images (using Basic Mapping Support), store the screen images temporarily in auxiliary storage, and then ask the terminal operator which "page" (screen image) is desired. The application programmer can provide a program (as a generalized routine or unique to a single application) to advance page by page, advance or back up a relative number of pages, and so on.
- A suspend data set. Assume a data collection task is in progress at a terminal. The task reads one or more units of input and then allows the terminal operator to interrupt the process by some kind of coded input. If not interrupted, the task repeats the data collection process. If interrupted, the task writes its "incomplete" data to temporary

storage and terminates. The terminal is now free to process a different transaction (perhaps a high-priority inquiry). When the terminal is available to continue data collection, the operator initiates the task in a "resume" mode, causing the task to recall its suspended data from temporary storage and continue as though it had not been interrupted.

- Preprinted forms. An application program can accept data to be written as output on a preprinted form. This data can be stored in temporary storage as it arrives. When all the data has been stored, it can first be validated and then sent to output in the order required by the format of the preprinted form.

WRITE DATA TO A TEMPORARY STORAGE QUEUE (WRITEQ TS)

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

Conditions: INVREQ, IOERR, ISCINVREQ,
ITEMERR, NOSPACE, QIDERR, SYSIDERR
```

This command is used to store temporary data (records) in a temporary storage queue in main or auxiliary storage. Data written to a temporary storage queue on a remote system will always be written to auxiliary storage.

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

If the ITEM option is specified, CICS/VS assigns an item number to the record in the queue, and sets the data area supplied in that option to the item number. If the record starts a new queue, the item number assigned is 1; subsequent item numbers follow on sequentially.

The REWRITE option specifies that records are to be updated, in which case the ITEM option must also be specified to identify the item (record) that is to be replaced by the data identified in the FROM option. If the specified queue exists, but the specified item cannot be found, the ITEMERR condition occurs. If the specified queue does not exist, the QIDERR condition occurs.

The maximum temporary storage record size is based on user-specified data set characteristics. (See the relevant CICS/VS System Programmer's Guide for details.)

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

```
EXEC CICS WRITEQ TS
     QUEUE(UNIQNAME)
     FROM(MESSAGE)
     LENGTH(LENGTH)
     ITEM(DREF)
```

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

```
EXEC CICS WRITEQ TS
     QUEUE('TEMPQ1')
     FROM(DATAFLD)
     LENGTH(40)
     ITEM(ITEMFLD)
     REWRITE
     MAIN
```

READ DATA FROM TEMPORARY STORAGE QUEUE (READQ TS)

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

Conditions: INVREQ, IOERR, ISCINVREQ,
ITEMERR, LENGERR, QIDERR, SYSIDERR
```

This command is used to retrieve data from a temporary storage queue in main or auxiliary storage. The queue is identified in the QUEUE option.

The INTO option specifies 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, 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 the record. The area is retained until another READQ TS command is executed. 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 (item) 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 in effect (either explicitly or by default), the next record after the last record to be retrieved (**by any task**) is retrieved. Therefore, if different tasks are to access the same queue and each task is to start at the beginning of the queue, the ITEM option must be used.

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
     QUEUE(UNIQNAME)
     INTO(DATA)
     LENGTH(LDATA)
```

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
     QUEUE(DESCRQ)
     SET(PREF)
     LENGTH(LENG)
     NEXT
```

DELETE TEMPORARY STORAGE QUEUE (DELETEQ TS)

```
DELETEQ TS QUEUE(name)
[SYSID(name)]

Conditions:  ISCINVREQ, QIDERR,
SYSIDERR
```

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

Temporary data should be deleted at the earliest possible time to avoid using excessive amounts of storage.

TEMPORARY STORAGE CONTROL OPTIONS

AUXILIARY
specifies that the temporary storage queue is on a direct-access storage device in auxiliary storage.

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

INTO(data-area)
specifies the 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, "parameter" must be a data area that is to be set to the item (record) 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, "parameter" 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 the length (as a halfword binary value) of the data to be used with WRITEQ TS and READQ TS commands.

For a WRITEQ TS command, the parameter must 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 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 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.

MAIN
specifies that the temporary storage queue is in main storage.

NEXT
specifies that the next sequential logical record following the last record to be retrieved (by any task) is to be retrieved.

QUEUE(name)
specifies the symbolic name of the queue to be written to, read from, or deleted. If the queue name appears in the TST, and the entry is marked as remote, the request is shipped to a remote system. 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.

REWRITE
specifies that the existing record in the queue is to be overwritten with the data provided. If the REWRITE option is specified, the ITEM option must also be specified. If the specified queue does not exist, the QIDERR condition occurs. If the correct item within an existing queue cannot be found, the ITEMERR condition occurs but the data is not stored.

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

SYSID(name) **(remote systems only)**
specifies the name of the system whose resources are to be used for intercommunication facilities. The name may be up to four characters in length.

TEMPORARY STORAGE CONTROL EXCEPTIONAL CONDITIONS

INVREQ
occurs when a WRITEQ TS command refers to data whose length is equal to zero or exceeds a certain size related to the size of the control interval of the auxiliary data set. (Refer to the relevant CICS/VS System Programmer's Guide for details.) This condition occurs also for a READQ TS command when the record to be retrieved has been created by a DFHTS TYPE=PUT macro.

Default action: terminate the task abnormally.

IOERR
occurs when there is an unrecoverable input/output error.

Default action: terminate the task abnormally.

ISCINVREQ
occurs when the remote system indicates a failure which does not correspond to a known condition.

Default action: terminate the task abnormally.

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).

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.

NOSPACE
occurs when insufficient space is available in the temporary storage queue to contain the data.

Default action: suspend the task until space becomes available as it is released by other tasks; then return normally.

QIDERR
occurs when the queue cannot be found, either in main storage or in auxiliary storage.

Default action: terminate the task abnormally.

SYSIDERR
occurs when the SYSID option specifies either a name which is not defined in the intersystem table, or a system to which the link is closed.

Default action: terminate the task abnormally.

Part 5. Recovery and Debugging

Chapter 5.1. Introduction to Recovery and Debugging

CICS/VS application programs are executed in an interactive environment. As a result, the operating system, CICS/VS itself, and the application programs must be responsive to many factors. Because the network on which the CICS/VS system is based consists of a variety of terminals and subsystems from which requests for services are received at random, the relationships between application programs and data set activity differ from one moment to the next.

CICS/VS provides the following aids to the testing, monitoring, and debugging of application programs:

- Execution (Command Level) Diagnostic Facility (EDF). Allows commands to be displayed in source form on a screen, both before and after execution so that they can be checked and altered if necessary. This facility is described in "Chapter 1.7. Execution (Command-Level) Diagnostic Facility" on page 35.
- Sequential terminal support. Enables sequential devices, such as card readers and disk units, to simulate online interactive terminals or subsystems of a CICS/VS network so that early testing can be carried out.
- Abnormal termination recovery. The HANDLE ABEND command can be used to deal with abnormal termination conditions, and the ABEND command can be used to cause a task to be terminated abnormally.
- Trace facility. A trace table containing entries that reflect the execution of various CICS/VS commands, and entries generated by application programs, can be written to main storage and, optionally, to an auxiliary storage device.
- Dump facility. Specified areas of main storage can be dumped onto a sequential data set, either tape or disk, for subsequent offline formatting and printing using a CICS/VS utility program.
- Journals. Facilities are provided for creating entries in special data sets called journals, for statistical or monitoring purposes; the system log is a journal.
- Recovery. When a task is abnormally terminated, CICS/VS can restore certain resources to their original state so that a transaction can be resubmitted for restart with no further action by the operator. The SYNCPOINT command can be used to subdivide a program so that only the uncompleted part of a transaction need be resubmitted.

Sequential terminal support, for which no special CICS/VS commands are required, is described below. The other facilities, and the commands that enable the application programmer to make use of them, are discussed in the other chapters of this part.

SEQUENTIAL TERMINAL SUPPORT

Even at the simplest level of program testing, the programmer should take the following into consideration. It is inefficient and error-prone to test a program from a terminal if all test data must be keyed into the system from that terminal for each test case. The programmer cannot easily retain a backlog of proven test data and quickly test programs through the key-driven terminal as changes are made.

CICS/VS allows the application programmer to begin testing his programs without the use of a telecommunication device. It is possible for the system programmer to specify through the terminal control table (TCT) that sequential devices be used as terminals. These sequential devices may be card readers, line printers, disk units, or magnetic tape units. In fact, the terminal control table can include combinations of sequential devices such as: card reader and line printer (CRLP), one or more disk or tape data sets as input, one or more disk or tape data sets as output. A TCT that contains references to these sequential terminals can also define other true telecommunications terminals in the system.

The input data submitted from a sequential device must be prepared in the form in which it would come from a telecommunication device. The input data must start with a transaction identification code of up to four characters, unless the transaction identification is predefined in the TCT. If there is more data, and the transaction identification code has less than four characters, a system-defined transaction code delimiter or a blank must precede the extra data. If a sequential device is being used as a terminal, an end-of-data indicator (a

0-2-8 punched card code (X'E0') or the equivalent as specified when the CICS/VS system is generated) must follow the input message or the system-defined data termination character. The input is processed sequentially and must be unblocked. The Sequential Access Method (SAM) is used to read and write the necessary inputs and outputs. The operating system utilities can be used to create the input data sets and print the output data sets.

Using this approach, it is possible to prepare a stream of transaction test cases to do the basic testing of a program module. As the testing progresses, the user can generate additional transaction streams to validate the multiprogramming capabilities of his programs or to allow transaction test cases to be run concurrently.

For operational convenience, it is usually appropriate to place a terminating transaction at the end of each input stream. For tests that use a single input stream, the transaction can be CSMT SHUTDOWN with appropriate responses following the initial message to respond to the CSMT queries about the mode of shutdown. In a batch-only testing environment, this enables CICS/VS to be terminated in an orderly manner without operator intervention.

Where more than one sequential input stream is used, only one should include the CSMT SHUTDOWN transaction. Others can be terminated with CSSF GOODNIGHT.

At some point in testing, it is necessary to use telecommunication devices to ensure that the transaction formats are satisfactory, that the terminal operational approach is satisfactory, and that the transactions can be processed on the terminal. The terminal control table can be altered to contain more and different devices as the testing requirements change.

When the testing has proved that transactions can be processed concurrently and the necessary data sets (actual or duplicate) for online operation have been created, the user begins testing in a controlled environment with the telecommunication devices. In this controlled environment, the transaction test cases should represent all functions of the eventual system, but on a smaller, measurable scale. For example, a company whose information system will work with 15 district offices may select one district office for the controlled test. During the controlled test, all transactions, data set activity, and output activity from the system should be monitored closely.

Requests for input or output from a sequential terminal are expressed by means of terminal control commands in the normal way. In response to a RECEIVE command, where the terminal has been described in the terminal control table as a CRLP, DISK, or TAPE terminal, data is read from the input data set until any one of the following situations occurs:

- An end-of-data indicator is detected in the input stream. (The indicator must be defined by the user when the CICS/VS system is generated.)
- Sufficient input has been read to fill the input area associated with the line used for transmission. If an end-of-data indicator is not detected before the input area is filled, all further data preceding an end-of-data indicator is bypassed and treated as a system error, which is passed to the user-installation terminal error program (DFHTEP).
- End-of-file (EOF) is detected. The input operation is considered complete. Any subsequent RECEIVE command is treated as a system error, which is passed to the user-installation terminal error program (DFHTEP) with a response code of 4. (In a CICS/DOS/VS system, EOF applies only to a card reader.)

In response to a SEND command for a CRLP terminal, lines are written in print format as follows:

- If there is no new-line (X'15') character within the number of characters contained in one print line of the specified line size (as defined by the system programmer in the LPLEN option of the DFHTCT TYPE=TERMINAL macro), the output is written in fixed-length lines of the size specified.
- If new-line characters are encountered, a new line is begun for each one.

Writing of output continues until the end of the user data is reached. For additional information concerning terminal control commands, refer to "Chapter 3.2. Terminal Control" on page 85.

Chapter 5.2. Abnormal Termination Recovery

During abnormal termination of a task, a program-level abend exit facility is provided in CICS/VS so that a user-written exit routine can be executed if desired. One example of a function performed by such a routine is the "clean-up" of a program that has started but not completed normally. An abend exit within an application program is activated in response to a HANDLE ABEND command. The same command can be used to cancel a previously activated exit.

The ABEND command can be used to abnormally terminate a task and so cause an active exit routine to be executed. The ABEND command can include a request for a dump.

A HANDLE ABEND command overrides any preceding such command in any application program at the same logical level. Each application program of a transaction can have its own exit, but only one exit at each logical level can be active. (Logical levels are explained in "Chapter 4.4. Program Control" on page 189.)

When a task is abnormally terminated, CICS/VS searches for an active exit, starting at the logical level of the application program in which the abend occurred, and proceeding, if necessary, to successively higher levels. The first active exit found, if any, is given control. This procedure is shown in Figure 21 on page 216, which also shows how subsequent abend exit processing is determined by the user-written exit routine.

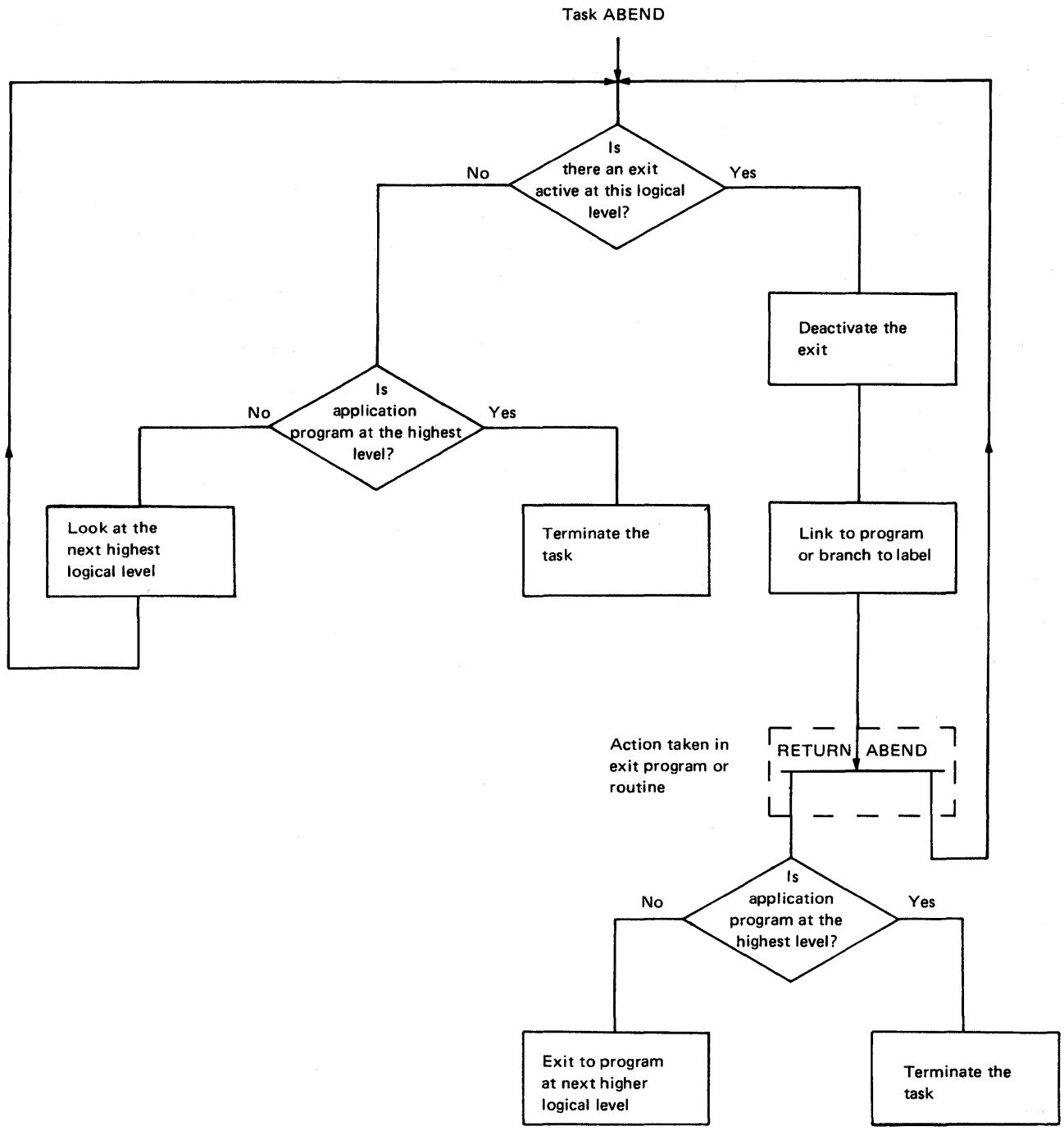


Figure 21. ABEND Exit Processing

To prevent recursive abends in an exit routine, CICS/VS deactivates an exit upon entry to the exit routine. If a retry of the operation is attempted, the application programmer can branch to a point in the program that was in control at the time of the abend and issue a HANDLE ABEND RESET command to reactivate the exit. This command can also be used to reactivate an exit (at the logical level of the issuing program) that was canceled previously as described above.

Refer to the section dealing with creation of task abend exits in the CICS/VS System Programmer's Reference Manual for additional information about exit routines, and to the CICS/VS Messages and Codes manual for a list of the transaction abend codes generated for abnormal terminations initiated by CICS/VS.

HANDLE AN ABNORMAL TERMINATION EXIT (HANDLE ABEND)

```
HANDLE ABEND {PROGRAM(name)|
LABEL(label)|CANCEL|RESET}

Condition:  PGMIDERR
 (if PROGRAM specified)
```

This command is used to activate, cancel, or reactivate an exit for abnormal termination processing.

When activating an exit, the PROGRAM option must be used to specify the name of a program to receive control, or (except for PL/I programs) the LABEL option must be used to specify a program label to which control will branch, when an abnormal termination condition occurs. A HANDLE ABEND PROGRAM or HANDLE ABEND LABEL command overrides any previous such request in any application program at the same logical level.

If intersystem communication is being used, an abnormal termination in the remote system may cause a branch to the specified program or label, but subsequent requests to use resources in the remote system will not be processed.

A HANDLE ABEND command with the CANCEL option will cancel a previously established exit at the logical level of the application program in control.

A HANDLE ABEND command with the RESET option will reactivate an abnormal termination exit that was canceled by a HANDLE ABEND CANCEL command or by CICS/VS. This command would usually be issued in an abnormal termination exit routine.

When an XCTL command is to be used to transfer control from an application program, there is a potential problem if an exit label (rather than a program) has been specified within that application program, and the exit is still active when control is transferred. If, later, CICS/VS needs more storage, the storage occupied by the application program may be reused, and an attempt to transfer control to the reused storage, as a result of a subsequent task abend, will have unpredictable results. This situation will not occur if an exit program is specified, instead of a label. Labels can be used without this risk in application programs that do not use an XCTL command.

If an abend occurs as a result of a CICS/VS BMS command, control will not be returned to CICS/VS to clean up the control blocks. Results will be unpredictable if the command is retried.

When the label specified in a HANDLE ABEND LABEL command receives control, the registers are set as follows:

ASM: R15 - Abend label.
R0-14 - Contents at the time of the last CICS/VS service request.

COBOL: Control returns to the HANDLE ABEND command with the registers restored; COBOL GO TO statement is then executed.

The following example shows how to establish a program as an exit:

```
EXEC CICS HANDLE ABEND
     PROGRAM('EXITPGM')
```

TERMINATE TASK ABNORMALLY (ABEND)

```
ABEND [ABCODE(name)]
[CANCEL]
```

This command is used to request that a task be terminated abnormally.

The main storage associated with the terminated task is released; optionally, a dump of this storage can be obtained first by using the ABCODE option to specify a four-character abnormal termination code, which CICS/VS will place in the formatted storage dump to identify it.

If the CANCEL option is specified, all abnormal termination exits established by HANDLE ABEND commands at any level in the task are canceled before the task is terminated. If the PL/I STAE execution-time option has been specified, an abnormal termination exit will have been established by PL/I. This exit is revoked by the CANCEL option. (Refer to the PL/I Optimizing Compiler Programmer's Guide for further information.)

The following example shows how to terminate a task abnormally:

```
EXEC CICS ABEND
     ABCODE(ABCD)
```

ABNORMAL TERMINATION RECOVERY OPTIONS

ABCODE(name)
specifies that main storage related to the task that is being terminated is to be dumped and provides a name to identify the dump. The specified name may consist of up to four characters.

CANCEL
specifies that exits established by HANDLE ABEND and ABEND commands are to be canceled; in effect they are ignored. A HANDLE ABEND CANCEL command cancels a previously established exit at the logical level of the application program in control. An ABEND CANCEL command cancels all exits at any level in the task (and terminates the task abnormally).

LABEL(label)
specifies the program label to which control will branch if abnormal termination occurs. This option cannot be used for PL/I application programs.

PROGRAM(name)
specifies the name of the program to which control is to be passed if the task is terminated abnormally. The name can consist of up to eight alphameric characters and must have been defined in the processing program table (PPT).

RESET
specifies that an exit canceled by a HANDLE ABEND CANCEL command is to be reactivated.

ABNORMAL TERMINATION RECOVERY EXCEPTIONAL CONDITIONS

PGMIDERR
occurs if a program cannot be found in the PPT or is disabled.

Default action: terminate the task abnormally.

Chapter 5.3. Trace Control

The CICS/VS trace control program is a debugging and monitoring aid for application programmers and IBM field engineers. This facility makes use of a trace table consisting of entries produced in response to trace control requests; the trace table resides in main storage. The CICS/VS auxiliary trace facility allows trace records to be written on a sequential device for later analysis.

Trace control commands are provided to:

- Specify user trace entry point or user event monitoring point (ENTER).
- Control the CICS/VS trace facility (TRACE ON and TRACE OFF).

TRACE ENTRY POINTS

The points at which trace entries are produced during CICS/VS operation are of two types: system trace entry points and user trace entry points.

System trace entry points: points within CICS/VS at which trace control requests are made. The only system trace entry points that need concern the command-level application programmer are the EXEC-interface-program trace points, which produce entries in the trace table whenever a CICS/VS command is executed. Two trace entries are made: the first when the command is issued, and the second when CICS/VS has performed the required function and is about to return control to the application program. Between them, these two trace entries allow the flow of control through an application program to be traced, and a check to be made on which exceptional conditions occurred during its execution. (The TRACE ON, TRACE OFF, ABEND, XCTL, and RETURN commands produce single entries.)

User trace entry points: additional points within an application program that need to be included in the trace table to allow complete program debugging. For example, a program loop would need a trace entry to be produced containing a counter value showing the number of times that the loop had been entered. User trace entries are produced wherever an ENTER command is issued. Each trace entry request can be given a unique identifer and can cause 8 bytes of data to be placed in the trace table.

EVENT MONITORING POINTS

A user event monitoring point (EMP) can be defined in an application program by means of the MONITOR option of the ENTER command. At a user EMP, information can be added to the user fields in accounting and performance class monitoring data records. The classes of data records to be eligible for the addition of user information are specified by the ACCOUNT and PERFORM options. The actual user information to be recorded is defined in the monitoring control table. The user information recorded, in conjunction with similar data recorded automatically by the system can be used as input to offline analysis and reporting programs. More information on the use of monitoring is given in the CICS/VS System Programmer's Reference Manual.

TRACE FACILITY CONTROL

The CICS/VS trace facility is controlled by a number of trace flags; the flags are stored within CICS/VS and the TRACE ON and TRACE OFF commands are used to turn them on or off.

There is a master system trace flag, which must be on before any system trace entries are produced, and a separate system flag for each type of system trace entry. The master system trace flag can be turned on or off independently of individual system trace flags; thus the system trace pattern of activity can be left intact but controlled as a single unit. When the master system trace flag and one or more system trace flags are on, the relevant system trace entries are produced for all active tasks, and tasks that become active subsequently, until the flags are turned off again.

The TRACE command can be used to control the system trace flags for other parts of CICS/VS, should it be necessary to debug a program down to the level of the CICS/VS macro instructions issued by the EXEC interface program; for further details, see later in this chapter under "Control the CICS/VS Trace Facility".

There is a master user trace flag, and an individual user trace flag for each task. If the master user trace flag is on, requested user trace entries are produced for all active tasks, and tasks that become active subsequently, until the flag is turned off again. Each individual user trace flag controls user trace entries only for the task that turns the flag on or off.

The master terminal operator can turn the whole CICS/VS trace facility on or off by entering suitable instructions; all flags are turned on or off together when this method is used.

TRACE TABLE FORMAT

The CICS/VS trace table is located in main storage; it is possible to gain access to it by investigating a dump. The trace table consists of a trace header and a variable number of fixed-length entries produced by trace control requests. Each entry in the trace table is 16 bytes in length and is aligned on a double-doubleword boundary. The trace table area is of a fixed size specified by the system programmer, and entries are placed in the table in a wraparound manner; that is, when the table is full, the next entry is placed at the head of the table, overwriting the original entry. The format of the trace header is:

Bytes	Contents
0-3	Address of last-used entry
4-7	Address of start of table
8-11	Address of end of table
12-15	Reserved

The format of the EXEC interface program trace entry on issuance of a command is as follows:

Bytes	Contents
0	X'E1' trace identifier.
1-3	Return point in application program.
4	Not used.
5(0-3)	X'0', identifying first entry for command.
5(4-7)	Not used.
6,7	User task sequence number (packed decimal).
8-11	ASM: address of dynamic storage addressed by DFHEISTG DSECT. COBOL: address of working-storage section. PL/I: address of dynamic storage area (DSA)
12,13	Not used.
14,15	Code identifying CICS/VS command. See field EIBFN in Appendix A for details.

The format of the EXEC interface program trace entry on completion of a command is as follows:

Bytes	Contents
0	X'E1' trace identifier.
1-3	Return point in application program; if response code in bytes 8-13 is non-zero, these bytes will contain address of the label specified in HANDLE CONDITION command associated with response.
4	EIBGDI
5(0-3)	X'F', identifying second entry for command.
5(4-7)	Not used.
6,7	User task sequence number (packed decimal).
8-13	Response code. Zero response code signifies that no exceptional conditions occurred during execution of command. If response is non-zero, see field EIBRCODE in Appendix A for details.
14,15	Code identifying CICS/VS command (same as bytes 14 and 15 at issuance of command).

The format of a user trace entry is as follows:

Bytes	Contents
0	Trace identifier, being binary value specified in ENTER command.
1-3	Return point in application program.
4	Not used.
5(0-3)	Not used.
5(4-7)	X'2', identifying this entry as a user trace entry.
6,7	User task sequence number (packed decimal).
8-15	Data field supplied in ENTER command.

If consecutive, duplicate entries for the trace table are generated, the first entry has the form of a standard entry, but subsequent identical entries are replaced by a single special entry, immediately following the first entry. The trace identifier of this special

entry (in byte 0) is X'FD'; bytes 1-3 contain a packed decimal number that shows how many repeated entries have been replaced by this single entry. Trace table entries with the trace identifiers X'FE' or X'FF' indicate the turning on or turning off, respectively, of the trace facility. Details of these and other CICS/VS trace entries are given in the CICS/VS Problem Determination Guide.

CICS/VS AUXILIARY TRACE FACILITY

All trace entries that are written to the trace table can also be written to the auxiliary trace data set (provided that the auxiliary trace program has been generated and has been activated by the master terminal operator). Whereas the entries written to the trace table wrap around, the auxiliary trace data set contains all of the trace table entries that have been made. The CICS/VS Trace Utility Program (DFHTUP) can be used to process and print selected trace entries from the data set (for example, all the EXEC-interface-program trace entries). The printout also shows the time at which each trace entry was produced.

USER TRACE ENTRY POINT AND EVENT MONITORING POINT (ENTER)

```
ENTER TRACEID(data-value)
[FROM(data-area)]
[ACCOUNT]
[MONITOR]
[PERFORM]
```

This command is used to specify a point within an application program at which a user trace table entry is to be produced (if the trace facility has been turned on for this type of entry).

This command is used also to define a user event monitoring point (specify MONITOR). The classes of monitoring data for which user information is to be collected at this user event monitoring point can be specified by the ACCOUNT and PERFORM options.

A trace identifier in the range 0 through 199 must be provided in the TRACEID option; this will appear in the first byte of the trace table entry that is produced. Optionally, 8 bytes of data can be supplied in the FROM option; this data will appear in bytes 8-15 of the trace table entry.

For a user event monitoring point, the TRACEID specified should match the identification number of a TYPE=EMP entry in the monitoring control table that defines the user information to be collected. If no such entry exists, the ENTER command will have no effect. This provides a way of coding optional recording points which are activated by the use of an appropriate monitoring control table.

If both the ACCOUNT and PERFORM options are specified in the application program, the corresponding entry in the monitoring control table can specify recording of either accounting or performance data, or both. If only one option is specified at the user EMP, only that class of recording is possible. Thus greater flexibility is obtained by specifying both options for the user EMP and controlling run-time activity by a suitably coded monitoring control table.

The following example shows how to specify that a user trace table entry should be produced:

```
EXEC CICS ENTER
     TRACEID(123)
     FROM(MSG)
```

CONTROL THE CICS/VS TRACE FACILITY (TRACE ON, TRACE OFF)

```
TRACE {ON|OFF} [SYSTEM]
[EI]
[USER]
[SINGLE]
```

These commands are used to control the CICS/VS trace facility by turning on and off the various trace flags. (See the section "Trace Facility Control" earlier in this chapter for details of trace flags.)

A TRACE ON or TRACE OFF command without options controls the entire CICS/VS trace facility but leaves the established pattern of trace activity undisturbed.

The SYSTEM option controls the master system trace flag, which must be on before any system trace table entries are produced. The EI option controls the EXEC-interface-program system trace flag. The USER option controls the master user trace flag, and the SINGLE option controls the user trace flag for the task.

The following example shows how to turn on the master system and EXEC-interface-program system trace flags to start tracing of CICS/VS commands:

```
EXEC CICS TRACE ON
     SYSTEM
     EI
```

MACRO-LEVEL TRACE FACILITIES

If debugging at the macro level is necessary, an additional option, ALL, can be used, specifying that the entire CICS/VS trace facility is to be controlled by the TRACE ON and TRACE OFF commands. It has the same effect as a master terminal trace control instruction and affects all master, system, and user trace flags.

The following options can only be used in conjunction with the SYSTEM option but no system trace entries will be produced unless the master system trace flag is on. Each option specifies that the system trace entries produced by the associated program are controlled by the TRACE ON and TRACE OFF commands. The options can be specified in any combination and in any order.

Option	CICS/VS Program
BF	Built-in Function
BM	Basic Mapping Support
DC	Dump Control
DI	Batch Data Interchange
FC	File Control
IC	Interval Control
IS	ISC
JC	Journal Control
KC	Task Control
PC	Program Control
SC	Storage Control
SP	Sync Point
TC	Terminal Control
TD	Transient Data
TS	Temporary Storage
UE	User Exit Interface

TRACE CONTROL OPTIONS

ACCOUNT
specifies, for a user event monitoring point, that user information is to be collected in the accounting class monitoring data records.

EI
specifies that tracing of CICS/VS commands through the EXEC interface program is affected by the TRACE ON or TRACE OFF command.

FROM(data-area)
specifies an 8-byte data area whose contents are to be entered into the data field of the trace table entry. When used for monitoring, the data area is regarded as two successive fullword fields. These correspond, in order, to the keywords DATA1 and DATA2 that can be specified in the DFHMCT TYPE=EMP system macro. If the FROM option is omitted, two fullwords of binary zeros are passed as the values of DATA1 and DATA2.

MONITOR
specifies that a user event monitoring point, rather than a trace entry point, is to be recorded.

PERFORM
specifies, for a user event monitoring point, that user information is to be collected in the performance class monitoring data records.

SINGLE
specifies that the TRACE ON or TRACE OFF command applies to user entries of the single task issuing the request for the duration of the task.

SYSTEM
specifies that all trace entries made from within CICS/VS are affected by the TRACE ON or TRACE OFF command.

This option controls the master system trace flag but does not change the status of individual system trace flags; the established pattern of system trace activity remains intact but is controlled as a single unit. (This characteristic is useful when macro-level trace facilities are in use, as described earlier in this chapter.)

TRACEID(data-value)
specifies the trace identifier for a user trace table entry as a halfword binary value in the range 0 through 199. When used for monitoring, the data value is the user-event monitoring point identifier as specified in the DFHMCT TYPE=EMP system macro.

USER
specifies that all user entries for all current transactions are affected by the TRACE ON or TRACE OFF command.

TRACE CONTROL EXCEPTIONAL CONDITIONS

There are no trace control exceptional conditions.

Chapter 5.4. Dump Control

The CICS/VS dump control program allows specified areas of main storage to be dumped, by means of the DUMP command, onto a sequential data set, which can be either on tape or on disk. This data set contains only the information applicable to the user's transaction or application program, and can be formatted subsequently and printed offline (or while the dump data set is closed) using the CICS/VS Dump Utility Program (DFHDUP).

Only one dump control command is processed at a time. If additional commands are issued while a dump is in progress, activity within the tasks associated with those commands is suspended until the dump is completed. Remaining dump commands are processed in the order in which they are made. The use of the DUMP command will cause certain fields (for example, EIBFN and EIBRCODE) in the EIB and the TCA to be overwritten.

Options of the DUMP command allow the following areas of main storage to be dumped in various combinations:

- Selected main storage areas related to the requesting task. A dump of these areas is normally used during the testing and debugging of an application program. (CICS/VS automatically provides this service if the related task is terminated abnormally.)
- CICS/VS tables: program control table (PCT), processing program table (PPT), system initialization table (SIT), terminal control table (TCT), file control table (FCT), destination control table (DCT). A dump of these tables is typically the first dump taken in a test in which the base of the test must be established; subsequent dumps are usually of the task-related-storage type.
- Task-related storage areas and CICS/VS control tables (a complete dump). To request a complete dump is sometimes appropriate during execution of a task, but this facility should not be used excessively. CICS/VS control tables are primarily static areas; therefore, requesting one CICS/VS-tables dump and a number of task-related-storage dumps is generally more efficient than requesting a comparable number of complete dumps.

DUMP MAIN STORAGE (DUMP)

```
DUMP DUMPCODE(name)
[FROM(data-area) LENGTH(data-value)]
[COMPLETE]
[TASK]
[STORAGE]
[PROGRAM]
[TERMINAL]
[TABLES]
[DCT] [FCT] [PCT] [PPT] [SIT] [TCT]
```

This command is used to dump any or all of the main storage areas related to a task, any or all of the CICS/VS tables (FCT, DCT, PCT, PPT, SIT, TCT), or all of these together.

The following example shows how to request a dump of the entire task-related storage areas, the terminal control table, and a specified data area:

```
EXEC CICS DUMP
     TASK
     TCT
     FROM(AREA1)
     LENGTH(200)
     DUMPCODE('DUM1')
```

DUMP CONTROL OPTIONS

The dump control options can be specified in any combination; only one copy of each area or table will be dumped, even if specified more than once.

If no options are specified, the areas dumped will be the same as those dumped when the TASK option is specified, except that the DL/I control blocks will not be dumped.

COMPLETE
dumps all main storage areas related to a task, all of the CICS/VS tables, and for CICS/OS/VS only, the DL/I control blocks.

DCT
dumps the destination control table

DUMPCODE(name)
specifies a name (up to four characters) that identifies the dump.

FCT
dumps the file control table.

FROM(data-area)
dumps the specified data area which must be a valid area, that is, storage allocated by the operating system within the CICS/VS region or partition. In addition, the following areas are dumped:

- Task control area (TCA) and, if applicable, the transaction work area (TWA).
- Common system area (CSA), including the user's portion of the CSA (CWA).
- Trace table.
- Contents of general-purpose registers upon entry to dump control from the requesting task.
- Either the terminal control table terminal entry (TCTTE) or the destination control table entry associated with the requesting task.

Whenever the TCTTE is dumped, the terminal control table user area (if any) and the message control blocks (if any) associated with the TCTTE are dumped. The latter are used by basic mapping support.

LENGTH
specifies the length (halfword binary) of the data area specified in the FROM option.

PCT
dumps the program control table.

PPT
dumps the processing program table.

PROGRAM
specifies that program storage areas associated with this task are to be dumped, as follows:

- Task control area (TCA) and, if applicable, the transaction work area (TWA).
- Common system area (CSA), including the user's portion of the CSA (CWA).
- Trace table.
- All program storage areas containing user-written application program(s) active on behalf of the requesting task.
- Register save areas (RSAs) indicated by the RSA chain off the TCA.
- Contents of general-purpose registers upon entry to dump control from the requesting task.
- Either the terminal control table terminal entry (TCTTE) or the destination control table entry associated with the requesting task.

Whenever the TCTTE is dumped, the terminal control table user area (if any) and the message control blocks (if any) associated with the TCTTE are dumped.

SIT
dumps the system initialization table.

STORAGE
specifies that storage areas associated with this task are to be dumped, as follows:

- Task control area (TCA) and, if applicable, the transaction work area (TWA).
- Common system area (CSA), including the user's portion of the CSA (CWA).
- Trace table.
- Contents of general-purpose registers upon entry to dump control from the requesting task.
- All transaction storage areas chained off the TCA storage accounting field.
- Either the terminal control table terminal entry (TCTTE) or the destination control table entry associated with the requesting task.

Whenever the TCTTE is dumped, the terminal control table user area (if any) and the message control blocks (if any) associated with the TCTTE are dumped.

TABLES
dumps the DCT, FCT, PCT, PPT, SIT, and the TCT.

TASK
specifies that storage areas associated with this task are to be dumped, as follows:

- Task control area (TCA) and, if applicable, the transaction work area (TWA).
- Common system area (CSA), including the user's portion of the CSA (CWA).

- Trace table.
- All program storage areas containing user-written application programs active on behalf of the requesting task.
- Contents of general-purpose registers upon entry to dump control from the requesting task.
- All transaction storage areas chained off the TCA storage accounting field.
- Either the terminal control table terminal entry (TCTTE) or the destination control table entry associated with the requesting task.
- Register save areas (RSAs) indicated by the RSA chain off the TCA.
- All terminal input/output areas (TIOAs) chained off the terminal control table terminal entry (TCTTE) for the terminal associated with the requesting task.
- DL/I control blocks (CICS/OS/VS only).

Whenever the TCTTE is dumped, the terminal control table user area (if any) and the message control blocks (if any) associated with the TCTTE are dumped.

TCT

dumps the terminal control table.

TERMINAL

specifies that storage areas associated with the terminal are to be dumped, as follows:

- Task control area (TCA) and, if applicable, the transaction work area (TWA).
- Common system area (CSA), including the user's portion of the CSA (CWA).
- Trace table.
- All terminal input/output areas (TIOAs) chained off the terminal control table terminal entry (TCTTE) for the terminal associated with the requesting task.
- Contents of general-purpose registers upon entry to dump control from the requesting task.
- Either the terminal control table terminal entry (TCTTE) or the destination control table entry associated with the requesting task.

Whenever the TCTTE is dumped, the terminal control table user area (if any) and the message control blocks (if any) associated with the TCTTE are dumped. The latter area used by basic mapping support.

DUMP CONTROL EXCEPTIONAL CONDITIONS

There are no dump control exceptional conditions.

Chapter 5.5. Journal Control

CICS/VS provides facilities for creating and managing special-purpose sequential data sets, called **journals**, during CICS/VS execution. Journals may contain any and all data the user needs to facilitate subsequent reconstruction of events or data changes. For example, a journal might act as an audit trail, a change-file of data-base updates and additions, or a record of transactions passing through the system (often called a **log**). Each journal can be written from any task.

Only the CICS/VS facilities dealing with creation of journals (journal output) using journal control commands are dealt with in this manual; the CICS/VS System Programmer's Reference Manual contains information about reading journal data sets (journal input), which involves the use of CICS/VS journal control macro instructions.

Journal control commands are provided to allow the application programmer to:

- Create a journal record (JOURNAL).
- Synchronize with (wait for completion of) journal output (WAIT JOURNAL).

Exceptional conditions that occur during execution of a journal control command are handled as described in "Chapter 1.5. Exceptional Conditions" on page 25.

Journal Records

Data may be directed to any journal data set specified in the journal control table (JCT), which defines the journals available during a particular CICS/VS execution. The JCT may define one or more journals on tape or direct access storage. Each journal is identified by a number known as the journal file identifier. This number may range from 2 through 99; the value 1 is reserved for a journal known as the system log.

When a journal record is built, the data is moved to the journal buffer area. All buffer space and other work areas needed for journal data set operations are acquired and managed by CICS/VS. The user task supplies only the data to be written to the journal.

Journal records are built into blocks compatible with standard variable-blocked format. CICS/VS uses the host operating system's Sequential Access Method to write the blocks to auxiliary storage.

Each journal record begins with a standard fullword length field, a user-specified identifier, and a system-supplied prefix. This data is followed in the journal record by any user-supplied prefix data (optional), and finally by the user-specified data. Journal control is designed so that the application programmer requesting output services need not be concerned further with the detailed layout and precise contents of journal records. He needs to know only which journal to use, what user data to specify, and what unique user-identifier to supply.

Journal Output Synchronization

When a journal record is created by issuing the JOURNAL command with the WAIT option, the requesting task can wait until the output has been completed. By specifying that this should happen, the application programmer ensures that the journal record is written on the external storage device associated with the journal before processing continues; the task is said to be **synchronized** with the output operation.

The application programmer can also request asynchronous journal output. This causes a journal record to be created in the journal buffer area and, optionally, initiates the data output operation from the buffer to the external device, but allows the requesting task to retain control and thus to continue with other processing. The task may check and wait for output completion (that is, synchronize) at some later time by issuing the WAIT JOURNAL command.

The basic process of building journal records in the buffer space of a given journal continues until one of the following situations occurs:

- A request specifying the STARTIO option is made (from any task) for output of a journal record.
- A request is rejected because of insufficient journal buffer space.
- The available buffer space is reduced below a level that is specified by the system programmer.
- One second elapses after the last occasion on which any task started writing to this journal buffer.

When any of these situations occurs, all journal records present in the buffer,

including any deferred output resulting from asynchronous requests, are written to auxiliary storage as one block.

The advantages that may be gained by deferring journal output are:

- Transactions may get better response times by waiting less.
- The load of physical I/O requests on the host system may be reduced.
- Journal data sets may contain fewer but larger blocks and so better utilize auxiliary storage devices.

However, these advantages are achievable only at the cost of more buffer space and greater programming complexity. It is necessary to plan and program to control synchronizing with journal output. Additional decisions that depend on the data content of the journal record and how it is to be used must be made in the application program. In any case, the full benefit of deferring journal output is obtained only when the load on the journal data set is high.

The STARTIO option is used with JOURNAL output requests to specify that the journal output operation is to be initiated immediately. For asynchronous output requests, control returns directly to the requesting program. The STARTIO option should not be used unnecessarily because, if every journal request used STARTIO, no improvement over synchronous output requests, in terms of reducing the number of physical I/O operations and increasing the average block size, would be possible.

If the journal buffer space available at the time of the request is not sufficient to contain the journal record, the NOJBUFSP exceptional condition occurs. If no HANDLE CONDITION request is active for this condition, the requesting task loses control, the contents of the current buffer are written out, and the journal record is built in the resulting freed buffer space before control returns to the requesting task.

If the requesting task is not willing to lose control (for example, if some housekeeping must be performed before other tasks get control), a HANDLE CONDITION command should be issued. If the NOJBUFSP condition occurs, no journal record is built for the request, and control is returned directly to the requesting program at the location provided in the HANDLE CONDITION request. The requesting program can perform any housekeeping needed before reissuing the journal output request.

CREATE A JOURNAL RECORD (JOURNAL)

```
JOURNAL
JFILEID(data-value)
JTYPEID(data-value)
FROM(data-area)
LENGTH(data-value)
[REQID(data-area)]
[PREFIX(data-value)
 PFXLENG(data-value)]
[STARTIO]
[WAIT]

Conditions:  JIDERR, IOERR, LENGERR,
NOJBUFSP, NOTOPEN
```

This command is used to create a journal record. The request can be for synchronous or asynchronous output; definitions of these terms, and detailed information regarding the synchronization of journal output, are contained in the section "Journal Output Synchronization," earlier in this chapter. The following options must be specified.

- JFILEID specifies the journal data set to receive the data. (JFILEID(1) specifies the system log.)
- JTYPEID specifies a two-character identifier for the journal record.
- FROM specifies the user data to be included in the journal record.
- LENGTH specifies the length of the user data. This length should include the amount of space reserved for CICS/VS use within the maximum defined by the BUFSIZE operand of the DFHJCT TYPE=ENTRY system macro.

The following are optional:

- PREFIX specifies the user prefix data for the journal record.
- PFXLENG specifies the length of the prefix data.

To request synchronous journal output the WAIT option must be specified. For asynchronous output, (WAIT option not specified), the REQID option can be included to provide a unique identifier for the journal record; the identifier can be used later in a WAIT JOURNAL command to synchronize the task with the creation of the journal record.

The STARTIO option can be included in a synchronous or asynchronous request to specify that the journal output operation should start immediately. STARTIO reduces absolute waiting time at the expense of general system performance and input/output load.

The following example shows how to request synchronous journal output and wait for the output operation to be completed:

```
EXEC CICS JOURNAL
     JFILEID(2)
     JTYPEID('XX')
     FROM(KEYDATA)
     LENGTH(8)
     PREFIX(PROGNAME)
     PFXLENG(6)
     WAIT
```

In this example, since STARTIO is not specified, the task will wait until the journal buffer is full or until output is initiated by a STARTIO request in another task. CICS/VS limits the wait to one second.

The following example shows how to request deferred (asynchronous) journal output:

```
EXEC CICS JOURNAL
     FROM(COMDATA)
     LENGTH(10)
     JFILEID(1)
     JTYPEID('SD')
     REQID(ENTRYID)
```

SYNCHRONIZE WITH JOURNAL OUTPUT (WAIT JOURNAL)

```
WAIT JOURNAL
JFILEID(data-value)
[REQID(data-value)]
[STARTIO]

Conditions:  JIDERR, INVREQ, IOERR,
NOTOPEN
```

This command is used to synchronize the task with the output of a one or more journal records that have been created but whose output has been deferred; that is, with asynchronous journal output requests.

The JFILEID option specifies the journal file identifier, and the REQID option optionally specifies a particular journal record. If the REQID option is not specified, the task is synchronized with the output of the the last record created for the journal specified in the JFILEID option.

The journal records in the journal buffer area may already be written out to auxiliary storage, or the journal record output operation may be in progress. If the output operation has already been completed, control returns immediately to the requesting task; if not, the requesting task waits until the operation has been completed. If STARTIO is specified, output is initiated immediately.

If the requesting program has made a succession of successful asynchronous output requests to the same journal data set, it is necessary to synchronize on only the last of these requests to ensure that all of the journal records have reached auxiliary storage. This may be done either by issuing a stand-alone WAIT JOURNAL command, or by making the last output command itself synchronous (by specifying the WAIT option in the JOURNAL command).

The following example shows how to request synchronization with the output of a journal record:

```
EXEC CICS WAIT JOURNAL
     JFILEID(4)
     REQID(ENTRYID)
```

JOURNAL CONTROL OPTIONS

FROM(data-area)
specifies the user data to be built into the journal record.

JFILEID(data-value)
specifies a halfword numeric value in the range 1 through 99 to be taken as the journal file identifier. The value 1 specifies that the system log data set is the journal for this operation.

JTYPEID(data-value)
specifies a two-character identifier to be placed in the journal record to identify its origin.

LENGTH(data-value)
specifies as a halfword binary value the length in bytes of the user data to be built into the journal record. The minimum value is 1, and the maximum value is such that the sum of the LENGTH and PFXLENG values does not exceed the journal buffer size specified by the system programmer.

PFXLENG(data-value)
specifies as a halfword binary value the length in bytes of the user prefix data to be included in the journal record. The minimum value is 1, and the maximum value is such that the sum of the LENGTH and PFXLENG values does not exceed the journal buffer size specified by the system programmer.

PREFIX(data-value)
specifies the user prefix data to be included in the journal record. A data area must be provided in COBOL programs.

REQID(parameter)
specifies a fullword binary variable. For a JOURNAL command, the REQID option specifies that asynchronous output is required; the parameter must be a data area. CICS/VS sets the variable to a unique value to identify the journal record that is created.

When used with a WAIT JOURNAL command, the REQID option specifies a variable set to a number that identifies the journal record that has been created but possibly not yet written out; the parameter is a data value.

STARTIO
specifies that output of the journal record is to be initiated immediately. If WAIT is specified for a journal with a low utilization, STARTIO should be specified also to prevent the requesting task waiting for the journal buffer to be filled. Very high utilization ensures that the buffer is flushed quickly, so that STARTIO is unnecessary.

WAIT
specifies that synchronous journal output is required. The journal record is written out; the requesting task waits until the record has been written.

JOURNAL CONTROL EXCEPTIONAL CONDITIONS

INVREQ
occurs if a WAIT JOURNAL command is issued before any JOURNAL command has been issued in the same task.

Default action: terminate the task abnormally.

IOERR
occurs if the physical output of a journal record was not accomplished because of an unrecoverable I/O error.

Default action: terminate the task abnormally.

JIDERR
occurs if the specified journal file identifier does not exist in the journal control table (JCT).

Default action: terminate the task abnormally.

LENGERR
occurs if the computed length for the journal record exceeds the total buffer space allocated for the journal data set, as specified in the journal control table (JCT) entry for the data set.

Default action: terminate the task abnormally.

NOJBUFSP
occurs if the journal buffer space allocated by the system programmer is not sufficient to contain a journal record.

Default action: write out the contents of the current buffer; suspend task activity until the JOURNAL command is satisfied.

NOTOPEN
occurs if the journal command cannot be satisfied because the specified journal data set has been disabled and is not available.

Default action: terminate the task abnormally.

Chapter 5.6. Recovery (Sync Points)

To facilitate recovery in the event of abnormal termination of a CICS/VS task or of failure of the CICS/VS system, the system programmer can during CICS/VS table generation define certain resources (for example, files) as recoverable. If a task is terminated abnormally, these resources are restored to the condition they were in at the start of the task, which can then be rerun. The process of restoring the resources associated with a task is termed **backout**.

If an individual task fails, backout is performed by the dynamic transaction backout program. If the CICS/VS system fails, backout is performed as part of the emergency-restart process. The CICS/VS System/Application Design Guide and the CICS/VS System Programmer's Reference Manual describe these facilities, which in general have no effect on the coding of application programs.

However, for long-running programs, it may be undesirable to have a large number of changes, accumulated over a period of time, exposed to the possibility of backout in the event of task or system failure. This possibility can be avoided by using the SYNCPOINT command to split the program into logically separate sections termed **logical units of work (LUWs)**; the end of an LUW is called a synchronization point (**sync point**).

In addition to those defined with the SYNCPOINT command, sync points also occur at the end of a task and at each DL/I termination or checkpoint (CHKP) call. For the purposes of backout, each of these sync points is treated as though it marked the end of a task: if failure occurs after a sync point but before the task has been completed, only changes made since the sync point are backed out.

It is recommended that LUWs be entirely logically independent, not merely with regard to protected resources, but also with regard to execution flow. Typically, an LUW would comprise a complete conversational operation bounded by SEND and RECEIVE commands.

In addition to a DL/I termination call being considered to be a sync point, the execution of a SYNCPOINT command will cause CICS/VS to issue a DL/I termination call. If a DL/I PSB is required in a subsequent LUW, it must be rescheduled by means of a PCB call.

A BMS logical message started but not completed when a SYNCPOINT command is executed is forced to completion by an implied SEND PAGE command.

The system programmer should be consulted if sync points are to be issued in a transaction that is eligible for transaction restart.

ESTABLISH A SYNC POINT (SYNCPOINT)

```
SYNCPOINT [ROLLBACK]
```

This command is used to divide a task (usually a long-running one) into smaller units known as logical units of work (LUWs). Each SYNCPOINT command causes a sync point to be established to mark the completion of a logical unit of work.

SYNC POINT OPTION

ROLLBACK
specifies that all changes to recoverable resources, made by the task since its last sync point, are to be backed out. This option must not be used if remote recoverable resources have been updated in the same LUW (because those resources will not be backed out). In order to backout any remote recoverable resources, the transaction could be abended.

This option can be used, for example, to tidy up in a HANDLE ABEND routine, or to revoke data base changes after the application program finds unrecoverable errors in its input data.

Part 6. The CICS/VS Built-In Function Command

Chapter 6.1. The Field Edit Built-In Function (BIF DEEDIT) Command

The built-in function, DEEDIT, is provided by means of the BIF DEEDIT command.

```
BIF DEEDIT
FIELD(data-area)
LENGTH(data-value)
```

This command specifies that alphabetic and special characters are to be removed from an EBCDIC data field, the remaining digits being right justified and padded left with zeros as necessary. This field is specified by the FIELD option and its length, in bytes, by the LENGTH option.

If the field ends with a minus sign or a 'CR', a negative zone (X'D') is placed over the rightmost (low-order) byte. The zone portion of the rightmost byte may contain one of the hexadecimal characters X'A' through X'F'; the digit portion may contain one of the hexadecimal digits from X'0' through X'9'. Where this is the case, the rightmost byte is returned unaltered (see the example below). This permits the application program to operate on a zoned numeric field. The returned value is in the field that initially contained the unedited data.

For example, execution of the command:

```
EXEC CICS BIF
     DEEDIT
     FIELD(CONTG)
     LENGTH(9)
```

removes all characters other than EBCDIC digits from CONTG, a nine-byte field, and returns the edited result in that field to the application program. Two examples of the contents of CONTG before and after execution of the command are:

Original value	Returned value
14-6704/B	00146704B
$25.68	000002568

Note that a decimal point is an EBCDIC special character and as such is removed.

There are no exceptional conditions with DEEDIT.

Part 7. Appendixes

Appendix A. EXEC Interface Block

This appendix describes the fields of the EXEC interface block (EIB) referred to in "Chapter 1.6. Access to System Information" on page 29. An application program can access all of the fields in the EIB of the associated task 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. Fields are listed in alphabetical order.

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.

ASM: CL1
COBOL: PIC X(1)
PL/I: CHAR(1)

EIBATT
indicates that the RU contains attach header data (X'FF').

ASM: CL1
COBOL: PIC X(1)
PL/I: CHAR(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 is passed, this field contains zeros.

ASM: H
COBOL: PIC S9(4) COMP
PL/I: FIXED BIN(15)

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.

ASM: H
COBOL: PIC S9(4) COMP
PL/I: FIXED BIN(15)

EIBDATE
contains the date the task is started (this field is updated by the ASKTIME command). The date is in packed decimal form (00YYDDD+).

ASM: PL4
COBOL: PIC S9(7) COMP-3
PL/I: FIXED DEC(7,0)

EIBDS
contains the symbolic identifier of the last data set referred to in a file control request.

ASM: CL8
COBOL: PIC X(8)
PL/I: CHAR(8)

EIBEOC
indicates that an end-of-chain indicator appears in the RU just received (X'FF').

ASM: CL1
COBOL: PIC X(1)
PL/I: CHAR(1)

EIBFMH
indicates that the user data just received or retrieved contains an FMH (X'FF').

ASM: CL1
COBOL: PIC X(1)
PL/I: CHAR(1)

EIBFN
contains a code that identifies the last CICS/VS command to be issued by the task (updated when the requested function has been completed). Details of the codes used in this field are shown in the first table later in the appendix.

ASM: CL2
COBOL: PIC X(2)
PL/I: CHAR(2)

EIBFREE
indicates that the application program cannot continue using the facility. The application program should either free the facility or should terminate so that the facility is freed by CICS/VS (X'FF').

ASM: CL1
COBOL: PIC X(1)
PL/I: CHAR(1)

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 the HANDLE CONDITION command.

Details of the codes used in this field are shown in the second table later in the appendix.

ASM: CL6
COBOL: PIC X(6)
PL/I: CHAR(6)

EIBRECV
indicates that the application program is to continue receiving data from the facility by executing RECEIVE commands (X'FF').

ASM: CL1
COBOL: PIC X(1)
PL/I: CHAR(1)

EIBREQID
contains the request identifier assigned to an interval control command by CICS/VS; this field is not used when a request identifier is specified in the application program.

ASM: CL8
COBOL: PIC X(8)
PL/I: CHAR(8)

EIBRSRCE
contains the symbolic identifier of the resource being accessed by the latest executed command. For file control commands this will be the name of the data set. For transient data and temporary storage commands it will be the name of the queue. For terminal control commands it will be the name of the terminal or logical unit. Identifiers less than eight characters in length are padded on the right with blanks.

ASM: CL8
COBOL: PIC X(8)
PL/I: CHAR(8)

EIBSYNC
indicates that the application program must take a sync point or terminate. Before either is done, the application program must ensure that any other facilities, owned by it, are put into the send state, or are freed (X'FF').

ASM: CL1
COBOL: PIC X(1)
PL/I: CHAR(1)

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.

ASM: PL4
COBOL: PIC S9(7) COMP-3
PL/I: FIXED DEC(7,0)

EIBTIME
contains the time at which the task is started (this field is updated by the ASKTIME command). The time is in packed decimal form (0HHMMSS+).

ASM: PL4
COBOL: PIC S9(7) COMP-3
PL/I: FIXED DEC(7,0)

EIBTRMID
contains the symbolic terminal identifier of the principal facility (terminal or logical unit) associated with the task.

ASM: CL4
COBOL: PIC X(4)
PL/I: CHAR(4)

EIBTRNID
contains the symbolic transaction identifier of the task.

ASM: CL4
COBOL: PIC X(4)
PL/I: CHAR(4)

Code	Command	Code	Command
02 02	ADDRESS	0C 02	GETMAIN
02 04	HANDLE CONDITION	0C 04	FREEMAIN
02 06	HANDLE AID	0E 02	LINK
02 08	ASSIGN	0E 04	XCTL
02 0A	IGNORE CONDITION	0E 06	LOAD
04 02	RECEIVE	0E 08	RETURN
04 04	SEND	0E 0A	RELEASE
04 06	CONVERSE	0E 0C	ABEND
04 08	ISSUE EODS	0E 0E	HANDLE ABEND
04 0A	ISSUE COPY	10 02	ASKTIME
04 0C	WAIT TERMINAL	10 04	DELAY
04 0E	ISSUE LOAD	10 06	POST
04 10	WAIT SIGNAL	10 08	START
04 12	ISSUE RESET	10 0A	RETRIEVE
04 14	ISSUE DISCONNECT	10 0C	CANCEL
04 16	ISSUE ENDOUTPUT	12 02	WAIT EVENT
04 18	ISSUE ERASEAUP	12 04	ENQ
04 1A	ISSUE ENDFILE	12 06	DEQ
04 1C	ISSUE PRINT	12 08	SUSPEND
04 1E	ISSUE SIGNAL	14 02	JOURNAL
04 20	ALLOCATE	14 04	WAIT JOURNAL
04 22	FREE	16 02	SYNCPOINT
04 24	POINT	18 02	RECEIVE MAP
04 26	BUILD ATTACH	18 04	SEND MAP
04 28	EXTRACT ATTACH	18 06	SEND TEXT
04 2A	EXTRACT TCT	18 08	SEND PAGE
06 02	READ	18 0A	PURGE MESSAGE
06 04	WRITE	18 0C	ROUTE
06 06	REWRITE	1A 02	TRACE ON/OFF
06 08	DELETE	1A 04	ENTER
06 0A	UNLOCK	1C 02	DUMP
06 0C	STARTBR	1E 02	ISSUE ADD
06 0E	READNEXT	1E 04	ISSUE ERASE
06 10	READPREV	1E 06	ISSUE REPLACE
06 12	ENDBR	1E 08	ISSUE ABORT
06 14	RESETBR	1E 0A	ISSUE QUERY
08 02	WRITEQ TD	1E 0C	ISSUE END
08 04	READQ TD	1E 0E	ISSUE RECEIVE
08 06	DELETEQ TD	1E 10	ISSUE NOTE
0A 02	WRITEQ TS	1E 12	ISSUE WAIT
0A 04	READQ TS	1E 14	ISSUE SEND
0A 06	DELETEQ TS	20 02	BIF DEEDIT

EIBFN Codes

EIBFN Byte 0	EIBRCODE Byte	EIBRCODE Bit(s)	Meaning	EIBFN Byte 0	EIBRCODE Byte	EIBRCODE Bit(s)	Meaning
02	0	E0	INVREQ	0A	0	D0	SYSIDERR
04	0	04	EOF	0A	0	D1	ISCINVREQ
04	0	10	EODS	0A	0	E1	LENGERR
04	0	C1	EOF	0C	0	E2	NOSTG
04	0	C2	ENDINPT	0E	0	01	PGMIDERR
04	0	E1	LENGERR	0E	0	E0	INVREQ
04	0	E3	WRBRK	10	0	01	ENDDATA
04	0	E4	RDATT	10	0	04	IOERR
04	0	E5	SIGNAL	10	0	11	TRANSIDERR
04	0	E6	TERMIDERR	10	0	12	TERMIDERR
04	0	E7	NOPASSBKRD	10	0	14	INVTSREQ
04	0	E8	NOPASSBKWR	10	0	20	EXPIRED
04	0	EA	IGREQCD	10	0	81	NOTFND
04	0	EB	CBIDERR	10	0	D0	SYSIDERR
04	0	D0	SYSIDERR[3]	10	0	D1	ISCINVREQ
04	0	D2	SESSIONERR[3]	10	0	E1	LENGERR
04	0	D3	SYSBUSY	10	0	E9	ENVDEFERR
04	0	D4	SESSBUSY	10	0	FF	INVREQ
04	0	D5	NOTALLOC	12	0	32	ENQBUSY
04	1	20	EOC	14	0	01	JIDERR
04	1	40	INBFMH	14	0	02	INVREQ
04	3	F6	NOSTART	14	0	05	NOTOPEN
04	3	F7	NONVAL	14	0	06	LENGERR
06	0	01	DSIDERR	14	0	07	IOERR
06	0	02	ILLOGIC[1]	14	0	09	NOJBUFSP
06	0	04	SEGIDERR	18	0	01	INVREQ
06	0	08	INVREQ	18	0	02	RETPAGE
06	0	0C	NOTOPEN	18	0	04	MAPFAIL
06	0	0F	ENDFILE	18	0	08	INVMPSZ[2]
06	0	80	IOERR[1]	18	0	20	INVERRTERM
06	0	81	NOTFND	18	0	40	RTESOME
06	0	82	DUPREC	18	0	80	RTEFAIL
06	0	83	NOSPACE	18	0	E3	WRBRK
06	0	84	DUPKEY	18	0	E4	RDATT
06	0	D0	SYSIDERR[3]	18	1	10	INVLDC
06	0	D1	ISCINVREQ	18	1	40	IGREQCD
06	0	E1	LENGERR	18	1	80	TSIOERR
08	0	01	QZERO	18	2	01	OVERFLOW
08	0	02	QIDERR	18	2	04	EODS
08	0	04	IOERR	18	2	08	EOC
08	0	08	NOTOPEN	18	2	10	IGREQID
08	0	10	NOSPACE	1E	0	04	DSSTAT
08	0	C0	QBUSY	1E	0	08	FUNCERR
08	0	D0	SYSIDERR[3]	1E	0	0C	SELNERR
08	0	D1	ISCINVREQ	1E	0	10	UNEXPIN
08	0	E1	LENGERR	1E	0	E1	LENGERR
0A	0	01	ITEMERR	1E	1	11	EODS
0A	0	02	QIDERR	1E	1	15	NODATARECD
0A	0	04	IOERR	1E	1	2B	IGREQCD
0A	0	08	NOSPACE	1E	2	20	EOC
0A	0	20	INVREQ				

EIBRCODE Codes

Notes:

1. When this condition occurs during File Control operations, further information is provided in field EIBRCODE, as follows:

   bytes 1-4 = DAM response (OS/VS only)
   bytes 1-2 = ISAM response
   byte 1 = VSAM return code
   byte 2 = VSAM error code

2. When this condition occurs during BMS operations, byte 3 of field EIBRCODE contains the terminal code. See "Terminal Code Table" in "Chapter 3.3. Basic Mapping Support (BMS)."

3. When this condition occurs, further information is provided in byte 1 of EIBRCODE, as follows:

   04 Name not that of system entry.
   08 Link out of service.
   0C Name unknown to CICS/VS.

Appendix B. Translation Tables for the 2980

This appendix contains translation tables for the following components of the IBM 2980 General Banking Terminal System:

- 2980 Teller Station Model 1 (Figure 22 on page 244)
- 2980 Administrative Station Model 2 (Figure 23 on page 245)
- 2980 Teller Station Model 4 (Figure 24 on page 246)

The line codes and CPU codes listed in these tables are unique to CICS/VS and are represented as standard EBCDIC characters.

KEY No.	ENGRAVING Top(LC)	Front(UC)	LINE Code	CPU CODE Numeric(LC)	Alpha(UC)	HLL ID
0	MSG ACK	1	F1	AA	F1	1
1	SEND AGAIN	Q	D8	D9	D8	
2	CORR	A	C1	C3	C1	
3	HOLD OVRDE	2	F2	C8	F2	
4	VOID	Z	E9	E5	E9	
5	ACCT INQ	W	E6	D8	E6	
6	ACCT TFR	S	E2	AB	E2	2
7	CIF	3	F3	AC	F3	3
8	MISC	X	E7	AD	E7	4
9	CLSD ACCT	E	C5	E7	C5	
10	NO BOOK	D	C4	AE	C4	5
11	MORT LOAN	4	F4	AF	F4	6
12		C	C3	B0	C3	7
13	NEW ACCT	R	D9	B1	D9	8
14	BOOK BAL	F	C6	B2	C6	9
15	INST LOAN	5	F5	B3	F5	10
16	SPEC TRAN	V	E5	B4	E5	11
17	SAV BOND	T	E3	B5	E3	12
18	SAV	G	C7	B6	C7	13
19	XMAS CLUB	6	F6	B7	F6	14
20	•	B	C2	4B	C2	
21	DDA	Y	E8	B8	E8	15
22	00	H	C8	B9	C8	16
23	MON ORD	7	F7	BA	F7	17
24	0	N	D5	F0	D5	
25	7	U	E4	F7	E4	
26	4	J	D1	F4	D1	
27	CSHR CHK	8	F8	BB	F8	18
28	1	M	D4	F1	D4	
29	8	I	C9	F8	C9	
30	5	K	D2	F5	D2	
31	CASH RECD	9	F9	BC	F9	19
32	2	'	6B	F2	6B	
33	9	0	D6	F9	D6	
34	6	L	D3	F6	D3	
35	UTIL BILL	0	F0	E4	F0	
36	3	•	4B	F3	4B	
37	DEP +	P	D7	4E	D7	
38	WITH −	$	5B	60	5B	
39	FEES	−	60	C6	60	
40	TOTL	/	61	E3	61	
41	CASH IN	*	5C	BD	5C	20
42	CASH CHK	#	7B	BE	7B	21
43	VAL	&	50	STATION ID	50	
44	TAB		05	05	05	TABCHAR
45	ALPHA ENTRY		36			
46	NUM ENTRY		06			
47	SEND		26-ETB 03-ETX			
48	RETURN		15	15	15	JRNLCR
49	NUM ENTRY		06			
50	SPACE		40	40	40	
58	MSGLIGHT		17	17	17	MSGLITE

Figure 22. 2980-1 Character Set/Translate Table

KEY No.	ENGRAVING Top(LC)		LINE Code	CPU CODE Numeric(LC)	Alpha(UC)	HLL ID
0	= 1		F1	F1 (1)	7E (=)	
1	Q		D8	98 (q)	D8 (Q)	
2	A		C1	81 (a)	C1 (A)	
3	2		F2	F2 (2)	4C (<)	
4	Z		E9	A9 (z)	E9 (Z)	
5	W		E6	A6 (w)	E6 (W)	
6	S		E2	A2 (s)	E2 (S)	
7	; 3		F3	F3 (3)	5E (;)	
8	X		E7	A7 (x)	E7 (X)	
9	E		C5	85 (e)	C5 (E)	
10	D		C4	84 (d)	C4 (D)	
11	: 4		F4	F4 (4)	7A (:)	
12	C		C3	83 (c)	C3 (C)	
13	R		D9	99 (r)	D9 (R)	
14	F		C6	86 (f)	C6 (F)	
15	% 5		F5	F5 (5)	6C (%)	
16	V		E5	A5 (v)	E5 (V)	
17	T		E3	A3 (t)	E3 (T)	
18	G		C7	87 (g)	C7 (G)	
19	' 6		F6	F6 (6)	7D (')	
20	B		C2	82 (b)	C2 (B)	
21	Y		E8	A8 (y)	E8 (Y)	
22	H		C8	88 (h)	C8 (H)	
23	> 7		F7	F7 (7)	6E (>)	
24	N		D5	95 (n)	D5 (N)	
25	U		E4	A4 (u)	E4 (U)	
26	J		D1	91 (j)	D1 (J)	
27	* 8		F8	F8 (8)	5C (*)	
28	M		D4	94 (m)	D4 (M)	
29	I		C9	89 (i)	C9 (I)	
30	K		D2	92 (k)	D2 (K)	
31	(9		F9	F9 (9)	4D (()	
32	| ,		6B	6B (,)	4F (|)	
33	O		D6	96 (o)	D6 (O)	
34	L		D3	93 (l)	D3 (L)	
35	) 0		F0	F0 (0)	5D ())	
36	¬ .		4B	4B (.)	5F (¬)	
37	P		D7	97 (p)	D8 (P)	
38	! $		5B	5B ($)	5A (!)	
39	_ —		60	60 (—)	6D (_)	
40	? /		61	61 (/)	6F (?)	
41	c à		5C	70 (à)	4A (c)	
42	" #		7B	7B (#)	7F (")	
43	+ &		50	50 (&)	4E (+)	
44	TAB		05	05	05	
45	LOCK		36	36	36	
46	SHIFT		06	06	06	
47	BACKSPACE		16	10	16	BCKSPACE
48	RETURN		15	15	15	
49	SHIFT		06	06	06	
50	(SPACE)		40	40	40	
53	SEND		26-ETB 03-ETX			

Figure 23. 2980-2 Character Set/Translate Table

KEY No.	ENGRAVING Top(LC)	Front(UC)	LINE Code	CPU CODE Numeric(LC)	Alpha(UC)	HLL ID
0	CK $	—	D9	BC	60	19
1		Q	D3	D3	D8	
2		A	C1	C1	C1	
3	CK #	0	C9	B7	C9	14
4		Z	E9	4B	E9	
5		W	E6	5C	E6	
6		S	E2	5B	E2	
7	IMD 2	1	5B	4F	F1	
8		X	E7	AE	E7	5
9		E	C5	C5	C5	
10		D	C4	6F	C4	
11	IMD 1	2	4B	BF	F2	
12		C	C3	C3	C3	
13		R	60	60	D9	
14		F	C6	C6	C6	
15	CODE	3	E8	BB	F3	
16		V	E5	A0	E5	22
17		T	E3	A1	E3	23
18		G	C7	C7	C7	
19	AMT	4	5C	BE	F4	21
20		B	C2	C2	C2	
21		Y	61	61	E8	
22		H	D7	D7	C8	
23	0B	5	D8	B2	F5	9
24		N	D5	D5	D5	
25		U	E4	AF	E4	6
26		J	D1	D1	D1	
27	ACCT #	6	C8	7B	F6	
28		N	D4	E7	D4	
29		I	D6	D6	C9	
30		K	D2	D2	D2	
31	7	7	F7	F7	F7	
32	...	...	6B	BLANK	6B	
33	4	0	F4	F4	D6	
34	1	L	F1	F1	D3	
35	8	8	F8	F8	F8	
36	0	.	F0	F0	4B	
37	5	P	F5	F5	D7	
38	2	$	F2	F2	5B	
39	9	9	F9	F9	F9	
40	...	...	7B	B0	7B	7
41	6	*	F6	F6	5C	
42	3	#	F3	F3	7B	
43	VAL	&	50	50	50	
44	TAB		05	05	05	
45	ALPHA		36			
46	NUMERIC		06			
47	SEND		26-ETB 03-ETX			
48	RETURN		15	15	15	
49	NUMERIC		06			
50	SPACE		40	40	40	
51	FEED OPEN		04			OPENCH

Figure 24. 2980-4 Character Set/Translate Table

Appendix C. CICS/VS Macros and Equivalent Commands

This appendix provides a list of the macro instructions available to the CICS/VS application programmer, and shows for each macro instruction the command that will perform the same function. Command options may have different defaults and/or functions from macro-level operands having similar names. Some CICS/VS macros do not have an equivalent command; for example, there is only one CICS/VS built-in function that can be invoked by a command.

Although the TYPE=CHECK macro performs a similar function to the HANDLE CONDITION command, it is used in a completely different way.

Macro	Command
DFHBFTA	-
DFHBIF	
TYPE=DEEDIT	BIF DEEDIT
DFHBMS	
TYPE=CHECK	HANDLE CONDITION
TYPE=IN	RECEIVE MAP
TYPE=MAP	RECEIVE MAP FROM
TYPE=OUT	SEND TEXT
TYPE=OUT,MAP=	SEND MAP
TYPE=PAGEBLD	SEND MAP ACCUM
TYPE=PAGEOUT	SEND PAGE
TYPE=PURGE	PURGE MESSAGE
TYPE=RETURN	SEND {MAP|TEXT} SET
TYPE=ROUTE	ROUTE
TYPE=STORE	SEND {MAP|TEXT} PAGING
TYPE=TEXTBLD	SEND TEXT ACCUM
DFHDC	
TYPE=CICS	DUMP TABLES
TYPE=COMPLETE	DUMP COMPLETE
TYPE=PARTIAL LIST=PROGRAM	DUMP PROGRAM
LIST=TERMINAL	DUMP TERMINAL
LIST=TRANSACTION	DUMP STORAGE
LIST=SEGMENT	DUMP FROM
TYPE=TRANSACTION	DUMP [TASK]
DFHDI	
TYPE=ABORT	ISSUE ABORT
TYPE=ADD	ISSUE ADD
TYPE=CHECK	HANDLE CONDITION
TYPE=END	ISSUE END
TYPE=ERASE	ISSUE ERASE
TYPE=NOTE	ISSUE NOTE
TYPE=QUERY	ISSUE QUERY
TYPE=RECEIVE	ISSUE RECEIVE
TYPE=REPLACE	ISSUE REPLACE
TYPE=SEND	ISSUE SEND
TYPE=WAIT	ISSUE WAIT
DFHFC	
TYPE=CHECK	HANDLE CONDITION
TYPE=DELETE	DELETE RIDFLD
(DL/I types)	-
TYPE=ESETL	ENDBR
TYPE=GET	READ
TYPE=GET, TYPOPER=UPDATE	READ UPDATE
TYPE=GETAREA	-
TYPE=GETNEXT	READNEXT
TYPE=GETPREV	READPREV
TYPE=PUT, TYPOPER=DELETE	DELETE
TYPE=PUT, TYPOPER=NEWREC	WRITE
TYPE=PUT, TYPOPER=UPDATE	REWRITE
TYPE=RELEASE	UNLOCK
TYPE=RESETL	RESETBR
TYPE=SETL	STARTBR
DFHIC	
TYPE=CANCEL	CANCEL

TYPE=CHECK	HANDLE CONDITION
TYPE=GET	RETRIEVE
TYPE=GETIME	ASKTIME
TYPE=INITIATE	START
TYPE=POST	POST
TYPE=PUT	START FROM
TYPE=RETRY	RETRIEVE
TYPE=WAIT	DELAY

DFHJC

TYPE=CHECK	HANDLE CONDITION
TYPE=GETJCA	-
TYPE=PUT	JOURNAL WAIT
TYPE=WAIT	WAIT JOURNAL
TYPE=WRITE	JOURNAL

DFHKC

TYPE=ATTACH	-
TYPE=CHAP	-
TYPE=DEQ	DEQ
TYPE=ENQ	ENQ
TYPE=NOPURGE	-
TYPE=PURGE	-
TYPE=WAIT	SUSPEND
TYPE=WAIT, ECADDR	WAIT EVENT

DFHMDF	-
DFHMDI	-
DFHMSD	-

DFHPC

TYPE=ABEND	ABEND
TYPE=CHECK	HANDLE CONDITION
TYPE=COBADDR	-
TYPE=DELETE	RELEASE
TYPE=LINK	LINK
TYPE=LOAD	LOAD
TYPE=RESETXIT	HANDLE ABEND RESET
TYPE=RETURN	RETURN
TYPE=SETXIT	HANDLE ABEND
TYPE=XCTL	XCTL

DFHSC

TYPE=FREEMAIN	FREEMAIN
TYPE=GETMAIN	GETMAIN

DFHSP

TYPE=USER	SYNCPOINT
TYPE=ROLLBACK	SYNCPOINT ROLLBACK

DFHTC

TYPE=CBUFF	SEND CBUFF
TYPE=CONVERSE	CONVERSE
TYPE=COPY	ISSUE COPY
TYPE=DISCONNECT	ISSUE DISCONNECT
TYPE=EODS	ISSUE EODS
TYPE=ERASEAUP	ISSUE ERASEAUP
TYPE=GET	RECEIVE
TYPE=PAGE	-
TYPE=PASSBK	SEND PASSBK
TYPE=PRINT	ISSUE PRINT
TYPE=PROGRAM	ISSUE LOAD
TYPE=PUT	SEND WAIT
TYPE=READ	RECEIVE (WAIT assumed)
TYPE=READB	RECEIVE BUFFER
TYPE=READL	RECEIVE LEAVEKB
TYPE=RESET	ISSUE RESET
TYPE=SIGNAL	WAIT SIGNAL
TYPE=WAIT	WAIT TERMINAL
TYPE=WRITE	SEND
TYPE=WRITEL	SEND LEAVEKB

DFHTD

TYPE=CHECK	HANDLE CONDITION
TYPE=FEOV	-
TYPE=GET	READQ TD
TYPE=PURGE	DELETEQ TD
TYPE=PUT	WRITEQ TD

DFHTR

TYPE=ENTRY	ENTER

TYPE=OFF	TRACE OFF	TYPE=PUT[1]	WRITEQ TS
TYPE=ON	TRACE ON	TYPE=PUTQ	WRITEQ TS
		TYPE=RELEASE	DELETEQ TS

DFHTS

TYPE=CHECK	HANDLE CONDITION	[1] Because single units of information cannot be handled by the command-level interface, data stored by a DFHTS TYPE=PUT macro cannot be retrieved by a READQ TS command. Conversely, data stored by a WRITEQ TS command cannot be retrieved by a DFHTS TYPE=GET macro.
TYPE=GET[1]	READQ TS	
TYPE=GETQ	READQ TS	
TYPE=PURGE	DELETEQ TS	

Appendix D. Sample Programs (Assembler Language)

This appendix consists of sample CICS/VS application programs written in the assembler language. The BMS maps and file record descriptions used by the sample programs are included after the sample programs. The maps are unaligned. Users of aligned maps should protect the alignment of their map DSECTS.

The sample maps include examples of how the COLOR, EXTATT, and HILIGHT attributes are specified in the map definition macros. However, due to production limitations, the associated screen layouts do not show the effects of these attributes; they show how the maps would be displayed on, for example, a 3277.

Specifying EXTATT=MAPONLY enables attributes to be added without changing the application program. Any attribute, that specifies a facility not available at the terminal, will be ignored.

The sample programs illustrate basic applications that can serve as a framework for the installation's first programs. Each program has a description and program notes. The program listings are of source code. Numbered coding lines correspond to the numbered program notes.

All transactions are initiated by the terminal operator entering a four-character transaction code. (An account number must also be entered, except in the case of the operator instruction sample program.)

There are six sample programs, as follows:

- Operator Instruction Sample Program
- Update Sample Program
- Browse Sample Program
- Order Entry Sample Program
- Order Entry Queue Print Sample Program
- Report Sample Program

All the sample programs operate on a sample VSAM or ISAM file which must first be created using a program provided on the library. The file consists of records containing details of individual accounts. The programs are used to display, alter, update, or browse through the entries. For information on how to create the sample VSAM or ISAM file refer to the CICS/VS System Programmer's Guide.

All the sample programs are for use with the IBM 3270 Information Display System.

EXECUTING THE SAMPLE PROGRAMS

Once CICS/VS is running, 3270 users can enter the following transaction id's:

AMNU	Display other transaction id's (except AORD, ACOM, AREP)
AINQ	Display an entry.
AADD	Create a new entry.
AUPD	Update an entry.
ABRW	Browse through entries.
AORD	Order entry.
ACOM	Print order entry queue.
AREP	Display a report (entries not greater than $50).

Note: The transaction ACOM should be used once in the morning, after which it will invoke itself at the printer in one hour (unless the time is 1400 hrs or after).

OPERATOR INSTRUCTION SAMPLE PROGRAM (ASSEMBLER LANGUAGE)

DESCRIPTION

To begin 3270 operations, a terminal operator must enter a transaction code of AMNU. Whenever the screen is cleared this transaction code must be reentered, as no data is accepted from an unformatted screen.

The instruction program displays map XDFHAMA containing operator instructions. This map lists the ASSEMBLER LANGUAGE CICS/VS sample applications and the transaction codes (with the exception of AORD and ACOM which are entered onto a clear screen), and provides space for entering the code and an account number.

SOURCE LISTING

```
DFHEISTG DSECT
XDFHAMNU CSECT
   1     EXEC CICS SEND MAP('XDFHAMA') MAPONLY ERASE
   2     EXEC CICS RETURN
         END
```

PROGRAM NOTES

1. The BMS command erases the screen and displays map XDFHAMA.
2. RETURN ends the program.

UPDATE SAMPLE PROGRAM (ASSEMBLER LANGUAGE)

DESCRIPTION

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-id, 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, then 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. Fields are 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.

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

SOURCE LISTING

```
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 2ND ADD ROUTINE
  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 COMMAREA
          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 IN IN FILE AREA
ADRTST   OC    ADDRI,ADDRI           HAS ADRESS BEEN ENTERED?
         BZ    PHNTST                ..NO, BRANCH
         MVC   ADDRX,ADDRI           ..YES, PUT IN IN FILE AREA
PHNTST   OC    PHONEI,PHONEI         HAS PHONE BEEN ENTERED?
         BZ    DATTST                ..NO, BRANCH
         MVC   PHONE,PHONEI          ..YES, PUT IN IN FILE AREA
DATTST   OC    DATEI,DATEI           HAS DATE BEEN ENTERED?
         BZ    AMTTST                ..NO, BRANCH
         MVC   DATEX,DATEI           ..YES, PUT IN IN FILE AREA
AMTTST   OC    AMOUNTI,AMOUNTI       HAS AMOUNT BEEN ENTERED?
         BZ    COMTST                ..NO, BRANCH
         MVC   AMOUNT,AMOUNTI        ..YES, PUT IN IN FILE AREA
COMTST   OC    COMMENTI,COMMENTI HAS COMMENT BEEN ENTERED?
         BZ    CONTINUE              ..NO, CONTINUE
         MVC   COMMENT,COMMENTI      ..YES, PUT IN 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-id's 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 is invoked by the transaction-id '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. If no data is entered, the field contains the original data if it has not been modified when the contents of map XDFHAMB are 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 is invoked by the transaction-id 'AINQ', a title and command message are moved to the map area.
13. This invocation of the program ends.
14. If the program is invoked by the transaction-id '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/VS 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 map XDFHAMB is sent to the terminal.

30. Any required editing steps should be inserted here. A suitable form of editing should be used to ensure 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 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.

BROWSE SAMPLE PROGRAM (ASSEMBLER LANGUAGE)

DESCRIPTION

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 F causes retrieval of the next page or paging forward. If the operator wishes to reexamine 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 setup. 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.

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 pressing 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 stored in FLDA, to be ready for a page forward.

```
| 18  | 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 (18) to retrieve records 20 and 25. These are displayed after the keys are shifted right. The last record read (25) is stored in FLDA.

```
| 25  | 18  | 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 previous 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 stored in FLDA for the next page forward.

```
| 18  | 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.

SOURCE LISTING

```
DFHEISTG DSECT
MESSAGES DS    CL80
KEYS     DS    0CL24
FLDA     DS    CL6
FLDB     DS    CL6
FLDC     DS    CL6
FLDD     DS    CL6
HEXZERO  DS    X'00'                CONSTANT FOR CLEARING MAPS
         COPY  XDFHAMA              MENU MAP
         COPY  XDFHAMC              DISPLAY MAP
         COPY  FILEA                FILE RECORD DESCRIPTION
         COPY  DFHBMSCA             BMS ATTRIBUTE BYTES
XDFHABRW CSECT
         MVI   KEYS,X'F0'           INSERT '0' INTO TOP BYTE OF KEYS
         MVC   KEYS+1(L'KEYS-1),KEYS  INITIALISE KEYS TO ZERO
         MVI   MESSAGES,X'40'       INSERT ' ' INTO TOP BYTE OF MESSAGES
         MVI   MESSAGES+1(L'MESSAGES-1),MESSAGES  CLEAR MESSAGES FIELD
   1     EXEC  CICS HANDLE CONDITION ERROR(ERRORS)MAPFAIL(AMNU)        *
                    ENDFILE(ENDFILE) NOTFND(NOTFOUND)
   2     EXEC CICS RECEIVE MAP('XDFHAMA')
   3     EXEC CICS HANDLE AID CLEAR(AMNU)                             *
               PF1 (PAGEF) PF2 (PAGEB)
   4     MVC   FLDA,KEYI
   5     EXEC CICS STARTBR DATASET('FILEA') RIDFLD(FLDA)
PAGEF    DS    0H
         MVC   FLDD,FLDC
   6     MVC   FLDC,FLDB
         MVC   FLDB,FLDA
BUILD    DS    0H
         LA    4,1                  SET COUNTER TO 1
         LA    6,XDFHAMCO           R6->START OF MAP XDFHAMC
         LA    7,(XDFHAMCE-XDFHAMCO) R7 CONTAINS LENGTH OF XDFHAMC
         LA    8,HEXZERO            R8-> X'00'
         LA    9,L'HEXZERO          R9 CONTAINS LENGTH OF HEXZERO
         ICM   9,B'100',HEXZERO     X'00' INTO TOP BYTE OF R9
         MVCL  6,8                  MOVE X'00' INTO XDFHAMCO
NEXTLIN  DS    0H
   7     EXEC CICS READNEXT INTO(FILEA) DATASET('FILEA') RIDFLD(FLDA)
   8     CLI   STAT,X'FF'           IS RECORD CODED AS NOT FOUND
         BE    NEXTLIN              ..YES, SET UP NEXT LINE
         CH    4,=H'1'              FIRST LINE?
         BNE   SECLIN               ..NO, GO TEST FOR 2ND LINE
   9     MVC   NUMBER1O,NUMB        MOVE NUMBER TO MAP AREA
         MVC   NAME1O,NAME          MOVE NAME TO MAP AREA
         MVC   AMOUNT1O,AMOUNT      MOVE AMOUNT TO MAP AREA
         B     CONT                 GO CONTINUE
SECLIN   CH    4,=H'2'              SECOND LINE?
  10     BNE   THRLIN               ..NO, GO TEST FOR THIRD LINE
         MVC   NUMBER2O,NUMB        MOVE NUMBER TO MAP AREA
         MVC   NAME2O,NAME          MOVE NAME TO MAP AREA
         MVC   AMOUNT2O,AMOUNT      MOVE AMOUNT TO MAP AREA
         B     CONT                 GO CONTINUE
THRLIN   CH    4,=H'3'              THIRD LINE?
         BNE   FORLIN               ..NO, GP TEST FOR FOURTH LINE
         MVC   NUMBER3O,NUMB        MOVE NUMBER TO MAP AREA
         MVC   NAME3O,NAME          MOVE NAME TO MAP AREA
         MVC   AMOUNT3O,AMOUNT      MOVE AMOUNT TO MAP AREA
         B     CONT                 GO CONTINUE
FORLIN   CH    4,=H'4'              FOURTH LINE?
         BNE   CONT                 ..NO, CONTINUE
         MVC   NUMBER4O,NUMB        MOVE NUMBER TO MAP AREA
```

```
         MVC   NAME40,NAME          MOVE NAME TO MAP AREA
         MVC   AMOUNT40,AMOUNT      MOVE AMOUNT TO MAP AREA
CONT     DS    0H
         LA    4,1(,4)              INCREMENT COUNT
         CH    4,=H'5'              FINISHED?
         BNE   NEXTLIN              ..NO, GO BUILD NEXT LINE
DISPREC  DS    0H                   ..YES, SEND MAP
  11     EXEC CICS SEND MAP('XDFHAMC') ERASE
REPEAT   DS    0H
  12     EXEC CICS RECEIVE MAP('XDFHAMC')
         CLI   DIRI,C'F'            PAGE FORWARD REQUIRED?
         BE    PAGEF                ..YES, GO TO PAGE FORWARD ROUTINE
         CLI   DIRI,C'B'            PAGE BACK REQUIRED?
         BE    PAGEB                ..YES, GO TO PAGE BACKWARD ROUTINE
         BNE   AMNU                 ..NO, GO SEND MENU MAP
ENDFILE  DS    0H                   ENDFILE IS REACHED
         MVC   MSG10,=CL(L'MSG10)'END OF FILE' MESSAGE
  13     MVI   MSG2A,DFHBMBRY       ATTRIBUTE BYTE FOR INSTRUCTION FLD
         B     DISPREC              GO SEND MAP
PAGEB    DS    0H                   PAGE FORWARD ROUTINE
  14     CLC   FLDC(6),=C'000000'   FLDC = ZEROS?
         BE    TOOFAR               ..YES, SO RAISE TOO FAR CONDITION
         MVC   FLDA,FLDC            ..NO, SET UP KEYS FOR FILE
  15     MVC   FLDB,FLDC
         MVC   FLDC,FLDD
         MVC   FLDD,=C'000000'
         EXEC CICS RESETBR DATASET('FILEA') RIDFLD(FLDA)
         B     BUILD                GO BUILD MAP
TOOFAR   DS    0H                   GONE TOO FAR
  16     MVI   MSG1A,DFHBMBRY       BRIGHTEN MESSAGE
         MVI   MSG2A,DFHBMDAR       DARKEN MESSAGE
  17     EXEC CICS SEND MAP('XDFHAMC') DATAONLY
         B     REPEAT               GO GET MAP
NOTFOUND DS    0H
  18     MVC   MESSAGES,=CL(L'MESSAGES)'INVALID NUMBER-PLEASE REENTER'
         B     AMNU
ERRORS   DS    0H                   GENERAL ERROR ROUTINE
  19     EXEC CICS DUMP DUMPCODE('ERRS')
         MVC   MESSAGES,=CL(L'MESSAGES)'TRANSACTION TERMINATED'
AMNU     DS    0H                   END ROUTINE
  20     XC    XDFHAMA0(XDFHAMAE-XDFHAMA0),XDFHAMA0 CLEAR MAP A
         MVI   MSGA,DFHBMBRY        BRIGHTEN MESSAGE FIELD
         MVC   MSG0,MESSAGES        MOVE MESSAGES TO MAP AREA
  21     EXEC CICS SEND MAP('XDFHAMA') ERASE
         EXEC CICS RETURN
         END
```

PROGRAM NOTES

1. The error 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 pressed or B typed in, indicating page backward, and FLDC contains zeros, further backward paging is not possible. The program branches to TOO-FAR (see note 17).

15. If not, the key fields are shifted left to retrieve the previous page and the starting point for the browse is 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. On the record NOTFND condition, the message 'INVALID NUMBER - PLEASE REENTER' is moved to messages.

18. An error message is written to the terminal.

19. On an error (notes 2, 5, 7, 11, 12, 17, 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 XDFHAMA is displayed, and the program ends.

ORDER ENTRY SAMPLE PROGRAM (ASSEMBLER LANGUAGE)

DESCRIPTION

The order entry sample application program accepts input relating to the ordering of parts from a warehouse. When sufficient orders have been accumulated in the headquarters of a business, these are automatically sent off to a warehouse, or some other distribution point.

The program displays the map XDFHAMK on the screen requesting the operator to input details regarding the ordering of certain parts. The screen contains entry positions relating to the customer number, the part number, and the quantity of that part required. (Any integer up to six digits in length may be entered: the customer number must be valid, that is, it must exist on FILEA.) When the screen has been filled, the operator presses ENTER. The screen is then mapped in and the data is checked, errors being returned to the operator for reentering. When all the input is correct it is sent to a transient data queue called 'L860' which is also a terminal-id where a transaction is to be triggered when the number of items on the queue reaches 30.

The trigger level may be changed using the CSMT command, as follows:

```
CSMT TRIGGER,n,DESTID=L860
```

where n is the destination trigger level (any integer from 0 through 32767).

When all orders have been entered, the operator presses CLEAR and RESET, and a new transaction may be started.

SOURCE LISTING

```
DFHEISTG DSECT
         COPY  XDFHAMK              COPY MAP
         COPY  L860                 COPY QUEUE RECORD
         COPY  FILEA                COPY FILE RECORD DESCRIPTION
         COPY  DFHBMSCA             COPY BMS ATTRIBUTE BYTES
FLAGS    DS    1B                   GROUP OF ERROR FLAGS
FLAG1    EQU   X'80'                ERROR FLAG FOR SCREEN MESSAGES
XDFHAREN CSECT
         NI    FLAGS,X'FF'-FLAG1    SET ERROR FLAG TO 0
   1     EXEC CICS HANDLE AID CLEAR(ENDA)
   2     EXEC CICS HANDLE CONDITION MAPFAIL(MAPFAIL) NOTFND(NOTFOUND)  *
               ERROR(ERRORS)
         XC    XDFHAMKO(XDFHAMKE-XDFHAMKO),XDFHAMKO   CLEAR MAP
SEND     DS    0H
   3     EXEC CICS SEND MAP('XDFHAMK') ERASE
RECEIVE  DS    0H
   4     EXEC CICS RECEIVE MAP('XDFHAMK')
QBUILD   DS    0H
   5     EXEC CICS READ DATASET('FILEA') INTO(FILEA) RIDFLD(CUSTNOI)
         MVC   CUSTNO,CUSTNOI       CREATE QUEUE RECORD
   6     MVC   PARTNO,PARTNOI
         MVC   QUANTITY,QUANTI
         MVC   TERMID,EIBTRMID      PUT TERMID ON QUEUE RECORD
QWRITE   DS    0H                   WRITE QUEUE RECORD
   7     EXEC CICS WRITEQ TD QUEUE('L860') FROM(L860) LENGTH(22)
         TM    FLAGS,FLAG1          ERROR MESSAGE ON SCREEN?
         BZ    EAUP                 ..NO BRANCH
   8     EXEC CICS SEND MAP('XDFHAMK') MAPONLY ERASE
         NI    FLAGS,X'FF'-FLAG1    CLEAR FLAG
         B     RECEIVE              GO GET NEXT RECORD
EAUP     DS    0H                   NO ERROR MESSAGE ON SCREEN
   9     EXEC CICS ISSUE ERASEAUP
         B     RECEIVE              GO GET NEXT RECORD
NOTFOUND DS    0H
  10     MVC   CUSTNOA,=C'I'        MOVE BRT AND MDT ATTRIBUTE TO NUMBER
         MVI   PARTNOA,DFHBMFSE     AND MDT TO OTHER FIELDS SO AS TO
         MVI   QUANTA,DFHBMFSE      PRESERVE CONTENTS OF SCREEN
         OI    FLAGS,FLAG1          SET ERROR FLAG
         MVI   MSG1A,DFHBMBRY       BRIGHTEN ERROR MESSAGE ON SCREEN
         B     SEND                 GO SEND MAP
MAPFAIL  DS    0H
         XC    XDFHAMKO(XDFHAMKE-XDFHAMKO),XDFHAMKO CLEAR MAP
         OI    FLAGS,FLAG1          SET ERROR FLAG
  11     MVI   MSG2A,DFHBMBRY       BRIGHTEN ERROR MESSAGE ON SCREEN
```

```
         B     SEND                GO SEND MAP
ERRORS   DS    0H
         MVI   MSG2A,DFHBMBRY
  12     MVC   MSG20,=C'TRANSACTION TERMINATED'  MESSAGE TO MAP
         EXEC CICS SEND MAP('XDFHAMK')
         EXEC CICS DUMP DUMPCODE('ERRS')
ENDA     DS    0H
  13     EXEC CICS RETURN
         END
```

PROGRAM NOTES

1. Set up exit for clear key.
2. The error exits are set up.
3. The screen is erased and the map is displayed at the terminal.
4. This command maps in the customer number, part number, and quantity. The user should add further editing steps necessary to ensure only valid orders are accepted.
5. The file control READ reads the record into a record area in order to find whether a particular record exists.
6. The input from the map is moved to the queue area.
7. The transient data WRITEQ obtains a log area, and writes this record to a sequential file.
8. If an error message is left on the screen, the screen is cleared and only the map is sent.
9. The entered fields, having been mapped in and processed, are erased, and the screen is ready to receive more input.
10. If the customer number entered was not found, the message 'NUMBER NOT FOUND - REENTER', having been stored on the screen with a dark attribute character, is brightened.
11. If no fields were entered, the message 'DATA ERROR - REENTER', also having been stored on the screen with a dark attribute character, is brightened. The customer number field is also brightened.
12. On an error a dump is taken, and the message 'TRANSACTION TERMINATED' is moved to the top message area, and the map is sent to the screen.
13. This routine ends.

ORDER ENTRY QUEUE PRINT PROGRAM (ASSEMBLER LANGUAGE)

DESCRIPTION

This transaction is invoked by entering the transaction-id 'ACOM' at the terminal. The program checks to see whether it was started from a terminal or the printer. If from a terminal, (that is, the operator is starting this transaction for the first time) the program starts the transaction at the printer in one hour. (This time interval could be changed using EDF for demonstration purposes.) The operator may then press RESET and CLEAR and enter another transaction. If from the printer, the program executes and starts again in one hour. If there are no items on the queue, a message indicating that the queue is empty, is sent to the warehouse. The last communication with the warehouse occurs not later than 1500 hours. This transaction is also started when the number of items on the queue 'L860' reaches 30.

The trigger level may be changed using the CSMT command, as follows:

```
CSMT TRIGGER,n,DESTID=L860
```

where n is the destination trigger level (any integer from 0 through 32767).

This program reads items off the queue 'L860', until the queue is empty. Should the queue have been empty initially, a message is sent to the warehouse. Using the number from the queue as a key it reads the file FILEA, and checks the amount field to see if the customer is good for credit on this order. If he is, the number, name, address, part number, and quantity are moved to the map XDFHAML and this is sent to the printer. If he is not, the time, date, queue-item, and a comment field are moved to a data area, which may be used for later processing. A message is then sent to the warehouse indicating that the queue is empty. The EIBTIME is then updated and if the time is before 1400 hours, the transaction is started in one hour.

SOURCE LISTING

```
DFHEISTG DSECT
         COPY   XDFHAML              MAP
         COPY   L860                 Q RECORD
         COPY   FILEA                FILE RECORD
LOGORD   DS     0CL92                RECORD TO BE WRITTEN ONTO LOGA
LDATE    DS     PL7
LTIME    DS     PL7
LITEM    DS     CL22
COMMNT   DS     CL11
FILLER   DS     CL51
QLENGTH  DS     1H       SIZE OF Q RECORD
XDFHACOM CSECT
         MVC    COMMNT,=C'ORDER ENTRY'
         MVI    FILLER,X'40'
         MVC    FILLER+1(L'FILLER-1),FILLER
   1     EXEC CICS HANDLE CONDITION ERROR(ERRORS)QZERO(ENDA)
   2     CLC    EIBTRMID(4),=C'L860'    TERMID='L860'?
         BNE    TIME                 IF NOT START TRANSACTION LATER
         XC     XDFHAMLO(XDFHAMLE-XDFHAMLO),XDFHAMLO   CLEAR MAP
QREAD    DS     0H
         MVC    QLENGTH,=H'+22' INITIALISATION
   3     EXEC CICS READQ TD INTO(L860)LENGTH(QLENGTH)QUEUE('L860')
   4     EXEC CICS READ DATASET('FILEA') INTO(FILEA) RIDFLD(CUSTNO)
   5     CLC    AMOUNT(8),=C'$0100.00' IS ORDER VALID?
         BL     LWRITE               IF <100  BRANCH AND WRITE LOG
         MVC    ADDRO,ADDRX          SET UP MAP
         MVC    NAMO,NAME
   6     MVC    PARTO,PARTNO
         MVC    NUMBO,CUSTNO
         MVC    LITEM,ITEM
         MVC    QUANTO,QUANTITY
   7     EXEC CICS SEND MAP('XDFHAML') ERASE PRINT L80
         B      QREAD                GET NEXT RECORD
LWRITE   DS     0H
         MVC    LDATE,EIBDATE        SET UP LOG RECORD
   8     MVC    LTIME,EIBTIME
         MVC    LITEM,ITEM
   9     EXEC CICS WRITEQ TD QUEUE('LOGA') FROM(LOGORD) LENGTH(92)
         B      QREAD                GET NEXT RECORD
```

```
ERRORS   DS    0H
  10     EXEC CICS DUMP DUMPCODE('ERRS')
         B     FIN                  BRANCH TO END
ENDA     DS    0H
  11     XC    XDFHAMLO(XDFHAMLE-XDFHAMLO),XDFHAMLO   CLEAR MAP
         MVC   TITLEO,=CL(L'TITLEO)'ORDER QUEUE IS EMPTY' SET UP TITLE
  12     EXEC CICS SEND MAP('XDFHAML') DATAONLY ERASE L80 PRINT
TIME     DS    0H
  13     EXEC CICS ASKTIME
  14     CP    EIBTIME,=P'0140000'  TIME AFTER 1400 HOURS?
         BH    FIN                  ..YES, SO STOP
  15     EXEC CICS START TRANSID('ACOM') INTERVAL(10000) TERMID('L860')
FIN      DS    0H
  16     EXEC CICS RETURN
         END
```

PROGRAM NOTES

1. The error exits are set up.
2. The terminal-id is tested to see whether this transaction was started from a terminal or at the printer.
3. The queue item is read into the program.
4. The file control READ command reads the record into a record area so that the amount may be checked.
5. The amount is tested.
6. If it is over $100, the record on the queue is moved to the map XDFHAML. This test is only a suggestion; a suitable form of editing should be inserted here to ensure valid orders are sent to the warehouse.
7. The map XDFHAML is sent to the printer.
8. If the order is not valid for this account, the record on the queue is moved to a data area, together with the terminal-id associated with the entering of this piece of data, the time, and date.
9. The transient data WRITEQ command obtains a log area, and writes this record to a sequential file.
10. On an error (notes 3, 4, 7, 9, 12, and 15) a dump is taken.
11. When the queue is empty, a message is moved to the map area.
12. The map is displayed on the screen.
13. The current time-of-day clock is updated.
14. The current time-of-day is tested.
15. If the current time is not past 1400 hours, the transaction is started again in one hour, at the warehouse printer.
16. The program ends.

REPORT SAMPLE PROGRAM (ASSEMBLER LANGUAGE)

DESCRIPTION

The report sample program produces a report that lists all entries in the data set 'FILEA' for which the amount is less than or equal to $50.00.

The program illustrates page building techniques and the use of the terminal paging facilities of BMS.

The transaction is invoked by entering the transaction code AREP. The program does a sequential scan through the file noting each entry that obeys the search criterion. The pages are built from four maps which comprise mapset XDFHAMD, using the paging option so that the data is not displayed immediately but instead is stored for later retrieval. The HEADING map is inserted at the head of each page. The detail map (XDFHAMD) is written repeatedly until the overflow condition occurs. The FOOTING map is then written at the foot of the page and the HEADING map written at the top of the next page. The command to write the detail map that caused overflow is then repeated. When all the data has been written the FINAL map is written at the bottom of the last page and the transaction terminated.

The terminal operator then enters paging commands to display the data, clearing the screen before entering each paging command.

SOURCE LISTING

```
DFHEISTG DSECT
KEYNUM   DS    CL6                 KEY TO FILE
EDVAL    DC    CL3'000'            PAGE NUMBER EDITING FIELD
RETREG   EQU   4                   LINK REG
         COPY  XDFHAMD             OUTPUT MAP
         COPY  FILEA               FILEA'S RECORD DESCRIPTION
XDFHAREP CSECT
   1     MVC   KEYNUM(6),=C'000000' SET RECORD KEY TO ZERO
   2     EXEC CICS HANDLE CONDITION ERROR(ERRORS) OVERFLOW(OFLOW)      *
               ENDFILE(ENDFILE)
   3     MVI   PAGENA,X'00'        MOVE X'00' TO ATTRIBUTE
         BAL   RETREG,MAPNUM       MOVE PAGENUMBER TO MAP AREA
   4     EXEC CICS SEND MAP('HEADING') MAPSET('XDFHAMD') ACCUM PAGING  *
               ERASE
   5     EXEC CICS STARTBR DATASET('FILEA') RIDFLD(KEYNUM)
REPEAT   DS    0H
   6     EXEC CICS READNEXT INTO(FILEA) DATASET('FILEA')               *
               RIDFLD(KEYNUM)
   7     CLI   STAT,X'FF'          RECORD CODED AS DELETED?
         BE    REPEAT              ..YES, SO GO READ NEXT RECORD
   8     CLC   AMOUNT,LOWLIM       COMPARE AMOUNT ON RECORD WITH LIM
         BH    REPEAT              ..OK, GREATER THAN $50, TRY NEXT
         XC    XDFHAMDO(XDFHAMDE-XDFHAMDO),XDFHAMDO CLEAR MAP
         MVC   AMOUNTO,AMOUNT      MOVE AMOUNT ON FILE TO MAP
   9     MVC   NUMBERO,NUMB        MOVE ACOUNT NUMBER TO MAP
         MVC   NAMEO,NAME          MOVE NAME TO MAP
         B     SENDREC             GO BUILD MAP
OFLOW    DS    0H                  PAGE BUILT HERE
  10     EXEC CICS SEND MAP('FOOTING') MAPSET('XDFHAMD')               *
               MAPONLY ACCUM PAGING ERASE
         AP    PAGEN,=P'1'         INCREMENT PAGE COUNT
         MVI   PAGENA,X'00'        MOVE X'00' INTO ATTRIBUTE
         BAL   RETREG,MAPNUM       GO SET UP PAGE NUMBER ON MAP
  11     EXEC CICS SEND MAP('HEADING') MAPSET('XDFHAMD') ACCUM PAGING  *
               ERASE
SENDREC  DS    0H
  12     EXEC CICS SEND MAP('XDFHAMD') MAPSET('XDFHAMD') ACCUM PAGING
         B     REPEAT              GO BUILD NEXT MAP
*****************************************************************
*                                                               *
*    END ROUTINE  AND GENERAL ROUTINES                          *
*                                                               *
*****************************************************************
MAPNUM   DS    0H                  ROUTINE PUTS PAGE NUM IN CHAR FORM
         UNPK  EDVAL,PAGEN
         OI    EDVAL+L'EDVAL-1,X'F0' ZERO FILL PAGE NUMBER
         MVC   PAGENO,EDVAL        MOVE PAGE NUMBER TO OUTPUT MAP
```

```
         BR    RETREG              RETURN
ENDFILE  DS    0H                  END OF FILE CONDITION RAISED
  13     EXEC CICS SEND MAP ('FINAL') MAPSET ('XDFHAMD') MAPONLY       *
               ACCUM PAGING
  14     EXEC CICS SEND PAGE
  15     EXEC CICS SEND TEXT FROM (OPINSTR) ERASE
  16     EXEC CICS ENDBR DATASET('FILEA')
  17     EXEC CICS RETURN
ERRORS   DS    0H
  18     EXEC CICS HANDLE CONDITION ERROR
  19     EXEC CICS PURGE MESSAGE
  20     EXEC CICS ABEND ABCODE('ERRS')
PAGEN    DC    PL2'1'              INITIAL PAGE NUM
LOWLIM   DC    CL8'$0050.00'       LOWER LIMIT FOR OK AMOUNT
OPINSTR  DC    CL22'ENTER PAGING COMMANDS.' OPERATOR INSTRUCTION.
         END
```

PROGRAM NOTES

1. The initial key value is set up for the START BROWSE command.
2. The program exits are set up.
3. The attribute byte for the page number is cleared.
4. This BMS request sets up the heading in the page build operation.
5. This command starts the browse through the file, at a record whose key is equal to or greater than that specified.
6. This command reads the next record on the file into the file area.
7. If the record is coded as deleted, it is treated as not found.
8. The search criterion for creating the report is that the customer has less than or equal to $50.
9. Fields are moved from the file area to the map area.
10. The BMS request sets up the footing in the page build operation.
11. The BMS request sets up the heading in the page build operation.
12. The customer detail map is set up.
13. When the END OF FILE condition is raised, the last map is built.
14. The page is sent to the terminal operator.
15. A message is sent to the terminal.
16. The BROWSE operation is ended.
17. The program ends.
18. On an error, the label to branch to on the ERROR condition is reset.
19. Any pages waiting to be displayed at the terminal are purged.
20. The program raises an abend condition, a dump is taken and the program ends.

XDFHAMA 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',  *
               HILIGHT=UNDERLINE
         DFHMDF POS=(3,1),LENGTH=29,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
```

XDFHAMA 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+
```

XDFHAMB 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
```

XDFHAMB SCREEN LAYOUT

```
          +XXXXXXXXXXXX

+NUMBER: +XXXXXX+
+NAME:   +XXXXXXXXXXXXXXXXXXX+
+ADDRESS:+XXXXXXXXXXXXXXXXXXX+
+PHONE:  +XXXXXXXX+
+DATE:   +XXXXXXXX+
+AMOUNT: +XXXXXXXX+
+COMMENT:+XXXXXXXXX+

+XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
+XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
```

XDFHAMC MAP DEFINITION

```
MAPSETC  DFHMSD TYPE=&SYSPARM,MODE=INOUT,CTRL=(FREEKB,FRSET),LANG=ASM, *
               TIOAPFX=YES,EXTATT=MAPONLY
XDFHAMC  DFHMDI SIZE=(12,40)
DIR      DFHMDF POS=(1,1),LENGTH=1,ATTRB=IC
         DFHMDF POS=(1,3),LENGTH=1
         DFHMDF POS=(1,15),LENGTH=11,INITIAL='FILE BROWSE',            *
               HILIGHT=UNDERLINE,COLOR=BLUE
         DFHMDF POS=(3,1),LENGTH=6,INITIAL='NUMBER',COLOR=BLUE
         DFHMDF POS=(3,17),LENGTH=4,INITIAL='NAME',COLOR=BLUE
         DFHMDF POS=(3,32),LENGTH=6,INITIAL='AMOUNT',COLOR=BLUE
NUMBER1  DFHMDF POS=(4,1),LENGTH=6
NAME1    DFHMDF POS=(4,9),LENGTH=20
AMOUNT1  DFHMDF POS=(4,30),LENGTH=8
NUMBER2  DFHMDF POS=(5,1),LENGTH=6
NAME2    DFHMDF POS=(5,9),LENGTH=20
AMOUNT2  DFHMDF POS=(5,30),LENGTH=8
NUMBER3  DFHMDF POS=(6,1),LENGTH=6
NAME3    DFHMDF POS=(6,9),LENGTH=20
AMOUNT3  DFHMDF POS=(6,30),LENGTH=8
NUMBER4  DFHMDF POS=(7,1),LENGTH=6
NAME4    DFHMDF POS=(7,9),LENGTH=20
AMOUNT4  DFHMDF POS=(7,30),LENGTH=8
MSG1     DFHMDF POS=(11,1),LENGTH=39,                                  *
               INITIAL='PRESS PF1 OR TYPE F TO PAGE FORWARD'
MSG2     DFHMDF POS=(12,1),LENGTH=39,                                  *
               INITIAL='PRESS PF2 OR TYPE B TO PAGE BACKWARD'
         DFHMSD TYPE=FINAL
         END
```

XDFHAMC SCREEN LAYOUT

```
                +FILE BROWSE

+NUMBER           +NAME          +AMOUNT
+XXXXXX +XXXXXXXXXXXXXXXXXXX+XXXXXX
+XXXXXX +XXXXXXXXXXXXXXXXXXX+XXXXXX
+XXXXXX +XXXXXXXXXXXXXXXXXXX+XXXXXX
+XXXXXX +XXXXXXXXXXXXXXXXXXX+XXXXXX



+PRESS PF1 OR TYPE F TO PAGE FORWARD
+PRESS PF2 OR TYPE B TO PAGE BACKWARD
+XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
+XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
```

XDFHAMD MAP DEFINITION

```
MAPSET    DFHMSD TYPE=&SYSPARM,MODE=INOUT,CTRL=(FREEKB,FRSET),          *
               LANG=ASM,STORAGE=AUTO,EXTATT=MAPONLY,COLOR=BLUE
XDFHAMD   DFHMDI SIZE=(1,40),COLOR=GREEN
NUMBER    DFHMDF POS=(1,1),LENGTH=6
NAME      DFHMDF POS=(1,9),LENGTH=20
AMOUNT    DFHMDF POS=(1,30),LENGTH=8
HEADING   DFHMDI SIZE=(3,40),HEADER=YES
          DFHMDF POS=(1,5),LENGTH=18,INITIAL='LOW BALANCE REPORT',      *
               HILIGHT=UNDERLINE
          DFHMDF POS=(1,30),LENGTH=4,INITIAL='PAGE'
PAGEN     DFHMDF POS=(1,35),LENGTH=3
          DFHMDF POS=(3,1),LENGTH=6,INITIAL='NUMBER'
          DFHMDF POS=(3,17),LENGTH=4,INITIAL='NAME'
          DFHMDF POS=(3,32),LENGTH=6,INITIAL='AMOUNT'
FOOTING   DFHMDI SIZE=(2,40),TRAILER=YES,JUSTIFY=LAST
          DFHMDF POS=(2,10),LENGTH=25,                                  *
               INITIAL='CONTINUED ON NEXT PAGE...'
FINAL     DFHMDI SIZE=(2,40),TRAILER=YES,JUSTIFY=LAST
          DFHMDF POS=(2,10),LENGTH=14,INITIAL='END OF REPORT.'
          DFHMSD TYPE=FINAL
          END
```

XDFHAMD SCREEN LAYOUT

```
    LOW BALANCE REPORT        PAGE XXX

NUMBER           NAME             AMOUNT
XXXXXX XXXXXXXXXXXXXXXXXXX   XXXXXXXX
XXXXXX XXXXXXXXXXXXXXXXXXX   XXXXXXXX
     (REPEAT TOTAL OF 19 TIMES)
XXXXXX XXXXXXXXXXXXXXXXXXX   XXXXXXXX
XXXXXX XXXXXXXXXXXXXXXXXXX   XXXXXXXX
XXXXXX XXXXXXXXXXXXXXXXXXX   XXXXXXXX

         CONTINUED ON NEXT PAGE...
```

XDFHAMK MAP DEFINITION

```
MAPSET   DFHMSD TYPE=&SYSPARM,MODE=INOUT,CTRL=(FREEKB),                *
               TIOAPFX=YES,LANG=ASM,EXTATT=MAPONLY
XDFHAMK  DFHMDI SIZE=(12,40)
         DFHMDF POS=(01,10),LENGTH=11,ATTRB=(BRT,ASKIP),               *
               INITIAL='ORDER ENTRY',COLOR=BLUE,HILIGHT=UNDERLINE
MSG1     DFHMDF POS=(03,04),LENGTH=26,ATTRB=(DRK,ASKIP),               *
               INITIAL='NUMBER NOT FOUND - REENTER',COLOR=RED,         *
               HILIGHT=BLINK
MSG2     DFHMDF POS=(04,04),LENGTH=22,ATTRB=(DRK,ASKIP),               *
               INITIAL='DATA ERROR - REENTER',COLOR=RED,HILIGHT=BLINK
         DFHMDF POS=(05,04),LENGTH=09,ATTRB=PROT,                      *
               INITIAL='NUMBER  :'
CUSTNO   DFHMDF POS=(05,14),LENGTH=06,ATTRB=(IC,NUM)
         DFHMDF POS=(05,21),LENGTH=01
         DFHMDF POS=(06,04),LENGTH=09,ATTRB=PROT,COLOR=BLUE,           *
               INITIAL='PART NO :'
PARTNO   DFHMDF POS=(06,14),LENGTH=06,ATTRB=NUM
         DFHMDF POS=(06,21),LENGTH=01
         DFHMDF POS=(07,04),LENGTH=09,ATTRB=PROT,COLOR=BLUE            *
               INITIAL='QUANTITY:'
QUANT    DFHMDF POS=(07,14),LENGTH=06,ATTRB=NUM
         DFHMDF POS=(07,21),LENGTH=01
         DFHMDF POS=(09,01),LENGTH=38,ATTRB=ASKIP,COLOR=BLUE,          *
               INITIAL='PRESS ENTER TO CONTINUE, CLEAR TO QUIT'
         DFHMSD TYPE=FINAL
         END
```

XDFHAMK SCREEN LAYOUT

```
        +ORDER ENTRY

   +NUMBER NOT FOUND - REENTER
   +DATA ERROR - REENTER
   +NUMBER  :+XXXXXX+
   +PART NO :+XXXXXX+
   +QUANTITY:+XXXXXX+


+PRESS ENTER TO CONTINUE, CLEAR TO QUIT
```

XDFHAML MAP DEFINITION

```
MAPSET   DFHMSD TYPE=&SYSPARM,MODE=INOUT,CTRL=(FREEKB),                *
               TIOAPFX=YES,LANG=ASM
XDFHAML  DFHMDI SIZE=(05,80)
TITLE    DFHMDF POS=(01,01),LENGTH=43,                                 *
               INITIAL='NUMBER     NAME                      ADDRESS'
NUMB     DFHMDF POS=(02,01),LENGTH=06
NAM      DFHMDF POS=(02,12),LENGTH=20
ADDR     DFHMDF POS=(02,37),LENGTH=20
         DFHMDF POS=(03,01),LENGTH=09,                                 *
               INITIAL='PART NO :'
PART     DFHMDF POS=(03,11),LENGTH=06
         DFHMDF POS=(04,01),LENGTH=09,                                 *
               INITIAL='QUANTITY:'
QUANT    DFHMDF POS=(04,11),LENGTH=06
         DFHMDF POS=(05,01),LENGTH=1,                                  *
               INITIAL=' '
         DFHMSD TYPE=FINAL
         END
```

XDFHAML PRINT FORMAT

```
+NUMBER     NAME                      ADDRESS
+XXXXXX    +XXXXXXXXXXXXXXXXXXXX    +XXXXXXXXXXXXXXXXXXX
+PART NO:+XXXXXX
+QUANTITY:+XXXXXX
+X
```

ADDITIONS TO TABLES FOR ASSEMBLER-LANGUAGE SAMPLE PROGRAMS

PPT

The following entries were made for the sample maps:

```
DFHPPT TYPE=ENTRY,PROGRAM=XDFHAMA
DFHPPT TYPE=ENTRY,PROGRAM=XDFHAMB
DFHPPT TYPE=ENTRY,PROGRAM=XDFHAMC
DFHPPT TYPE=ENTRY,PROGRAM=XDFHAMD
DFHPPT TYPE=ENTRY,PROGRAM=XDFHAMK
DFHPPT TYPE=ENTRY,PROGRAM=XDFHAML
```

The following entries were made for the sample programs:

```
DFHPPT TYPE=ENTRY,PROGRAM=XDFHAMNU
DFHPPT TYPE=ENTRY,PROGRAM=XDFHAALL
DFHPPT TYPE=ENTRY,PROGRAM=XDFHABRW
DFHPPT TYPE=ENTRY,PROGRAM=XDFHAREN
DFHPPT TYPE=ENTRY,PROGRAM=XDFHACOM
DFHPPT TYPE=ENTRY,PROGRAM=XDFHAREP
```

PCT

The following entries were made for the sample programs:

```
DFHPCT TYPE=ENTRY,TRANSID=AMNU
       ,PROGRAM=XDFHAMNU
DFHPCT TYPE=ENTRY,TRANSID=AINQ
       ,PROGRAM=XDFHAALL
DFHPCT TYPE=ENTRY,TRANSID=AADD
       ,PROGRAM=XDFHAALL
DFHPCT TYPE=ENTRY,TRANSID=AUPD
       ,PROGRAM=XDFHAALL
DFHPCT TYPE=ENTRY,TRANSID=ABRW
       ,PROGRAM=XDFHABRW
DFHPCT TYPE=ENTRY,TRANSID=AORD
       ,PROGRAM=XDFHAREN
DFHPCT TYPE=ENTRY,TRANSID=ACOM
       ,PROGRAM=XDFHACOM
DFHPCT TYPE=ENTRY,TRANSID=AREP
       ,PROGRAM=XDFHAREP
```

DCT

The following entry needs to be made in order that the Order Entry Queue Print sample program is triggered when the number of items on the queue reaches 30.

```
DFHDCT TYPE=INTRA,DESTID=L860
       ,TRIGLEV=30,TRANSID=ACOM
       ,DESTFAC=TERMINAL
```

RECORD DESCRIPTIONS FOR ASSEMBLER-LANGUAGE SAMPLE PROGRAMS

FILEA RECORD DESCRIPTION

The FILEA record description is used by the sample programs and is of the following format:

```
FILEA     DS    0CL80
FILEREC   DS    0CL80
STAT      DS    CL1
NUMB      DS    CL6
NAME      DS    CL20
ADDRX     DS    CL20
PHONE     DS    CL8
DATEX     DS    CL8
AMOUNT    DS    CL8
COMMENT   DS    CL9
```

LOGA RECORD DESCRIPTION

The LOGA record description is used by the sample programs when an audit trail is written to a transient data file. It has the following format:

```
LOGA      DS    0CL92
LOGHDR    DS    0CL12
LDAY      DS    PL4
LTIME     DS    PL4
LTERML    DS    CL4
LOGREC    DS    0CL80
LSTAT     DS    CL1
LNUMB     DS    CL6
LNAME     DS    CL20
LADDR     DS    CL20
LPHONE    DS    CL8
LDATE     DS    CL8
LAMOUNT   DS    CL8
LCOMMENT  DS    CL9
```

L860 RECORD DESCRIPTION

The L860 record description is used by the Order Entry Queue Print sample program when it writes to the transient data queue 'L860'. It has the following format:

```
L860      DS    0CL22
ITEM      DS    0CL22
CUSTNO    DS    CL6
PARTNO    DS    CL6
QUANTITY  DS    CL6
TERMID    DS    CL4
```

Appendix E. Sample Programs (COBOL)

This appendix consists of sample CICS/VS application programs written in the COBOL language. The BMS maps and file record descriptions used by the sample programs are included after the sample programs.

The sample maps include examples of how the COLOR, EXTATT, and HILIGHT attributes are specified in the map definition macros. However, due to production limitations, the associated screen layouts do not show the effects of these attributes; they show how the maps would be displayed on, for example, a 3277.

Specifying EXTATT=MAPONLY enables attributes to be added without changing the application program. Any attribute, that specifies a facility not available at the terminal, will be ignored.

The sample programs illustrate basic applications that can serve as a framework for the installation's first programs. Each program has a description and program notes. The program listings are of source code. Numbered coding lines correspond to the numbered program notes. The programs contain COPY statements coded according to the 1968 COBOL standard. If the programs are to be compiled on the OS/VS COBOL compiler, LANGLVL(1) should be specified.

All transactions are initiated by the terminal operator entering a four-character transaction code. (An account number must also be entered, except in the case of the operator instruction sample program.)

There are six sample programs, as follows:

- Operator Instruction Sample Program
- Update Sample Program
- Browse Sample Program
- Order Entry Sample Program
- Order Entry Queue Print Sample Program
- Report Sample Program

All the sample programs operate on a sample VSAM or ISAM file which must first be created using a program provided on the library. The file consists of records containing details of individual accounts. The programs are used to display, alter, update, or browse through the entries. For information on how to create the sample VSAM or ISAM file refer to the CICS/VS System Programmer's Guide.

All the sample programs are for use with the IBM 3270 Information Display System.

EXECUTING THE SAMPLE PROGRAMS

Once CICS/VS is running, 3270 users can enter the following transaction id's:

MENU	Display other transaction id's (except OREN, CCOM, and REPT.)
INQY	Display an entry.
ADDS	Create a new entry.
UPDT	Update an entry.
BRWS	Browse through entries.
OREN	Order entry.
CCOM	Print order entry queue.
REPT	Display a report (entries not greater than $50).

Note: The transaction CCOM should be used once in the morning, after which it will invoke itself at the printer in one hour (unless the time is 1400 hrs or after).

OPERATOR INSTRUCTION SAMPLE PROGRAM (COBOL)

DESCRIPTION

To begin 3270 operations, a terminal operator must enter a transaction code of MENU. Whenever the screen is cleared this transaction code must be reentered, as no data is accepted from an unformatted screen.

The instruction program displays map XDFHCMA containing operator instructions. This map lists the COBOL CICS/VS sample applications and the transaction codes (with the exception of OREN and CCOM which are entered onto a clear screen), and provides space for entering the code and an account number.

SOURCE LISTING

```
       IDENTIFICATION DIVISION.
       PROGRAM-ID. INSTRUCT.
       ENVIRONMENT DIVISION.
       DATA DIVISION.
       PROCEDURE DIVISION.
    1      EXEC CICS SEND MAP('XDFHCMA') MAPONLY ERASE END-EXEC.
    2      EXEC CICS RETURN END-EXEC.
```

PROGRAM NOTES

1. The BMS command erases the screen and displays map XDFHCMA.
2. RETURN ends the program.

UPDATE SAMPLE PROGRAM (COBOL)

DESCRIPTION

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-id is 'ADDS', reads the file record. The required fields from the file area, and a title depending on the invoking transaction-id, 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. Fields are 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.

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

SOURCE LISTING

```
       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 = 'INQY'
            AND EIBTRNID NOT = 'ADDS'
            AND EIBTRNID NOT = 'UPDT' 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 = 'ADDS' 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 = 'INQY' 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('MENU') END-EXEC.
        IF EIBTRNID = 'UPDT' 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 = 'UPDT' 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 = 'ADDS' 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 = 'ADDS' THEN
           IF NAMET NOT ALPHABETIC THEN GO TO DATA-ERROR.
        IF EIBTRNID = 'UPDT' THEN
           IF NAMEI NOT = LOW-VALUES
           AND NAMET NOT ALPHABETIC THEN GO TO DATA-ERROR.
    FILE-WRITE.
        IF EIBTRNID = 'ADDS' 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  TD  QUEUE('LOGA') FROM(LOGA) LENGTH(92)
                                                        END-EXEC.
           IF EIBTRNID = 'UPDT' 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 = 'ADDS' 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-id'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 is invoked by the transaction-id 'ADDS', 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 ADDS transaction, the amount field has the modified data tag and the numeric attribute byte set on so only numeric data can be entered. 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 is invoked by the transaction-id 'INQY', a title and command message are moved to the map area.
14. This invocation of the program ends.
15. If the program is invoked by the transaction-id 'UPDT', 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 to ensure 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 map XDFHCMB are sent to the screen.

36. Control is returned to CICS/VS 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.

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 is 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.

BROWSE SAMPLE PROGRAM (COBOL)

DESCRIPTION

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 reexamine 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.

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 stored in FLDA, to be ready for a page forward.

```
| 18  | 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 (18) to retrieve records 20 and 25. These are displayed after the keys are shifted right. The last record read (25) is stored in FLDA.

```
| 25  | 18  | 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 previous 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 stored in FLDA for the next page forward.

```
| 18  | 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.

SOURCE LISTING

```
       IDENTIFICATION DIVISION.
       PROGRAM-ID.  BROWSE.
       ENVIRONMENT DIVISION.
       DATA DIVISION.
       WORKING-STORAGE SECTION.
       77  I PIC 999 USAGE IS COMP.
       77  MESSAGES PICTURE X(39) VALUE ' '.
       77  FLDA PIC 9(6) VALUE IS ZERO.
       77  FLDB PIC 9(6) VALUE IS ZERO.
       77  FLDC PIC 9(6) VALUE IS ZERO.
       77  FLDD PIC 9(6) VALUE IS ZERO.
       01  XDFHCMAI COPY XDFHCMA.
       01  XDFHCMCI COPY XDFHCMC.
       01  FILEA COPY FILEA.
       01  DFHBMSCA COPY DFHBMSCA.
       PROCEDURE DIVISION.
 1          EXEC CICS HANDLE CONDITION ERROR(ERRORS)
                             MAPFAIL (MENU)
                             NOTFND(NOTFOUND)
                             ENDFILE (ENDFILE) END-EXEC
 2          EXEC CICS RECEIVE MAP('XDFHCMA') END-EXEC
 3          EXEC CICS HANDLE AID
                    CLEAR(MENU)
                    PF1 (PAGE-FORWARD)
                    PF2 (PAGE-BACKWARD) END-EXEC
 4          MOVE KEYI TO FLDA
 5          EXEC CICS STARTBR DATASET('FILEA') RIDFLD(FLDA) END-EXEC.
       PAGE-FORWARD.
            MOVE FLDC TO FLDD
 6          MOVE FLDB TO FLDC
            MOVE FLDA TO FLDB.
       BUILD.
            MOVE 1 TO I
            MOVE LOW-VALUES TO XDFHCMCO.
       NEXT-LINE.
 7          EXEC CICS READNEXT INTO(FILEA)
                               DATASET('FILEA') RIDFLD(FLDA) END-EXEC
 8          IF STAT EQUAL HIGH-VALUE THEN GO TO NEXT-LINE.
 9          IF I = 1 MOVE NUMB TO NUMBER1O
                         THEN MOVE NAME TO NAME1O
                         THEN MOVE AMOUNT TO AMOUNT1O.
10          IF I = 2 MOVE NUMB TO NUMBER2O
                         THEN MOVE NAME TO NAME2O
                         THEN MOVE AMOUNT TO AMOUNT2O.
            IF I = 3 MOVE NUMB TO NUMBER3O
                         THEN MOVE NAME TO NAME3O
                         THEN MOVE AMOUNT TO AMOUNT3O.
            IF I = 4 MOVE NUMB TO NUMBER4O
                         THEN MOVE NAME TO NAME4O
                         THEN MOVE AMOUNT TO AMOUNT4O.
            ADD 1 TO I
            IF I NOT EQUAL 5 GO TO NEXT-LINE.
       DISPLAY-RECORD.
11          EXEC CICS SEND MAP('XDFHCMC') ERASE END-EXEC.
       REPEAT.
12          EXEC CICS RECEIVE MAP('XDFHCMC') END-EXEC
            IF DIRI EQUAL 'F' THEN GO TO PAGE-FORWARD.
            IF DIRI EQUAL 'B' THEN GO TO PAGE-BACKWARD.
            GO TO MENU.
       ENDFILE.
            MOVE 'END OF FILE' TO MSG1O
```

```
13          MOVE DFHBMBRY TO MSG2A
            GO TO DISPLAY-RECORD.
       PAGE-BACKWARD.
14          IF FLDC EQUAL ZEROS GO TO TOO-FAR.
            MOVE FLDC TO FLDA
15          MOVE FLDC TO FLDB
            MOVE FLDD TO FLDC
            MOVE ZEROS TO FLDD
            IF FLDA NOT EQUAL KEYI THEN ADD 1 TO FLDA.
            EXEC CICS RESETBR DATASET('FILEA') RIDFLD(FLDA) END-EXEC
            GO TO  BUILD.
       TOO-FAR.
16          MOVE DFHBMBRY TO MSG1A
            MOVE DFHBMDAR TO MSG2A
17          EXEC CICS SEND MAP('XDFHCMC') DATAONLY END-EXEC
            GO TO REPEAT.
       NOTFOUND.
18          MOVE 'INVALID MNUMBER - PLEASE REENTER' TO MESSAGES
            GO TO MENU.
       ERRORS.
19          EXEC CICS DUMP DUMPCODE('ERRS') END-EXEC
            MOVE 'TRANSACTION TERMINATED' TO MESSAGES.
       MENU.
20          MOVE LOW-VALUE TO XDFHCMAO
            MOVE DFHBMBRY TO MSGA
            MOVE MESSAGES TO MSGO
21          EXEC CICS SEND MAP('XDFHCMA') ERASE END-EXEC
            EXEC CICS RETURN END-EXEC.
```

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 TOO-FAR (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 XDFHCMA is displayed, and the program ends.

ORDER ENTRY SAMPLE PROGRAM (COBOL)

DESCRIPTION

The order entry sample application accepts input relating to the ordering of parts from a warehouse. When sufficient orders have been accumulated in the headquarters of a business, these are automatically sent off to a warehouse, or some other distribution point.

The program displays the map XDFHCMK on the screen requesting the operator to input details regarding the ordering of a certain part. The screen contains entry positions relating to the customer number, the part number and the quantity of that part required. (Any integer up to six digits in length may be entered: the customer number must be valid, that is, it must exist on FILEA.) When the screen has been filled, the operator presses CLEAR to stop entering data, and ENTER to continue entering data. The screen is then mapped in and the data is checked, errors being returned to the operator for reentering. When all the input is correct it is sent to a transient data queue called 'L860' - which is also a terminal-id where a transaction is to be triggered when the number of items on the queue reaches 30.

The trigger level may be changed using the CSMT command, as follows:

```
CSMT TRIGGER,n,DESTID=L860
```

where n is the destination trigger level (any integer from 0 through 32767).

SOURCE LISTING

```
       IDENTIFICATION DIVISION.
       PROGRAM-ID.  XDFHOREN.
       ENVIRONMENT DIVISION.
       DATA DIVISION.
       WORKING-STORAGE SECTION.
       77  ERROR-FLAG PIC 9.
       77  WNG-MSG PIC 9 VALUE 0.
       77  BRTMDT PIC X VALUE IS 'I'.
       01  XDFHCMKI COPY XDFHCMK.
       01  FILEA COPY FILEA.
       01  L860 COPY L860.
       01  DFHBMSCA COPY DFHBMSCA.
       PROCEDURE DIVISION.
 1         EXEC CICS HANDLE AID CLEAR(ENDA) END-EXEC.
 2         EXEC CICS HANDLE CONDITION MAPFAIL(MAPFAIL)
                                      NOTFND(NOTFOUND)
                                      ERROR(ERRORS) END-EXEC.
           MOVE LOW-VALUES TO XDFHCMKO.
       SENDM.
 3         EXEC CICS SEND MAP('XDFHCMK') ERASE END-EXEC.
       RECEIVEM.
 4         EXEC CICS RECEIVE MAP('XDFHCMK') END-EXEC.
       TEST.
           MOVE 0 TO ERROR-FLAG
 5         MOVE DFHBMFSE TO CUSTNOA, PARTNOA, QUANTA.
           IF CUSTNOI NOT NUMERIC THEN
              MOVE BRTMDT TO CUSTNOA MOVE 1 TO ERROR-FLAG.
           IF PARTNOI NOT NUMERIC THEN
 6            MOVE BRTMDT TO PARTNOA MOVE 1 TO ERROR-FLAG.
           IF QUANTI NOT NUMERIC THEN
              MOVE BRTMDT TO QUANTA MOVE 1 TO ERROR-FLAG.
           IF ERROR-FLAG = 1 THEN
              MOVE 1 TO WNG-MSG
 7            MOVE DFHBMBRY TO MSG2A GO TO SENDM.
 8         EXEC CICS READ DATASET('FILEA') INTO(FILEA) RIDFLD(CUSTNOI)
                                                       END-EXEC.
       Q-BUILD.
           MOVE CUSTNOI TO CUSTNO
 9         MOVE PARTNOI TO PARTNO
           MOVE QUANTI TO QUANTITY
           MOVE EIBTRMID TO TERMID.
       Q-WRITE.
10         EXEC CICS WRITEQ TD QUEUE('1860') FROM(L860) LENGTH(22)
                                                        END-EXEC.
11         IF WNG-MSG = 1 THEN
              EXEC CICS SEND MAP('XDFHCMK') MAPONLY ERASE END-EXEC
```

```
                MOVE 0 TO WNG-MSG
            ELSE
12              EXEC CICS ISSUE ERASEAUP END-EXEC.
            GO TO RECEIVEM.
        NOTFOUND.
            MOVE 1 TO WNG-MSG.
13          MOVE DFHBMASB TO MSG1A
            GO TO SENDM.
        MAPFAIL.
            MOVE 1 TO WNG-MSG.
14          MOVE LOW-VALUES TO XDFHCMKO.
            MOVE DFHBMASB TO MSG2A
            GO TO SENDM.
        ERRORS.
15          MOVE 'TRANSACTION TERMINATED' TO MSG2O
            MOVE DFHBMBRY TO MSG2A
            EXEC CICS SEND MAP('XDFHCMK') END-EXEC
            EXEC CICS DUMP DUMPCODE('ERRS') END-EXEC.
        ENDA.
16          EXEC CICS RETURN END-EXEC.
            GOBACK.
```

PROGRAM NOTES

1. The exit for the clear key is set up.
2. The program exits are set up.
3. The screen is erased and the map is displayed at the terminal.
4. This command maps in the customer number, part number, and quantity.
5. The input areas on the map have the modified data tag set on in case they need to be sent back for reinput, should an error occur in entering data.
6. The input is tested, and erroneous fields are brightened, whilst the modified data tag is still set on. The user should add further editing necessary to ensure only valid orders are accepted.
7. If there is a data error, the message 'DATA ERROR - REENTER', having been stored on the screen with a dark attribute character, is brightened.
8. The file control READ reads the record into a record area in order to find whether a particular record exists.
9. The input from the map is moved to the queue area.
10. The transient data WRITEQ obtains a log area, and writes this record to a sequential file.
11. If an error message is left on the screen, the screen is cleared and only the map is sent.
12. The entered fields, having been mapped in and processed, are erased, and the screen is ready to receive more input.
13. If the customer number entered was not found, the message 'NUMBER NOT FOUND - REENTER', having been stored on the screen with a dark attribute character, is brightened.
14. If no fields were entered, the message 'DATA ERROR - REENTER', also having been stored on the screen with a dark attribute character, is brightened.
15. On an error (notes 3, 4, 8, 10, 11, and 15) a dump is taken, and the message 'TRANSACTION TERMINATED ' is moved to the top message area.
16. The program ends.

ORDER ENTRY QUEUE PRINT SAMPLE PROGRAM (COBOL)

DESCRIPTION

This transaction is invoked by entering the transaction-id 'CCOM' at the terminal. The program checks to see whether it was started from a terminal or the printer. If from a terminal, (that is, the operator is starting this transaction for the first time) the program starts the transaction at the printer in one hour. (This time interval could be changed using EDF for demonstration purposes.) The operator may then press RESET and CLEAR and enter another transaction. If from the printer, the program executes and starts again in one hour. If there are no items on the queue, a message indicating that the queue is empty, is sent to the warehouse. The last communications with the warehouse occurs not later than 1500 hours. This transaction is also started when the number of items on the queue 'L860' reaches 30.

The trigger level may be changed using the CSMT command, as follows:

```
CSMT TRIGGER,n,DESTID=L860
```

where n is the destination trigger level (any integer from 0 through 32767).

This program reads items off the queue 'L860', until the queue is empty. Should the queue have been empty initially, a message is sent to the warehouse. Using the number from the queue as a key it reads the file FILEA, and checks the amount field to see if the customer is good for credit on this order. If he is, the number, name, address, part number and quantity are moved to the map XDFHCML and this is sent to the printer. If he is not, the time, date, queue-item, and a comment field are moved to a data area, this may be used for later processing. A message is then sent to the warehouse indicating that the queue is empty. The EIBTIME is then updated and if the time is before 1400 hours, the transaction is started in one hour.

SOURCE LISTING

```
       IDENTIFICATION DIVISION.
       PROGRAM-ID.  XDFHCCOM.
       ENVIRONMENT DIVISION.
       DATA DIVISION.
       WORKING-STORAGE SECTION.
       77  Q-LENGTH PIC S9(4) COMP.
       01  LOGORD.
           02  LOGTIME.
               03  LDAY PIC S9(7) COMP-3.
               03  LTIME PIC S9(7) COMP-3.
           02  LITEM PIC X(22).
           02  COMMENT PIC X(11) VALUE 'ORDER ENTRY'.
           02  FILLER PIC X(51) VALUE SPACES.
       01  XDFHCMLO COPY XDFHCML.
       01  FILEA COPY FILEA.
       01  L860 COPY L860.
       01  DFHBMSCA COPY DFHBMSCA.
       PROCEDURE DIVISION.
  1       EXEC CICS HANDLE CONDITION ERROR(ERRORS)
                                   QZERO(ENDA) END-EXEC.
  2        IF EIBTRMID NOT = 'L860' THEN
              GO TO TIME.
           MOVE LOW-VALUES TO XDFHCMLO.
       Q-READ.
           MOVE 22 TO Q-LENGTH.
  3        EXEC CICS READQ TD INTO(L860) LENGTH(Q-LENGTH)
                              QUEUE('L860') END-EXEC.
       MAP-BUILD.
  4        EXEC CICS READ DATASET('FILEA') INTO(FILEA) RIDFLD(CUSTNO)
                                                       END-EXEC
  5        IF AMOUNT  > '$0100.00' THEN
              MOVE ADDRX TO ADDRO
              MOVE NAME TO NAMO
  6           MOVE PARTNO TO PARTO
              MOVE NUMB TO NUMBO
              MOVE ITEM TO LITEM
              MOVE QUANTITY TO QUANTO
  7           EXEC CICS SEND MAP('XDFHCML') ERASE PRINT L80 END-EXEC
              GO TO Q-READ
```

```
           ELSE
              MOVE EIBDATE TO LDAY
 8            MOVE EIBTIME TO LTIME
              MOVE ITEM TO LITEM
 9            EXEC CICS WRITEQ TD QUEUE('LOGA')
                                 FROM(LOGORD) LENGTH(92) END-EXEC
              GO TO Q-READ.
       ERRORS.
 10            EXEC CICS DUMP DUMPCODE('ERRS') END-EXEC.
               GO TO FIN.
       ENDA.
 11        MOVE LOW-VALUES TO XDFHCMLO
           MOVE 'ORDER QUEUE IS EMPTY' TO TITLEO
 12        EXEC CICS SEND MAP('XDFHCML') DATAONLY ERASE PRINT L80
                                                        END-EXEC.
       TIME.
 13        EXEC CICS ASKTIME END-EXEC.
 14        IF EIBTIME NOT > 140000 THEN
 15           EXEC CICS START TRANSID('CCOM') INTERVAL(10000)
                              TERMID('L860') END-EXEC.
       FIN.
 16        EXEC CICS RETURN END-EXEC.
           GOBACK.
```

PROGRAM NOTES

1. The program exits are set up.
2. The terminal-id is tested to see whether this transaction was started from a terminal or at the printer.
3. The queue item is read into the program.
4. The file control READ reads the record into a record area so that the amount may be checked.
5. The amount is tested.
6. If it is over $100, the record on the queue is moved to the map XDFHCML. This test is only a suggestion; a suitable form of editing should be inserted to ensure valid orders are sent to the warehouse.
7. The map XDFHCML is sent to the printer.
8. If the order is not valid for this account, the record on the queue is moved to a data area, together with the terminal-id associated with the entering of this piece of data, the time, and date.
9. The transient data WRITEQ obtains a log area, and writes this record to a sequential file.
10. On an error (notes 3, 4, 7, 9, 10, 12 and 15) a dump is taken.
11. When the queue is empty, a message is moved to the map area.
12. The map is displayed on the screen.
13. The current time-of-day clock is updated.
14. The current time-of-day is tested.
15. If the current time is not past 1400 hours, the transaction is started again in one hour at the warehouse printer.
16. The program ends.

REPORT SAMPLE PROGRAM (COBOL)

DESCRIPTION

The report sample program produces a report that lists all entries in the data set 'FILEA' for which the amount is less than or equal to $50.00.

The program illustrates page building techniques and the use of the terminal paging facilities of BMS.

The transaction is invoked by entering the transaction code REPT. The program does a sequential scan through the file noting each entry that obeys the search criterion. The pages are built from four maps which comprise mapset XDFHCMD, using the paging option so that the data is not displayed immediately but instead is stored for later retrieval. The HEADING map is inserted at the head of each page. The detail map (XDFHCMD) is written repeatedly until the overflow condition occurs. The FOOTING map is then written at the foot of the page and the HEADING map written at the top of the next page. The command to write the detail map that caused overflow is then repeated. When all the data has been written the FINAL map is written at the bottom of the last page and the transaction terminated.

The terminal operator then enters paging commands to display the data, clearing the screen before entering each paging command.

SOURCE LISTING

```
       IDENTIFICATION DIVISION.
       PROGRAM-ID.  REPORTC
       ENVIRONMENT DIVISION.
       DATA DIVISION.
       WORKING-STORAGE SECTION.
       77  LOWLIM PICTURE X(8) VALUE '$0050.00'.
       77  KEYNUM PICTURE 9(6) VALUE 0.
       77  PAGEN PICTURE 9(3) VALUE 1.
       77  OPINSTR PICTURE X(22) VALUE 'ENTER PAGING COMMANDS.'.
       01  XDFHCMDI COPY XDFHCMD.
       01  FILEA COPY FILEA.
       PROCEDURE DIVISION.
 1         EXECUTE CICS HANDLE CONDITION ERROR(ERRORS)
                OVERFLOW(OFLOW) ENDFILE(ENDFILE) END-EXEC
 2         MOVE LOW-VALUE TO PAGENA
           MOVE PAGEN TO PAGENO
 3         EXEC CICS SEND MAP('HEADING') MAPSET('XDFHCMD') ACCUM
                PAGING ERASE END-EXEC
 4         MOVE 0 TO KEYNUM.
 5         EXEC CICS STARTBR DATASET('FILEA') RIDFLD(KEYNUM) END-EXEC.
       REPEAT.
 6         EXEC CICS READNEXT INTO(FILEA) RIDFLD(KEYNUM)
                               DATASET('FILEA') END-EXEC
 7         IF STAT EQUAL HIGH-VALUE GO TO REPEAT.
           MOVE AMOUNT TO AMOUNTO
 8         IF AMOUNTO GREATER THAN LOWLIM GO TO REPEAT.
           MOVE LOW-VALUE TO XDFHCMDO
 9         MOVE AMOUNT TO AMOUNTO
           MOVE NUMB TO NUMBERO
           MOVE NAME TO NAMEO
           GO TO SEND-RECORD.
        OFLOW.
10            EXEC CICS SEND MAP('FOOTING') MAPSET('XDFHCMD')
                            MAPONLY ACCUM PAGING END-EXEC
           ADD 1 TO PAGEN
           MOVE PAGEN TO PAGENO
11            EXEC CICS SEND MAP('HEADING') MAPSET('XDFHCMD')
                            ACCUM PAGING ERASE END-EXEC.
       SEND-RECORD.
12           EXEC CICS SEND MAP('XDFHCMD') MAPSET('XDFHCMD')
                                    ACCUM PAGING END-EXEC
           GO TO REPEAT.
       ENDFILE.
13           EXEC CICS SEND MAP('FINAL') MAPSET('XDFHCMD')
                  MAPONLY ACCUM PAGING END-EXEC
14           EXEC CICS SEND PAGE END-EXEC
15         EXEC CICS SEND TEXT FROM(OPINSTR) LENGTH(22) ERASE END-EXEC
```

```
16          EXEC CICS ENDBR DATASET('FILEA') END-EXEC
17          EXEC CICS RETURN END-EXEC.
        ERRORS.
18          EXEC CICS HANDLE CONDITION ERROR END-EXEC
19          EXEC CICS PURGE MESSAGE END-EXEC
20          EXEC CICS ABEND ABCODE('ERRS') END-EXEC.
```

PROGRAM NOTES

1. The program exits are set up.
2. The attribute byte for the page number is cleared.
3. This BMS request sets up the heading in the page build operation.
4. The initial key value is set up for the START BROWSE command.
5. This command starts the browse through the file, at a record whose key is equal to or greater than that specified.
6. This command reads the next record on the file into the file area.
7. If the record is coded as deleted, it is treated as not found.
8. The search criterion for creating the report is that the customer has less than or equal to $50.
9. Fields are moved from the file area to the map area.
10. The BMS request sets up the footing in the page build operation.
11. The BMS request sets up the heading in the page build operation.
12. The customer detail map is set up.
13. When the END OF FILE condition is raised, the last map is built.
14. The page is sent to the terminal operator.
15. A message is sent to the terminal.
16. The BROWSE operation is ended.
17. The program ends.
18. On an error, the label to branch to on the ERROR condition is reset.
19. Any pages waiting to be displayed at the terminal are purged.
20. The program raises an abend condition, a dump is taken and the program ends.

SAMPLE MAPS AND SCREEN LAYOUTS FOR COBOL SAMPLE PROGRAMS

XDFHCMA 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=29,INITIAL='OPERATOR INSTR - ENTER MEN*
               U'
         DFHMDF POS=(4,1),LENGTH=38,INITIAL='FILE INQUIRY   - ENTER INQ*
               Y AND NUMBER'
         DFHMDF POS=(5,1),LENGTH=38,INITIAL='FILE BROWSE    - ENTER BRW*
               S AND NUMBER'
         DFHMDF POS=(6,1),LENGTH=38,INITIAL='FILE ADD       - ENTER ADD*
               S AND NUMBER'
         DFHMDF POS=(7,1),LENGTH=38,INITIAL='FILE UPDATE    - ENTER UPD*
               T 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
```

XDFHCMA SCREEN LAYOUT

```
          +OPERATOR INSTRUCTIONS

+OPERATOR INSTR - ENTER MENU
+FILE INQUIRY   - ENTER INQY AND NUMBER
+FILE BROWSE    - ENTER BRWS AND NUMBER
+FILE ADD       - ENTER ADDS AND NUMBER
+FILE UPDATE    - ENTER UPDT AND NUMBER



+PRESS PA1 TO PRINT--PRESS CLEAR TO EXIT
+ENTER TRANSACTION:+XXXX+NUMBER+XXXXXX+
```

XDFHCMB 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
```

XDFHCMB SCREEN LAYOUT

```
          +XXXXXXXXXXXX

+NUMBER: +XXXXXX+
+NAME:   +XXXXXXXXXXXXXXXXXXX+
+ADDRESS:+XXXXXXXXXXXXXXXXXXX+
+PHONE:  +XXXXXXXX+
+DATE:   +XXXXXXXX+
+AMOUNT: +XXXXXXXX+
+COMMENT:+XXXXXXXXX+

+XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
+XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
```

XDFHCMC MAP DEFINITION

```
MAPSET   DFHMSD TYPE=&SYSPARM,MODE=INOUT,CTRL=(FREEKB,FRSET),          *
               LANG=COBOL,TIOAPFX=YES,EXTATT=MAPONLY
XDFHCMC  DFHMDI SIZE=(12,40)
DIR      DFHMDF POS=(1,1),LENGTH=1,ATTRB=IC
         DFHMDF POS=(1,3),LENGTH=1
         DFHMDF POS=(1,15),LENGTH=11,INITIAL='FILE BROWSE',            *
               HILIGHT=UNDERLINE,COLOR=BLUE
         DFHMDF POS=(3,1),LENGTH=6,INITIAL='NUMBER',COLOR=BLUE
         DFHMDF POS=(3,17),LENGTH=4,INITIAL='NAME',COLOR=BLUE
         DFHMDF POS=(3,32),LENGTH=6,INITIAL='AMOUNT',COLOR=BLUE
NUMBER1  DFHMDF POS=(4,1),LENGTH=6
NAME1    DFHMDF POS=(4,9),LENGTH=20
AMOUNT1  DFHMDF POS=(4,30),LENGTH=8
NUMBER2  DFHMDF POS=(5,1),LENGTH=6
NAME2    DFHMDF POS=(5,9),LENGTH=20
AMOUNT2  DFHMDF POS=(5,30),LENGTH=8
NUMBER3  DFHMDF POS=(6,1),LENGTH=6
NAME3    DFHMDF POS=(6,9),LENGTH=20
AMOUNT3  DFHMDF POS=(6,30),LENGTH=8
NUMBER4  DFHMDF POS=(7,1),LENGTH=6
NAME4    DFHMDF POS=(7,9),LENGTH=20
AMOUNT4  DFHMDF POS=(7,30),LENGTH=8
MSG1     DFHMDF POS=(11,1),LENGTH=39,                                  *
               INITIAL='PRESS PF1 OR TYPE F TO PAGE FORWARD'
MSG2     DFHMDF POS=(12,1),LENGTH=39,                                  *
               INITIAL='PRESS PF2 OR TYPE B TO PAGE BACKWARD'
         DFHMSD TYPE=FINAL
         END
```

XDFHCMC SCREEN LAYOUT

```
                +FILE BROWSE

+NUMBER          +NAME          +AMOUNT
+XXXXXX +XXXXXXXXXXXXXXXXXXX+XXXXXX
+XXXXXX +XXXXXXXXXXXXXXXXXXX+XXXXXX
+XXXXXX +XXXXXXXXXXXXXXXXXXX+XXXXXX
+XXXXXX +XXXXXXXXXXXXXXXXXXX+XXXXXX



+PRESS PF1 OR TYPE F TO PAGE FORWARD
+PRESS PF2 OR TYPE B TO PAGE BACKWARD
+XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
+XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
```

XDFHCMD MAP DEFINITION

```
MAPSETD  DFHMSD TYPE=&SYSPARM,MODE=INOUT,CTRL=(FREEKB,FRSET),          *
               LANG=COBOL,STORAGE=AUTO,EXTATT=MAPONLY,COLOR=BLUE
XDFHCMD  DFHMDI SIZE=(1,40),COLOR=GREEN
NUMBER   DFHMDF POS=(1,1),LENGTH=6
NAME     DFHMDF POS=(1,9),LENGTH=20
AMOUNT   DFHMDF POS=(1,30),LENGTH=8
HEADING  DFHMDI SIZE=(3,40),HEADER=YES
         DFHMDF POS=(1,5),LENGTH=18,INITIAL='LOW BALANCE REPORT',      *
               HILIGHT=UNDERLINE
         DFHMDF POS=(1,30),LENGTH=4,INITIAL='PAGE'
PAGEN    DFHMDF POS=(1,35),LENGTH=3
         DFHMDF POS=(3,1),LENGTH=6,INITIAL='NUMBER'
         DFHMDF POS=(3,17),LENGTH=4,INITIAL='NAME'
         DFHMDF POS=(3,32),LENGTH=6,INITIAL='AMOUNT'
FOOTING  DFHMDI SIZE=(2,40),TRAILER=YES,JUSTIFY=LAST
         DFHMDF POS=(2,10),LENGTH=25,                                  *
               INITIAL='CONTINUED ON NEXT PAGE...'
FINAL    DFHMDI SIZE=(2,40),TRAILER=YES,JUSTIFY=LAST
         DFHMDF POS=(2,10),LENGTH=14,INITIAL='END OF REPORT.'
 .       DFHMSD TYPE=FINAL
         END
```

XDFHCMD SCREEN LAYOUT

```
    LOW BALANCE REPORT        PAGE XXX

NUMBER           NAME             AMOUNT
XXXXXX XXXXXXXXXXXXXXXXXXX  XXXXXXXX
XXXXXX XXXXXXXXXXXXXXXXXXX  XXXXXXXX
     (REPEAT TOTAL OF 19 TIMES)
XXXXXX XXXXXXXXXXXXXXXXXXX  XXXXXXXX
XXXXXX XXXXXXXXXXXXXXXXXXX  XXXXXXXX
XXXXXX XXXXXXXXXXXXXXXXXXX  XXXXXXXX

         CONTINUED ON NEXT PAGE...
```

XDFHCMK MAP DEFINITION

```
MAPSET   DFHMSD TYPE=&SYSPARM,MODE=INOUT,CTRL=(FREEKB),                *
               TIOAPFX=YES,LANG=COBOL,EXTATT=MAPONLY
XDFHCMK  DFHMDI SIZE=(12,40)
         DFHMDF POS=(01,10),LENGTH=11,ATTRB=(BRT,ASKIP),               *
               INITIAL='ORDER ENTRY',COLOR=BLUE,HILIGHT=UNDERLINE
MSG1     DFHMDF POS=(03,04),LENGTH=26,ATTRB=(DRK,ASKIP),               *
               INITIAL='NUMBER NOT FOUND - REENTER',COLOR=RED,         *
               HILIGHT=BLINK
MSG2     DFHMDF POS=(04,04),LENGTH=22,ATTRB=(DRK,ASKIP),               *
               INITIAL='DATA ERROR - REENTER',COLOR=RED,               *
               HILIGHT=BLINK
         DFHMDF POS=(05,04),LENGTH=09,ATTRB=PROT,                      *
               INITIAL='NUMBER  :'
CUSTNO   DFHMDF POS=(05,14),LENGTH=06,ATTRB=(IC,NUM)
         DFHMDF POS=(05,21),LENGTH=01
         DFHMDF POS=(06,04),LENGTH=09,ATTRB=PROT,COLOR=BLUE            *
               INITIAL='PART NO :'
PARTNO   DFHMDF POS=(06,14),LENGTH=06,ATTRB=NUM
         DFHMDF POS=(06,21),LENGTH=01
         DFHMDF POS=(07,04),LENGTH=09,ATTRB=PROT,COLOR=BLUE            *
               INITIAL='QUANTITY:'
QUANT    DFHMDF POS=(07,14),LENGTH=06,ATTRB=NUM
         DFHMDF POS=(07,21),LENGTH=01
         DFHMDF POS=(09,01),LENGTH=38,ATTRB=ASKIP,COLOR=BLUE           *
               INITIAL='PRESS ENTER TO CONTINUE, CLEAR TO QUIT'
         DFHMSD TYPE=FINAL
         END
```

XDFHCMK SCREEN LAYOUT

```
        +ORDER ENTRY

   +NUMBER NOT FOUND - REENTER
   +DATA ERROR - REENTER
   +NUMBER  :+XXXXXX+
   +PART NO :+XXXXXX+
   +QUANTITY:+XXXXXX+

+PRESS ENTER TO CONTINUE, CLEAR TO QUIT
```

XDFHCML MAP DEFINITION

```
MAPSET   DFHMSD TYPE=&SYSPARM,MODE=OUT,                              *
               TIOAPFX=YES,LANG=COBOL
XDFHCML  DFHMDI SIZE=(05,80)
TITLE    DFHMDF POS=(01,01),LENGTH=43,                               *
               INITIAL='NUMBER     NAME                    ADDRESS'
NUMB     DFHMDF POS=(02,01),LENGTH=06
NAM      DFHMDF POS=(02,12),LENGTH=20
ADDR     DFHMDF POS=(02,37),LENGTH=20
         DFHMDF POS=(03,01),LENGTH=09,                               *
               INITIAL='PART NO :'
PART     DFHMDF POS=(03,11),LENGTH=06
         DFHMDF POS=(04,01),LENGTH=09,                               *
               INITIAL='QUANTITY:'
QUANT    DFHMDF POS=(04,11),LENGTH=06
         DFHMDF POS=(05,01),LENGTH=1,                                *
               INITIAL=' '
         DFHMSD TYPE=FINAL
         END
```

XDFHCML PRINT LAYOUT

```
+NUMBER     NAME                      ADDRESS
+XXXXXX    +XXXXXXXXXXXXXXXXXXXX    +XXXXXXXXXXXXXXXXXXXX
+PART NO:+XXXXXX
+QUANTITY:+XXXXXX
+X
```

ADDITIONS TO TABLES FOR COBOL SAMPLE PROGRAMS

PPT

The following entries were made for the sample maps:

```
DFHPPT TYPE=ENTRY,PROGRAM=XDFHCMA
DFHPPT TYPE=ENTRY,PROGRAM=XDFHCMB
DFHPPT TYPE=ENTRY,PROGRAM=XDFHCMC
DFHPPT TYPE=ENTRY,PROGRAM=XDFHCMD
DFHPPT TYPE=ENTRY,PROGRAM=XDFHCMK
DFHPPT TYPE=ENTRY,PROGRAM=XDFHCML
```

The following entries were made for the sample programs:

```
DFHPPT TYPE=ENTRY,PROGRAM=XDFHINST
       ,PGMLANG=COBOL
DFHPPT TYPE=ENTRY,PROGRAM=XDFHCALL
       ,PGMLANG=COBOL
DFHPPT TYPE=ENTRY,PROGRAM=XDFHBRWS
       ,PGMLANG=COBOL
DFHPPT TYPE=ENTRY,PROGRAM=XDFHOREN
       ,PGMLANG=COBOL
DFHPPT TYPE=ENTRY,PROGRAM=XDFHCCOM
       ,PGMLANG=COBOL
DFHPPT TYPE=ENTRY,PROGRAM=XDFHREPT
       ,PGMLANG=COBOL
```

PCT

The following entries were made for the sample programs:

```
DFHPCT TYPE=ENTRY,TRANSID=MENU
       ,PROGRAM=XDFHINST
DFHPCT TYPE=ENTRY,TRANSID=INQY
       ,PROGRAM=XDFHCALL
DFHPCT TYPE=ENTRY,TRANSID=ADDS
       ,PROGRAM=XDFHCALL
DFHPCT TYPE=ENTRY,TRANSID=UPDT
       ,PROGRAM=XDFHCALL
DFHPCT TYPE=ENTRY,TRANSID=BRWS
       ,PROGRAM=XDFHBRWS
DFHPCT TYPE=ENTRY,TRANSID=OREN
       ,PROGRAM=XDFHOREN
DFHPCT TYPE=ENTRY,TRANSID=CCOM
       ,PROGRAM=XDFHCCOM
DFHPCT TYPE=ENTRY,TRANSID=REPT
       ,PROGRAM=XDFHREPT
```

DCT

The following entry was made:

```
DFHDCT TYPE=INTRA,DESTID=L860
       ,TRIGLEV=30, TRANSID=CCOM
       ,DESTFAC=TERMINAL
```

RECORD DESCRIPTIONS FOR COBOL SAMPLE PROGRAMS

FILEA RECORD DESCRIPTION

The FILEA record description is used by the sample programs and is of the following format:

```
01  FILEA.
    02 FILEREC.
       03 STAT PICTURE X.
       03 NUMB PICTURE X(6).
       03 NAME PICTURE X(20).
       03 ADDRX PICTURE X(20).
       03 PHONE PICTURE X(8).
       03 DATEX PICTURE X(8).
       03 AMOUNT PICTURE X(8).
       03 COMMENT PICTURE X(9).
```

LOGA RECORD DESCRIPTION

The LOGA record description is used by the sample programs when an audit trail is written to a transient data file. It has the following format:

```
01  LOGA.
    02 LOGHDR.
       03 LDAY PICTURE 9(7) COMP-3.
       03 LTIME PICTURE 9(7) COMP-3.
       03 LTERML PICTURE X(4).
    02 LOGREC.
       03 LSTAT PICTURE X.
       03 LNUMB PICTURE X(6).
       03 LNAME PICTURE X(20).
       03 LADDR PICTURE X(20).
       03 LPHONE PICTURE X(8).
       03 LDATE PICTURE X(8).
       03 LAMOUNT PICTURE X(8).
       03 LCOMMENT PICTURE X(9).
```

L860 RECORD DESCRIPTION

The L860 record description is used by the Order Entry Queue Print sample program when it writes to the transient data queue 'L860'. It has the following format:

```
01  L860.
    02  ITEM.
       03  CUSTNO PICTURE X(6).
       03  PARTNO PICTURE X(6).
       03  QUANTITY PICTURE X(6).
       03  TERMID PICTURE X(4).
```

Appendix F. Sample Programs (PL/I)

This appendix consists of sample CICS/VS application programs written in the PL/I language. The BMS maps and file record descriptions used by the sample programs are included after the sample programs.

The sample maps include examples of how the COLOR, EXTATT, and HILIGHT attributes are specified in the map definition macros. However, due to production limitations, the associated screen layouts do not show the effects of these attributes; they show how the maps would be displayed on, for example, a 3277.

Specifying EXTATT=MAPONLY enables attributes to be added without changing the application program. Any attribute, that specifies a facility not available at the terminal, will be ignored.

The sample programs illustrate basic applications that can serve as a framework for the installation's first programs. Each program has a description and program notes. The program listings are of source code. Numbered coding lines correspond to the numbered program notes.

All transactions are initiated by the terminal operator entering a four-character transaction code. (An account number must also be entered, except in the case of the operator instruction sample program.)

There are six sample programs, as follows:

- Operator Instruction Sample Program
- Update Sample Program
- Browse Sample Program
- Order Entry Sample Program
- Order Entry Queue Print Sample Program
- Report Sample Program

All the sample programs operate on a sample VSAM or ISAM file which must first be created using a program provided on the library. The file consists of records containing details of individual accounts. The programs are used to display, alter, update, or browse through the entries. For information on how to create the sample VSAM or ISAM file refer to the CICS/VS System Programmer's Guide.

All the sample programs are for use with the IBM 3270 Information Display System.

EXECUTING THE SAMPLE PROGRAMS

Once CICS/VS is running, 3270 users can enter the following transaction id's:

PMNU	Display other transaction id's (except PORD, PCOM, and PREP.)
PINQ	Display an entry.
PADD	Create a new entry.
PUPD	Update an entry.
PBRW	Browse through entries.
PORD	Order entry.
PCOM	Print order entry queue.
PREP	Display a report (entries not greater than $50).

Note: The transaction PCOM should be used once in the morning, after which it will invoke itself at the printer in one hour (unless the time is 1400 hrs or after).

OPERATOR INSTRUCTION SAMPLE PROGRAM (PL/I)

DESCRIPTION

To begin 3270 operations, a terminal operator must enter a transaction code of PMNU. Whenever the screen is cleared this transaction code must be reentered, as no data is accepted from an unformatted screen.

The instruction program displays map XDFHPMA containing operator instructions. This map lists the PL/I CICS/VS sample applications and the transaction codes (with the exception of PORD and PCOM which are entered onto a clear screen), and provides space for entering the code and an account number.

SOURCE LISTING

```
  INSTRCT: PROC OPTIONS(MAIN);
1    EXEC CICS SEND MAP('XDFHPMA') MAPONLY ERASE;
2    EXEC CICS RETURN;
     END;
```

PROGRAM NOTES

1. The BMS command erases the screen and displays map XDFHPMA.
2. RETURN ends the program.

UPDATE SAMPLE PROGRAM (PL/I)

DESCRIPTION

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-id, 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. Fields are 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.

This program demonstrates a pseudo-conversational programming technique, where control is returned to CICS/VS together with a transaction-id 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.

SOURCE LISTING

```
 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:
   38    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 is invoked by the transaction-id '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. 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 is invoked by the transaction-id 'PINQ', a title and command message are moved to the map area.

12. This invocation of the program ends.

13. If the program is invoked by the transaction-id '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 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 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 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 map XDFHPMB are sent to the screen.

34. Control is returned to CICS/VS 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.

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.

BROWSE SAMPLE PROGRAM (PL/I)

DESCRIPTION

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 reexamine 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.

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 stored in FLDA, to be ready for a page forward.

```
| 18   | 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 (18) to retrieve records 20 and 25. These are displayed after the keys are shifted right. The last record read (25) is stored in FLDA.

```
| 25   | 18   | 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 previous 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 stored in FLDA for the next page forward.

```
| 18   | 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.

SOURCE LISTING

```
 BROWSE: PROC OPTIONS(MAIN);
   DCL I FIXED BIN(15);
   DCL MESSAGES CHAR(39) INIT('');
   DCL (FLDA,FLDB,FLDC,FLDD) PIC'999999' INIT(0);
   DCL STRING CHAR(256) BASED;
   %INCLUDE XDFHPMA;
   %INCLUDE XDFHPMC;
   %INCLUDE FILEA;
   %INCLUDE DFHBMSCA;
 1 EXEC CICS HANDLE CONDITION ERROR(ERRORS)
                              MAPFAIL(PMNU)
                              NOTFND(NOTFOUND)
                              ENDFILE(ENDFILE);
 2 EXEC CICS RECEIVE MAP('XDFHPMA');
 3 EXEC CICS HANDLE AID CLEAR(PMNU)
                        PF1(PAGE_FORWARD)
                        PF2(PAGE_BACKWARD);
 4 FLDA=KEYI;
 5 EXEC CICS STARTBR DATASET('FILEA') RIDFLD(FLDA);
  PAGE_FORWARD:
   FLDD=FLDC;
 6 FLDC=FLDB;
   FLDB=FLDA;
  BUILD: I=1;
   SUBSTR(ADDR(XDFHPMCO)->STRING,1,STG(XDFHPMCO)) = LOW(STG(XDFHPMCO));
  NEXT_LINE:
 7 EXEC CICS READNEXT INTO(FILEA) DATASET('FILEA') RIDFLD(FLDA);
 8 IF STAT=HIGH(1) THEN GOTO NEXT_LINE;
 9 IF I=1 THEN DO;NUMBER1O=NUMB;
                  NAME1O=NAME;
                  AMOUNT1O=AMOUNT;
              END;
   ELSE
10 IF I=2 THEN DO; NUMBER2O=NUMB;
                   NAME2O=NAME;
                   AMOUNT2O=AMOUNT;
                   END;
   ELSE
   IF I=3 THEN DO; NUMBER3O=NUMB;
                   NAME3O=NAME;
                   AMOUNT3O=AMOUNT;
                   END;
   ELSE
   IF I=4 THEN DO; NUMBER4O=NUMB;
                   NAME4O=NAME;
                   AMOUNT4O=AMOUNT;
                END;
   I=I+1;
   IF I¬=5 THEN GOTO NEXT_LINE;
  DISPLAY_RECORD:
11 EXEC CICS SEND MAP('XDFHPMC') ERASE ;
  REPEAT:
12 EXEC CICS RECEIVE MAP('XDFHPMC');
   IF DIRI = 'F' THEN GOTO PAGE_FORWARD;
   IF DIRI = 'B' THEN GOTO PAGE_BACKWARD;
   GOTO PMNU;
  ENDFILE:
   MSG1O='END OF FILE';
13 MSG2A=DFHBMBRY;
   GOTO DISPLAY_RECORD;
  PAGE_BACKWARD:
14 IF FLDC=0 THEN GOTO TOO_FAR;
   FLDA=FLDC;
```

```
15 FLDB=FLDC;
    FLDC=FLDD;
   FLDD=0;
   IF FLDA¬=KEYI THEN FLDA=FLDA+1;
   EXEC CICS RESETBR DATASET('FILEA') RIDFLD(FLDA);
   GOTO BUILD;
  TOO_FAR:
16 MSG1A=DFHBMBRY;
   MSG2A=DFHBMDAR;
17 EXEC CICS SEND MAP('XDFHPMC') DATAONLY;
   GOTO REPEAT;
  NOTFOUND:
18 MESSAGES='INVALID NUMBER - PLEASE REENTER';
   GO TO PMNU;
  ERRORS:
19 EXEC CICS DUMP DUMPCODE('ERRS');
   MESSAGES='TRANSACTION TERMINATED';
  PMNU:
20 SUBSTR(ADDR(XDFHPMAO)->STRING,1,STG(XDFHPMAO)) = LOW(STG(XDFHPMAO));
   MSGA=DFHBMBRY;
   MSGO=MESSAGES;
21 EXEC CICS SEND MAP('XDFHPMA') ERASE;
   EXEC CICS RETURN;
  END;
```

PROGRAM NOTES

1. The error exits are set up.
2. The account number is mapped.
3. The exits for PF keys are set up.
4. The starting key is stored in field A in the key table.
5. Start of browse established.
6. The keys in the table are shifted right in anticipation of a continuation of a browse.
7. First record read into 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. Browsing command (CLEAR, PF1, or PF2 key, or 'F' or 'B') read from terminal, and control passed according to operator response (see note 3).
13. If end of file is reached on a READNEXT, records read to that point are displayed, together with message 'END OF FILE'. The label to which routine branches allows operator to restart browse at a different point. Bright attribute for page backward message is turned on.
14. If PF2 key is depressed or B typed, indicating page backward, and FLDC contains zeros, further backward paging not allowed; program branches to TOO-FAR (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. Table limit is exceeded, output map area is acquired, bright attribute for page forward message is turned on, and dark attribute is moved to page backward message.
17. Error message is sent to 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 XDFHPMA is displayed, and the program ends.

ORDER ENTRY SAMPLE PROGRAM (PL/I)

DESCRIPTION

The order entry sample application accepts input relating to the ordering of parts from a warehouse. When sufficient orders have been accumulated in the headquarters of a business, these are automatically sent off to a warehouse, or some other distribution point.

The program displays the map XDFHPMK on the screen requesting the operator to input details regarding the ordering of certain parts. The screen contains entry positions relating to the customer number, the part number and the quantity of that part required. (Any integer up to six digits in length may be entered: the customer number must be valid, that is, it must exist on FILEA.) When the screen has been filled, the operator presses CLEAR to stop entering data, and ENTER to continue entering data. The screen is then mapped in and the data is checked, errors being returned to the operator for reentering. When all the input is correct it is sent to a transient data queue called 'L860' - which is also a terminal-id where a transaction is to be triggered when the number of items on the queue reaches 30.

The trigger level may be changed using the CSMT command, as follows:

```
CSMT TRIGGER,n,DESTID=L860
```

where n is the destination trigger level (any integer from 0 through 32767).

SOURCE LISTING

```
 PORD:    PROC OPTIONS(MAIN);
          %INCLUDE XDFHPMK;
          %INCLUDE DFHBMSCA;
          %INCLUDE DFHAID;
          %INCLUDE L860;
          %INCLUDE FILEA;
          DCL BRTMDT CHAR(1) INIT('I');
          DCL ERROR_FLAG BIT(1);
          DCL WNG_MSG BIT(1) INIT('0'B);
          DCL NULL CHAR(1);
          DCL DUDCHAR CHAR(10) INIT('1234567890');
          DCL CHSTR CHAR(256) BASED;
    1     EXEC CICS HANDLE AID CLEAR(ENDA);
    2     EXEC CICS HANDLE CONDITION MAPFAIL(MAPFAIL) ERROR(ERRORS)
                                     NOTFND(NOTFOUND);
 STARTA:
          SUBSTR(ADDR(XDFHPMKO)->CHSTR,1,STG(XDFHPMKO))
                   =LOW(STG(XDFHPMKO));
 SEND:
    3     EXEC CICS SEND MAP('XDFHPMK') ERASE;
 RECEIVE:
    4     EXEC CICS RECEIVE MAP('XDFHPMK');
 TEST:
          ERROR_FLAG='0'B;
    5     CUSTNOA,PARTNOA,QUANTA=DFHBMFSE;
          IF VERIFY(CUSTNOI,'1234567890')¬=0 THEN DO;
             CUSTNOA=BRTMDT;
             ERROR_FLAG='1'B;
          END;
          IF VERIFY(PARTNOI,'1234567890')¬=0 THEN DO;
    6        PARTNOA=BRTMDT;
             ERROR_FLAG='1'B;
          END;
          IF VERIFY(QUANTI,'1234567890')¬=0 THEN DO;
             QUANTA=BRTMDT;
             ERROR_FLAG='1'B;
          END;
          IF ERROR_FLAG THEN DO;
             WNG_MSG='1'B;
    7        MSG2A=DFHBMASB;
             GO TO SEND;
           END;
    8     EXEC CICS READ DATASET('FILEA')
                         INTO(FILEA)
                         RIDFLD(CUSTNOI);
 Q_BUILD:
```

```
         CUSTNO=CUSTNOI;
         PARTNO=PARTNOI;
    9    QUANTITY=QUANTI;
         TERMID=EIBTRMID;
Q_WRITE:
   10    EXEC CICS WRITEQ TD QUEUE('L860')
                             FROM(L860)
                             LENGTH(22);
   11    IF WNG_MSG THEN DO;
            EXEC CICS SEND MAP('XDFHPMK') MAPONLY ERASE;
            WNG_MSG='0'B;
         END;
         ELSE
   12       EXEC CICS ISSUE ERASEAUP;
         GO TO RECEIVE;
NOTFOUND:
         WNG_MSG='1'B;
   13    MSG1A=DFHBMASB;
         GO TO SEND;
MAPFAIL:
         WNG_MSG='1'B;
         SUBSTR(ADDR(XDFHPMKO)->CHSTR,1,STG(XDFHPMKO))
                =LOW(STG(XDFHPMKO));
   14    MSG2A=DFHBMASB;
         GO TO SEND;
ERRORS:
   15    MSG2O='TRANSACTION TERMINATED';
         MSG2A=DFHBMBRY;
         EXEC CICS SEND MAP('XDFHPMK');
         EXEC CICS DUMP DUMPCODE('ERRS');
ENDA:
   16    EXEC CICS RETURN;
END;
```

PROGRAM NOTES

1. The exit for the clear key is set up.
2. The error exits are set up.
3. The screen is erased and the map is displayed at the terminal.
4. This command maps in the customer number, part number, and quantity.
5. The input areas on the map have the modified data tag set on in case the fields need to be sent back for reinput, should an error occur in entering the data.
6. The input is tested, and erroneous fields are brightened, whilst the modified data tag is still set on. The user should add further editing steps necessary to ensure only valid orders are accepted.
7. If there is a data error, the message 'DATA ERROR - REENTER', having been stored on the screen with a dark attribute character, is brightened.
8. The file control READ reads the record into a record area in order to find whether a particular record exists.
9. The input from the map is moved to the queue area.
10. The transient data WRITEQ obtains a log area, and writes this record to a sequential file.
11. If an error message is left on the screen, the screen is cleared and only the map is sent.
12. The entered fields, having been mapped in and processed, are erased, and the screen is ready to receive more input.
13. If the customer number entered was not found, the message 'NUMBER NOT FOUND - REENTER', having been stored on the screen with a dark attribute character, is brightened.
14. If no fields were entered, the message 'DATA ERROR - REENTER', also having been stored on the screen with a dark attribute character, is brightened.
15. On an error a dump is taken, and the message 'TRANSACTION TERMINATED' is moved to the top message area.
16. The program ends.

ORDER ENTRY QUEUE PRINT SAMPLE PROGRAM (PL/I)

DESCRIPTION

This transaction is invoked by entering the transaction-id 'PCOM' at the terminal. The program checks to see whether it was started from a terminal or the printer. If from a terminal, (that is, the operator is starting this transaction for the first time) the program starts the transaction at the printer in one hour. (This time interval could be changed using EDF for demonstration purposes.) The operator may then press RESET and CLEAR and enter another transaction. If from the printer, the program executes and starts again in one hour. If there are no items on the queue, a message indicating that the queue is empty, is sent to the warehouse. The last communications with the warehouse occurs not later than 3 'o' clock. This transaction is also started when the number of items on the queue 'L860' reaches 30.

The trigger level may be changed using the CSMT command, as follows:

```
CSMT TRIGGER,n,DESTID=L860
```

where n is the destination trigger level (any integer from 0 through 32767).

This program reads items off the queue 'L860', until the queue is empty. Should the queue have been empty initially, a message is sent to the warehouse. Using the number from the queue as a key it reads the file FILEA, and checks the amount field to see if the customer is good for credit on this order. If he is, the number,name,address,part number and quantity are moved to the map XDFHPML and this is sent to the printer. If he is not, the time,date,queue-item and a comment field are moved to a data area, this may be used for later processing. A message is then sent to the warehouse indicating that the queue is empty. The EIBTIME is then updated and if the time is before 1400 hours, the transaction is started in one hour.

SOURCE LISTING

```
PCOM:    PROC OPTIONS(MAIN);
         %INCLUDE FILEA;
         %INCLUDE L860;
         %INCLUDE XDFHPML;
         DCL Q_LENGTH FIXED BIN(15);
         DCL 1 LOGORD,
               2 LOGTIME,
                 3 LDATE FIXED DEC(7,0),
                 3 LTIME FIXED DEC(7,0),
               2 LITEM CHAR(22),
               2 COMMENT CHAR(11) INIT('ORDER ENTRY'),
               2 FILLER CHAR(51) INIT(' ');
         DCL CHSTR CHAR(256) BASED;
   1     EXEC CICS HANDLE CONDITION ERROR(ERRORS) QZERO(ENDA);
   2     IF EIBTRMID¬='L860' THEN
            GO TO TIME;
         SUBSTR(ADDR(XDFHPMLO)->CHSTR,1,STG(XDFHPMLO))
                  =LOW(STG(XDFHPMLO));
Q_READ:
         Q_LENGTH=22;
   3     EXEC CICS READQ TD INTO(L860) LENGTH(Q_LENGTH) QUEUE('L860');
MAP_BUILD:
   4     EXEC CICS READ DATASET('FILEA') INTO(FILEA) RIDFLD(CUSTNO);
   5     IF AMOUNT>'$0100.00' THEN DO;
            ADDRO=ADDRX;
            PARTO=PARTNO;
   6        NAMO=NAME;
            NUMBO=CUSTNO;
            QUANTO=QUANTITY;
   7        EXEC CICS SEND MAP('XDFHPML') ERASE PRINT L80;
            GO TO Q_READ;
         END;
         ELSE DO;
            LDATE=EIBDATE;
   8        LTIME=EIBTIME;
            LITEM=STRING(ITEM);
   9        EXEC CICS WRITEQ TD QUEUE('LOGA') FROM(LOGORD) LENGTH(92);
            GO TO Q_READ;
         END;
```

```
ERRORS:
  10     EXEC CICS DUMP DUMPCODE('ERRS');
         GO TO FIN;
ENDA:
  11     SUBSTR(ADDR(XDFHPMLO)->CHSTR,1,STG(XDFHPMLO))
                 =LOW(STG(XDFHPMLO));
         TITLEO='ORDER QUEUE IS EMPTY';
  12     EXEC CICS SEND MAP('XDFHPML') DATAONLY ERASE PRINT L80;
TIME:
  13     EXEC CICS ASKTIME;
  14     IF EIBTIME¬>140000 THEN
  15        EXEC CICS START TRANSID('PCOM') INTERVAL(10000)
                                           TERMID('L860');
FIN:
  16     EXEC CICS RETURN;
END;
```

PROGRAM NOTES

1. The error exits are set up.
2. The terminal-id is tested to see whether this transaction was started from a terminal or at the printer.
3. The queue item is read into the program.
4. The file control READ reads the record into a record area so that the amount may be checked.
5. The amount is tested.
6. If it is over $100, then the record on the queue is moved to the map XDFHPML. This test is only a suggestion; a suitable form of editing should be inserted to ensure valid orders are sent to the warehouse.
7. The map XDFHPML is sent to the printer.
8. If the order is not valid for this account, the record on the queue is moved to a data area, together with the terminal-id associated with the entering of this piece of data, the time, and date.
9. The transient data WRITEQ obtains a log area, and writes this record to a sequential file.
10. On an error (notes 3, 4, 7, 9, 12, and 15) a dump is taken.
11. When the queue is empty, a message is moved to the map area.
12. The map is displayed on the screen.
13. The current time-of-day clock is updated.
14. The current time-of-day is tested.
15. If the current time is not past 1400 hours, the transaction is started again in one hour, at the warehouse printer.
16. This routine ends.

REPORT SAMPLE PROGRAM (PL/I)

DESCRIPTION

The report sample program produces a report that lists all entries in the data set 'FILEA' for which the amount entry is less than or equal to $50.00.

The program illustrates page building techniques and the use of the terminal paging facilities of BMS.

The transaction is invoked by entering the transaction code PREP. The program does a sequential scan through the file noting each entry that obeys the search criterion. The pages are built from four maps which comprise mapset XDFHPMD, using the paging option so that the data is not displayed immediately but instead is stored for later retrieval. The HEADING map is inserted at the head of each page. The detail map (XDFHPMD) is written repeatedly until the overflow condition occurs. The FOOTING map is then written at the foot of the page and the HEADING map written at the top of the next page. The command to write the detail map that caused overflow is then repeated. When all the data has been written the FINAL map is written at the bottom of the last page and the transaction terminated.

The terminal operator then enters paging commands to display the data, clearing the screen before entering each paging command.

SOURCE LISTING

```
 REPORT: PROC OPTIONS(MAIN);
   DCL LOWLIM CHAR(8) INIT('$0050.00');
   DCL KEYNUM PIC'999999' INIT(0);
   DCL PAGEN PIC'999' INIT(1);
   DCL OPINSTR CHAR(22) STATIC INIT('ENTER PAGING COMMANDS.');
   DCL STRING CHAR(256) BASED;
   %INCLUDE XDFHPMD;
   %INCLUDE FILEA;
 1 EXEC CICS HANDLE CONDITION ERROR(ERRORS) OVERFLOW(OFLOW)
                              ENDFILE(ENDFILE);
 2 PAGENA=LOW(1);
   PAGENO=PAGEN;
 3 EXEC CICS SEND MAP('HEADING') MAPSET('XDFHPMD') ACCUM PAGING ERASE;
 4 KEYNUM=0;
 5 EXEC CICS STARTBR DATASET('FILEA') RIDFLD(KEYNUM);
  REPEAT:
 6 EXEC CICS READNEXT INTO(FILEA) DATASET('FILEA') RIDFLD(KEYNUM);
 7 IF STAT=HIGH(1) THEN GOTO REPEAT;
 8    IF AMOUNT<=LOWLIM THEN
      DO;
         SUBSTR(ADDR(XDFHPMDO)->STRING,1,STG(XDFHPMDO))=
                                                  LOW(STG(XDFHPMDO));
         AMOUNTO=AMOUNT;
 9       NUMBERO=NUMB;
         NAMEO=NAME;
         GOTO SEND_RECORD;
         OFLOW:
10       EXEC CICS SEND MAP('FOOTING') MAPSET('XDFHPMD')
                              MAPONLY ACCUM PAGING;
         PAGEN=PAGEN+1;
         PAGENA=LOW(1);
         PAGENO=PAGEN;
11       EXEC CICS SEND MAP('HEADING') MAPSET('XDFHPMD')
                        ACCUM PAGING ERASE;
         SEND_RECORD:
12        EXEC CICS SEND MAP('XDFHPMD') MAPSET('XDFHPMD') ACCUM PAGING;
    END;
   GOTO REPEAT;

  ENDFILE:
13 EXEC CICS SEND MAP('FINAL') MAPSET('XDFHPMD') MAPONLY ACCUM PAGING;
14 EXEC CICS SEND PAGE;
15 EXEC CICS SEND TEXT FROM(OPINSTR) ERASE;
16 EXEC CICS ENDBR DATASET('FILEA');
17 EXEC CICS RETURN;
  ERRORS:
```

```
18 EXEC CICS HANDLE CONDITION ERROR;
19 EXEC CICS PURGE MESSAGE;
20 EXEC CICS ABEND ABCODE('ERRS');
   END;
```

PROGRAM NOTES

1. The program exits are set up.
2. The attribute byte for the page number is cleared.
3. This BMS request sets up the heading in the page build operation.
4. The initial key value is set up for the START BROWSE command.
5. This command starts the browse through the file, at a record whose key is equal to or greater than that specified.
6. This command reads the next record on the file into the file area.
7. If the record is coded as deleted, it is treated as not found.
8. The search criterion for creating the report is that the customer has less than or equal to $50.
9. Fields are moved from the file area to the map area.
10. The BMS request sets up the footing in the page build operation.
11. The BMS request sets up the heading in the page build operation.
12. The customer detail map is set up.
13. When the END OF FILE condition is raised, the last map is built.
14. The page is sent to the terminal operator.
15. A message is sent to the terminal.
16. The BROWSE operation is ended.
17. The program ends.
18. On an error, the label to branch to on the ERROR condition is reset.
19. Any pages waiting to be displayed at the terminal are purged.
20. The program raises an abend condition, a dump is taken and the program ends.

XDFHPMA MAP DEFINITION

```
MAPSET   DFHMSD TYPE=&SYSPARM,MODE=INOUT,CTRL=(FREEKB,FRSET),LANG=PLI, *
               STORAGE=AUTO,EXTATT=MAPONLY,COLOR=BLUE
XDFHPMA  DFHMDI SIZE=(12,40)
         DFHMDF POS=(1,10),LENGTH=21,INITIAL='OPERATOR INSTRUCTIONS',  *
               HILIGHT=UNDERLINE
         DFHMDF POS=(3,1),LENGTH=29,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=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
```

XDFHPMA 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+
```

XDFHPMB 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
```

XDFHPMB SCREEN LAYOUT

```
          +XXXXXXXXXXXX

+NUMBER: +XXXXXX+
+NAME:   +XXXXXXXXXXXXXXXXXXX+
+ADDRESS:+XXXXXXXXXXXXXXXXXXX+
+PHONE:  +XXXXXXXX+
+DATE:   +XXXXXXXX+
+AMOUNT: +XXXXXXXX+
+COMMENT:+XXXXXXXXX+

+XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
+XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
```

XDFHPMC MAP DEFINITION

```
MAPSET    DFHMSD TYPE=&SYSPARM,MODE=INOUT,CTRL=(FREEKB,FRSET),LANG=PLI, *
               STORAGE=AUTO,EXTATT=MAPONLY
XDFHPMC   DFHMDI SIZE=(12,40)
DIR       DFHMDF POS=(1,1),LENGTH=1,ATTRB=IC
          DFHMDF POS=(1,3),LENGTH=1
          DFHMDF POS=(1,15),LENGTH=11,INITIAL='FILE BROWSE',           *
               HILIGHT=UNDERLINE,COLOR=BLUE
          DFHMDF POS=(3,1),LENGTH=6,INITIAL='NUMBER',COLOR=BLUE
          DFHMDF POS=(3,17),LENGTH=4,INITIAL='NAME',COLOR=BLUE
          DFHMDF POS=(3,32),LENGTH=6,INITIAL='AMOUNT',COLOR=BLUE
NUMBER1   DFHMDF POS=(4,1),LENGTH=6
NAME1     DFHMDF POS=(4,9),LENGTH=20
AMOUNT1   DFHMDF POS=(4,30),LENGTH=8
NUMBER2   DFHMDF POS=(5,1),LENGTH=6
NAME2     DFHMDF POS=(5,9),LENGTH=20
AMOUNT2   DFHMDF POS=(5,30),LENGTH=8
NUMBER3   DFHMDF POS=(6,1),LENGTH=6
NAME3     DFHMDF POS=(6,9),LENGTH=20
AMOUNT3   DFHMDF POS=(6,30),LENGTH=8
NUMBER4   DFHMDF POS=(7,1),LENGTH=6
NAME4     DFHMDF POS=(7,9),LENGTH=20
AMOUNT4   DFHMDF POS=(7,30),LENGTH=8
MSG1      DFHMDF POS=(11,1),LENGTH=39,                                 *
               INITIAL='PRESS PF1 OR TYPE F TO PAGE FORWARD'
MSG2      DFHMDF POS=(12,1),LENGTH=39,                                 *
               INITIAL='PRESS PF2 OR TYPE B TO PAGE BACKWARD'
          DFHMSD TYPE=FINAL
          END
```

XDFHPMC SCREEN LAYOUT

```
                +FILE BROWSE

+NUMBER           +NAME          +AMOUNT
+XXXXXX +XXXXXXXXXXXXXXXXXXXX+XXXXXX
+XXXXXX +XXXXXXXXXXXXXXXXXXXX+XXXXXX
+XXXXXX +XXXXXXXXXXXXXXXXXXXX+XXXXXX
+XXXXXX +XXXXXXXXXXXXXXXXXXXX+XXXXXX



+PRESS PF1 OR TYPE F TO PAGE FORWARD
+PRESS PF2 OR TYPE B TO PAGE BACKWARD
+XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
+XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
```

XDFHPMD MAP DEFINITION

```
MAPSET   DFHMSD TYPE=&SYSPARM,MODE=INOUT,CTRL=(FREEKB,FRSET),          *
               LANG=PLI,STORAGE=AUTO,EXTATT=MAPONLY,COLOR=BLUE
XDFHPMD  DFHMDI SIZE=(1,40),COLOR=GREEN
NUMBER   DFHMDF POS=(1,1),LENGTH=6
NAME     DFHMDF POS=(1,9),LENGTH=20
AMOUNT   DFHMDF POS=(1,30),LENGTH=8
HEADING  DFHMDI SIZE=(3,40),HEADER=YES
         DFHMDF POS=(1,5),LENGTH=18,INITIAL='LOW BALANCE REPORT',      *
               HILIGHT=UNDERLINE
         DFHMDF POS=(1,30),LENGTH=4,INITIAL='PAGE'
PAGEN    DFHMDF POS=(1,35),LENGTH=3
         DFHMDF POS=(3,1),LENGTH=6,INITIAL='NUMBER'
         DFHMDF POS=(3,17),LENGTH=4,INITIAL='NAME'
         DFHMDF POS=(3,32),LENGTH=6,INITIAL='AMOUNT'
FOOTING  DFHMDI SIZE=(2,40),TRAILER=YES,JUSTIFY=LAST
         DFHMDF POS=(2,10),LENGTH=25,                                  *
               INITIAL='CONTINUED ON NEXT PAGE...'
FINAL    DFHMDI SIZE=(2,40),TRAILER=YES,JUSTIFY=LAST
         DFHMDF POS=(2,10),LENGTH=14,INITIAL='END OF REPORT.'
         DFHMSD TYPE=FINAL
         END
```

XDFHPMD SCREEN LAYOUT

```
    LOW BALANCE REPORT        PAGE XXX

NUMBER           NAME             AMOUNT
XXXXXX XXXXXXXXXXXXXXXXXX   XXXXXXXX
XXXXXX XXXXXXXXXXXXXXXXXX   XXXXXXXX
     (REPEAT TOTAL OF 19 TIMES)
XXXXXX XXXXXXXXXXXXXXXXXX   XXXXXXXX
XXXXXX XXXXXXXXXXXXXXXXXX   XXXXXXXX
XXXXXX XXXXXXXXXXXXXXXXXX   XXXXXXXX

         CONTINUED ON NEXT PAGE...
```

XDFHPMK MAP DEFINITION

```
MAPSET   DFHMSD TYPE=&SYSPARM,MODE=INOUT,CTRL=(FREEKB),                *
               TIOAPFX=YES,LANG=PLI,STORAGE=AUTO,EXTATT=MAPONLY
XDFHPMK  DFHMDI SIZE=(12,40)
         DFHMDF POS=(01,10),LENGTH=11,ATTRB=(BRT,ASKIP),               *
               INITIAL='ORDER ENTRY',COLOR=BLUE,HILIGHT=UNDERLINE
MSG1     DFHMDF POS=(03,04),LENGTH=26,ATTRB=(DRK,ASKIP),               *
               INITIAL='NUMBER NOT FOUND - REENTER',COLOR=RED,         *
               HILIGHT=BLINK
MSG2     DFHMDF POS=(04,04),LENGTH=22,ATTRB=(DRK,ASKIP),               *
               INITIAL='DATA ERROR - REENTER',COLOR=RED,               *
               HILIGHT=BLINK
         DFHMDF POS=(05,04),LENGTH=09,ATTRB=PROT,                      *
               INITIAL='NUMBER  :'
CUSTNO   DFHMDF POS=(05,14),LENGTH=06,ATTRB=(IC,NUM)
         DFHMDF POS=(05,21),LENGTH=01
         DFHMDF POS=(06,04),LENGTH=09,ATTRB=PROT,COLOR=BLUE,           *
               INITIAL='PART NO :'
PARTNO   DFHMDF POS=(06,14),LENGTH=06,ATTRB=NUM
         DFHMDF POS=(06,21),LENGTH=01
         DFHMDF POS=(07,04),LENGTH=09,ATTRB=PROT,COLOR=BLUE,           *
               INITIAL='QUANTITY:'
QUANT    DFHMDF POS=(07,14),LENGTH=06,ATTRB=NUM
         DFHMDF POS=(07,21),LENGTH=01
         DFHMDF POS=(09,01),LENGTH=38,ATTRB=ASKIP,COLOR=BLUE,          *
               INITIAL='PRESS ENTER TO CONTINUE, CLEAR TO QUIT'
         DFHMSD TYPE=FINAL
         END
```

XDFHPMK SCREEN LAYOUT

```
        +ORDER ENTRY

   +NUMBER NOT FOUND - REENTER
   +DATA ERROR - REENTER
   +NUMBER  :+XXXXXX+
   +PART NO :+XXXXXX+
   +QUANTITY:+XXXXXX+

+PRESS ENTER TO CONTINUE, CLEAR TO QUIT
```

XDFHPML MAP DEFINITION

```
MAPSET   DFHMSD TYPE=&SYSPARM,MODE=OUT,                                *
               TIOAPFX=YES,LANG=PLI,STORAGE=AUTO
XDFHPML  DFHMDI SIZE=(05,80)
TITLE    DFHMDF POS=(01,01),LENGTH=43,                                 *
               INITIAL='NUMBER     NAME                      ADDRESS'
NUMB     DFHMDF POS=(02,01),LENGTH=06
NAM      DFHMDF POS=(02,12),LENGTH=20
ADDR     DFHMDF POS=(02,37),LENGTH=20
         DFHMDF POS=(03,01),LENGTH=09,                                 *
               INITIAL='PART NO :'
PART     DFHMDF POS=(03,11),LENGTH=06
         DFHMDF POS=(04,01),LENGTH=09,                                 *
               INITIAL='QUANTITY:'
QUANT    DFHMDF POS=(04,11),LENGTH=06
         DFHMDF POS=(05,01),LENGTH=1,                                  *
               INITIAL=' '
         DFHMSD TYPE=FINAL
         END
```

XDFHPML PRINT FORMAT

```
+NUMBER     NAME                      ADDRESS
+xxxxxx    +xxxxxxxxxxxxxxxxxxxx     +xxxxxxxxxxxxxxxxxxx
+PART NO:+xxxxxx
+QUANTITY:+xxxxxx
+x
```

ADDITIONS TO TABLES FOR PL/I SAMPLE PROGRAMS

PPT

The following entries were made for the sample maps:

```
DFHPPT TYPE=ENTRY,PROGRAM=XDFHPMA
DFHPPT TYPE=ENTRY,PROGRAM=XDFHPMB
DFHPPT TYPE=ENTRY,PROGRAM=XDFHPMC
DFHPPT TYPE=ENTRY,PROGRAM=XDFHPMD
DFHPPT TYPE=ENTRY,PROGRAM=XDFHPMK
DFHPPT TYPE=ENTRY,PROGRAM=XDFHPML
```

The following entries were made for the sample programs:

```
DFHPPT TYPE=ENTRY,PROGRAM=XDFHPMNU
       ,PGMLANG=PL/I
DFHPPT TYPE=ENTRY,PROGRAM=XDFHPALL
       ,PGMLANG=PL/I
DFHPPT TYPE=ENTRY,PROGRAM=XDFHPBRW
       ,PGMLANG=PL/I
DFHPPT TYPE=ENTRY,PROGRAM=XDFHPORD
       ,PGMLANG=PL/I
DFHPPT TYPE=ENTRY,PROGRAM=XDFHPCOM
       ,PGMLANG=PL/I
DFHPPT TYPE=ENTRY,PROGRAM=XDFHPREP
       ,PGMLANG=PL/I
```

PCT

The following entries were made for the sample programs:

```
DFHPCT TYPE=ENTRY,TRANSID=PMNU
       ,PROGRAM=XDFHPMNU
DFHPCT TYPE=ENTRY,TRANSID=PINQ
       ,PROGRAM=XDFHPALL
DFHPCT TYPE=ENTRY,TRANSID=PADD
       ,PROGRAM=XDFHPALL
DFHPCT TYPE=ENTRY,TRANSID=PUPD
       ,PROGRAM=XDFHPALL
DFHPCT TYPE=ENTRY,TRANSID=PBRW
       ,PROGRAM=XDFHPBRW
DFHPCT TYPE=ENTRY,TRANSID=PORD
       ,PROGRAM=XDFHPORD
DFHPCT TYPE=ENTRY,TRANSID=PCOM
       ,PROGRAM=XDFHPCOM
DFHPCT TYPE=ENTRY,TRANSID=PREP
       ,PROGRAM=XDFHPREP
```

DCT

The following entry was made:

```
DFHDCT TYPE=INTRA,DESTID=L860
       ,TRIGLEV=30,TRANSID=PCOM
       ,DESTFAC=TERMINAL
```

RECORD DESCRIPTIONS FOR THE PL/I SAMPLE PROGRAMS

FILEA RECORD DESCRIPTION

The FILEA record description is used by the sample programs and is of the following format:

```
DCL 1 FILEA,
      2 FILEREC,
        3 STAT CHAR(1),
        3 NUMB PIC'(6)9',
        3 NAME CHAR(20),
        3 ADDRX CHAR(20),
        3 PHONE CHAR(8),
        3 DATEX CHAR(8),
        3 AMOUNT CHAR(8),
        3 COMMENT CHAR(9);
```

LOGA RECORD DESCRIPTION

The LOGA record description is used by the sample programs when an audit trail is written to a transient data file. It has the following format:

```
DCL 1 LOGA,
      2 LOGHDR,
        3 LDAY FIXED DEC (7,0),
        3 LTIME FIXED DEC (7,0),
        3 LTERML CHAR(4),
      2 LOGREC,
        3 LSTAT CHAR(1),
        3 LNUMB CHAR(6),
        3 LNAME CHAR(20),
        3 LADDR CHAR(20),
        3 LPHONE CHAR(8),
        3 LDATE CHAR(8),
        3 LAMOUNT CHAR(8),
        3 LCOMMENT CHAR(9);
```

L860 RECORD DESCRIPTION

The L860 record description is used by the Order Entry Queue Print sample program when it writes to the transient data queue 'L860'. It has the following format:

```
DCL 1 L860,
      2 ITEM,
        3 CUSTNO CHAR(6),
        3 PARTNO CHAR(6),
        3 QUANTITY CHAR(6),
        3 TERMID CHAR(4);
```

Appendix G. Sample Programs for Distributed Transaction Processing

This appendix consists of sample CICS/VS application programs written in the assembler language illustrating distributed transaction processing between two CICS/VS systems, and between a CICS/VS system and an IMS/VS system.

Distributed transaction processing (DTP) allows transactions to be distributed to other CICS/VS (or similar) systems for processing. DTP provides the ability to distribute not only the resources, but also the processing of different parts of an application program, to the most appropriate places in the user network.

Commands are provided to allow a transaction in one system to initiate and converse with transactions in other systems in the network. Since one transaction exists before the conversation takes place, and since the other transaction is created by the conversation, there is a clear direction associated with each conversation.

The transaction that initiates the conversation is called the "front-end" transaction; the transaction created by the conversation is called the "back-end" transaction.

Transaction processing can be distributed between two systems irrespective of the number of sessions that exist between them. If more than one session exists, both user functions and CICS/VS functions can be distributed at the same time. However, if only a single session exists only one or the other type of function can be distributed.

A CICS/VS system can be connected either to another CICS/VS system, or to the IBM Information Management System (IMS/DC). These connections are made by logical unit type 6 (LU6) protocols.

Distributed transaction processing is controlled by the terminal control commands ALLOCATE, FREE, BUILD ATTACH, EXTRACT ATTACH, and POINT.

Further information about the SNA flows that take place between transactions is given in the CICS/VS System/Application Design Guide.

There are four sample programs, as follows:

- CICS to CICS Synchronous
- CICS to CICS (or other) Synchronous
- CICS to CICS Conversation (Synchronous)
- CICS to Other (Synchronous)

CICS TO CICS SYNCHRONOUS SAMPLE PROGRAM

DESCRIPTION

This sample is activated with the transaction code 'AIBL'.

The purpose of this first sample is to illustrate how an existing application program, which runs on a single system only, can be recoded so as to allow for part of the processing to be performed on a second remote system. The sample provided is based upon the existing File Browse sample application - transaction code ABRW - with which the user is assumed to be familiar and which is described in "Appendix D. Sample Programs (Assembler Language)" on page 251.

In this example, the first program handles only the interaction with the terminal operator; the file to be read is on the remote system and is accessed by a second transaction on that system. The operator enters the key only of the record at which browsing is to start, and this key, together with the name of the remote transaction is all that is sent across the link. The remote transaction now reads four file records and transmits them back to the originating system, where they are unblocked and moved to the output map.

As soon as it has transmitted four records, or sent an error or end-of-file message, the remote program terminates; the session must be freed, possibly to be reallocated again later, by the local program.

For simplicity, the use of the operator instruction menu - AMNU - has been avoided. The sample takes as input the transaction code plus data only.

The remote transaction name plus data is all that is passed across the link; no attach header is needed since, by default, CICS will assume that the first field in the data received represents the transaction name.

To illustrate the ease with which an existing application program can be 'converted' to handle Distributed Transaction Processing, comments are provided for the changed code only.

SOURCE LISTING OF LOCAL USER TRANSACTION

```
DFHEISTG DSECT
ATCHSESS DS    CL4
INLEN    DS    H
INAREA   DS    CL320
MESSAGES DS    CL80
OUTDATA  DS    0CL11
INDATA   DS    0CL14
TRAN     DS    CL5
KEYS     DS    0CL24
FLDA     DS    CL6
FLDB     DS    CL6
FLDC     DS    CL6
FLDD     DS    CL6
HEXZERO  DS    X'00'                CONSTANT FOR CLEARING MAPS
         COPY  XDFHAIB              DISPLAY MAP
         COPY  DFHXFILE             FILE RECORD DESCRIPTION
         COPY  DFHBMSCA             BMS ATTRIBUTE BYTES
XDFHAIBL CSECT
         MVI   KEYS,X'F0'           INSERT '0' INTO TOP BYTE OF KEYS
         MVC   KEYS+1(L'KEYS-1),KEYS  INITIALIZE KEYS TO ZERO
         MVI   MESSAGES,X'40'       INSERT ' ' INTO TOP BYTE OF MESSAGES
         MVC   MESSAGES+1(L'MESSAGES-1),MESSAGES  CLEAR MESSAGES FIELD
**************************************************************************
*                                                                        *
  1      EXEC CICS HANDLE CONDITION ERROR(ERRORS)MAPFAIL(AMNU)
         MVC   INLEN,=H'14'
  2      EXEC CICS RECEIVE INTO(INDATA) LENGTH(INLEN)
  3      EXEC CICS HANDLE AID CLEAR(QUIT)                               *
                PF1 (PAGEF) PF2 (PAGEB)
  4      MVC   TRAN,=CL5'AIBR'      REMOTE TRANSACTION NAME.
PAGEF    DS    0H
         MVC   INLEN,=H'326'         4 X 80 BYTE RECORDS PLUS FLDA.
  5      EXEC CICS ALLOCATE SYSID('REM1')
         MVC   ATCHSESS,EIBRSRCE
  6      EXEC CICS CONVERSE SESSION(ATCHSESS) FROMLENGTH(=H'11')        *
                FROM(OUTDATA) TOLENGTH(INLEN) INTO(INAREA)
         LA    5,INAREA
```

```
         MVC   FLDA,0(5)           LAST KEY READ
         LA    5,6(5)
  7      USING FILEA,5
  8      EXEC CICS FREE SESSION(ATCHSESS)
*                                                                      *
************************************************************************
         MVC   FLDD,FLDC
         MVC   FLDC,FLDB
         MVC   FLDB,FLDA
BUILD    DS    0H
         LA    4,1                 SET COUNTER TO 1
         LA    6,XDFHAMCO          R6->START OF MAP XDFHAMC
         LA    7,(XDFHAMCE-XDFHAMCO) R7 CONTAINS LENGTH OF XDFHAMC
         LA    8,HEXZERO           R8-> X'00'
         LA    9,L'HEXZERO         R9 CONTAINS LENGTH OF HEXZERO
         ICM   9,B'100',HEXZERO    X'00' INTO TOP BYTE OF R9
         MVCL  6,8                 MOVE X'00' INTO XDFHAMCO
************************************************************************
*                                                                      *
  9      LH    9,INLEN             PICK UP INPUT DATA LENGTH
NEXTLIN  DS    0H
  10     CH    9,=H'80'            < 80 BYTES RECEIVED ?
*                                                                      *
************************************************************************
         BL    NOTFOUND            ..YES, ERROR MESSAGE.
         CH    4,=H'1'             FIRST LINE?
         BNE   SECLIN              ..NO, GO TEST FOR 2ND LINE
         MVC   NUMBER1O,NUMB       MOVE NUMBER TO MAP AREA
         MVC   NAME1O,NAME         MOVE NAME TO MAP AREA
         MVC   AMOUNT1O,AMOUNT     MOVE AMOUNT TO MAP AREA
         B     CONT                GO CONTINUE
SECLIN   CH    4,=H'2'             SECOND LINE?
         BNE   THRLIN              ..NO, GO TEST FOR THIRD LINE
         MVC   NUMBER2O,NUMB       MOVE NUMBER TO MAP AREA
         MVC   NAME2O,NAME         MOVE NAME TO MAP AREA
         MVC   AMOUNT2O,AMOUNT     MOVE AMOUNT TO MAP AREA
         B     CONT                GO CONTINUE
THRLIN   CH    4,=H'3'             THIRD LINE?
         BNE   FORLIN              ..NO, GO TEST FOR FOURTH LINE
         MVC   NUMBER3O,NUMB       MOVE NUMBER TO MAP AREA
         MVC   NAME3O,NAME         MOVE NAME TO MAP AREA
         MVC   AMOUNT3O,AMOUNT     MOVE AMOUNT TO MAP AREA
         B     CONT                GO CONTINUE
FORLIN   CH    4,=H'4'             FOURTH LINE?
         BNE   CONT                ..NO, CONTINUE
         MVC   NUMBER4O,NUMB       MOVE NUMBER TO MAP AREA
         MVC   NAME4O,NAME         MOVE NAME TO MAP AREA
         MVC   AMOUNT4O,AMOUNT     MOVE AMOUNT TO MAP AREA
CONT     DS    0H
         LA    4,1(,4)             INCREMENT COUNT
************************************************************************
*                                                                      *
 11      LA    5,80(,5)            INCREMENT RECORD POINTER
         SH    9,=H'80'            REDUCE LENGTH.
*                                                                      *
************************************************************************
         CH    4,=H'5'             FINISHED?
         BNE   NEXTLIN             ..NO, GO BUILD NEXT LINE
DISPREC  DS    0H                  ..YES, SEND MAP
         EXEC CICS SEND MAP('XDFHAMC') ERASE
REPEAT   DS    0H
         EXEC CICS RECEIVE MAP('XDFHAMC')
         CLI   DIRI,C'F'           PAGE FORWARD REQUIRED?
         BE    PAGEF               ..YES, GO TO PAGEFORWARD ROUTINE
         CLI   DIRI,C'B'           PAGE BACK REQUIRED?
         BE    PAGEB               ..YES, GO TO PAGEBACKWARD ROUTINE
         BNE   AMNU                ..NO, GO SEND MENU MAP
PAGEB    DS    0H                  PAGEFORWARD ROUTINE
         CLC   FLDC(6),=C'000000'  FLDC = ZEROS?
         BE    TOOFAR              ..YES, SO RAISE TOO FAR CONDITION
         MVC   FLDA,FLDC           ..NO, SET UP KEYS FOR FILE
         MVC   FLDB,FLDC
         MVC   FLDC,FLDD
```

```
         MVC   FLDD,=C'000000'
         B     BUILD               GO BUILD MAP
TOOFAR   DS    0H                  GONE TOO FAR
         MVI   MSG1A,DFHBMBRY      BRIGHTEN MESSAGE
         MVI   MSG2A,DFHBMDAR      DARKEN MESSAGE
         EXEC CICS SEND MAP('XDFHAMC') DATAONLY
         B     REPEAT              GO GET MAP
NOTFOUND DS    0H
*****************************************************************
*                                                               *
         BCTR  9,0
 12      EX    9,MOVEMSG           MOVE MESSAGE TO MAP.
*                                                               *
*****************************************************************
         B     AMNU
MOVEMSG  MVC   MESSAGES(0),0(5)
ERRORS   DS    0H                  GENERAL ERROR ROUTINE
         EXEC CICS DUMP DUMPCODE('ERRS')
         MVC   MESSAGES,=CL(L'MESSAGES)'TRANSACTION TERMINATED'
AMNU     DS    0H                  END ROUTINE
*****************************************************************
*                                                               *
         MVI   MSG1A,DFHBMBRY      BRIGHTEN MESSAGE FIELD
         MVC   MSG1O,MESSAGES      MOVE MESSAGES TO MAP AREA
 13      EXEC CICS SEND MAP('XDFHAMC') ERASE
*                                                               *
*****************************************************************
QUIT     DS    0H
         EXEC CICS RETURN
         END
```

PROGRAM NOTES

1. The exit addresses to handle End-of-File and Not-Found are moved to the remote program.

2. The local program reads in the transaction code followed by the key at which the browsing is to be started. The read is from an unformatted screen.

3. The action when the operator presses the CLEAR key has been changed to cause the program to exit immediately.

4. The name of the remote transaction which performs the file reading is moved into the data area which will be shipped across the link.

5. A session must now be established with the remote system. (The name of the remote system is here assumed to be 'REM1'; this name is installation dependent, and will be assigned by the system programmer). The session is created by issuing the ALLOCATE command specifying only the remote system name; the name of the allocated session is required by the application program only in so far as it is used in all subsequent SEND/RECEIVE commands, and it can be found in field EIBRSRCE immediately after completion of the ALLOCATE.

6. A CONVERSE command - which comprises both a SEND and RECEIVE - is issued to transmit the remote transaction name and key to the system here defined as 'REM1'. (This name is of course installation dependent, and will be assigned by the system programmer). Otherwise, the command options are unchanged from their usage in CICS/VS Release 1.4. It should also be noted that the CONVERSE command utilizes the INTO option as opposed to SET; this is essential because the FREE SYSID command which follows causes all storage associated with the session, for example the TIOA, to be released immediately and thus become unavailable to the user program. With the INTO option being coded, the received data is moved into user storage where it remains under the control of the program. If the FREE SYSID were to be omitted, either INTO or SET could be used; the latter would normally be preferable for improved performance.

7. Addressability for the file records which will be returned to this program is provided using the file record descriptor FILEA. Note that the first six bytes of the received data contain the key of the last record read by the remote program.

8. A session to the remote system 'REM1' will have been allocated automatically by CICS/VS when the CONVERSE command was executed. This session is now freed to enable it to be reused by another application. Execution of a further CONVERSE will cause a session to be reestablished.

If it is desired to maintain the session until the application terminates, the FREE command should be omitted entirely, but in this case the remote program must itself keep the session open by issuing a CONVERSE in place of its SEND and RECEIVE commands.

9. The total length of the returned records is held in INLEN.

10. If the length of the buffer remaining is less than 80 bytes long, assume a message, rather than a file record, has been received.

11. The buffer pointer is incremented to address the next record and the length remaining value reduced.

12. Whatever warning message has been sent by the remote program is now transferred to the BMS map.

13. The message will appear on the file record detail display rather than the operator instruction menu.

SOURCE LISTING OF REMOTE USER TRANSACTION

```
DFHEISTG DSECT
INLEN    DS    H
OUTLENG  DS    H
INPUT    DS    0CL11
TRAN     DS    CL5
FLDA     DS    CL6
OUTBUF   DS    CL320
         COPY  DFHXFILE            FILE RECORD DESCRIPTION
XDFHAIBR CSECT
         EXEC CICS HANDLE CONDITION ENDFILE(ENDFILE) NOTFND(NOTFOUND)
*****************************************************************
*                                                               *
         MVC   INLEN,=H'11'        LENGTH OF INPUT DATA.
  1      EXEC CICS RECEIVE INTO(INPUT) LENGTH(INLEN)
*****************************************************************
         EXEC CICS STARTBR DATASET('FILEA') RIDFLD(FLDA)
SET4     DS    0H
         LA    4,1                 SET COUNTER TO 1
*****************************************************************
*                                                               *
         LA    5,OUTBUF            ADDRESS OUTPUT BUFFER.
         USING FILEA,5
NEXTLIN  DS    0H
  2      EXEC CICS READNEXT INTO(FILEA) DATASET('FILEA') RIDFLD(FLDA)
*****************************************************************
         CLI   STAT,X'FF'          IS RECORD CODED AS NOT FOUND
         BE    NEXTLIN             ..YES, SET UP NEXT LINE
         LA    4,1(,4)             INCREMENT COUNT
         LA    5,80(,5)            INCREMENT BUFFER POINTER
         CH    4,=H'5'             FINISHED?
         BNE   NEXTLIN             ..NO, GO BUILD NEXT LINE
DISPREC  DS    0H                  ..YES, SEND MAP
*****************************************************************
*                                                               *
  3      LA    4,OUTBUF
         SR    5,4                 NO. OF BYTES TO BE SENT
         STH   5,OUTLENG
*****************************************************************
         B     RETURN
ENDFILE  DS    0H                  ENDFILE IS REACHED
*****************************************************************
  4      MVC   0(11,5),=CL11'END OF FILE' MESSAGE
         LA    5,11(,5)
         B     DISPREC             GO SEND MAP
NOTFOUND DS    0H
         MVC   0(29,5),=CL29'INVALID NUMBER-PLEASE REENTER'
         LA    5,29(,5)
*****************************************************************
         B     DISPREC
RETURN   DS    0H
  5      EXEC CICS SEND FROM(FLDA) LENGTH(OUTLENG) WAIT LAST
         EXEC CICS RETURN
         END
```

PROGRAM NOTES

1. The transaction name and record key are received.

2. A record is read direct into the output buffer for transmission.

3. The length of the data to be sent is now calculated.

4. Any warning messages needed are sent to the output buffer and the length of the data is increased accordingly.

5. The records are returned across the link and the program ends, thereby automatically freeing its session. Should the originating sample be amended to cause repeated invocation of this program, these last two instructions could be replaced by, for example:

```
EXEC CICS CONVERSE FROM(OUTBUF)
     FROMLENGTH(OUTLENG)
     INTO(INPUT) TOLENGTH(INLEN)
B    SET4
```

CICS TO CICS (OR OTHER) SYNCHRONOUS SAMPLE PROGRAM

DESCRIPTION

This sample is activated with the transaction code 'AICC'.

The CICS to CICS synchronous sample application program allows a terminal operator to enter a command on the screen and have that command transmitted to a remote system for execution. If necessary, the remote system responds with a request for further details, and the operator is given the opportunity of replying.

The program is able to converse with any application on a remote system which sends output data either one line at a time or in multiple line format. The CICS supplied programs listed below have this capability, thus the main purpose of this example is to provide for the CICS system programmer a simple test transaction which will enable him to prove easily that he is able to establish contact with a second, remote CICS system without the need for any application program coding on his part. A successful test of this sample will indicate, to the extent of the features actually being tested, that the system network has been correctly set up and that the Inter-System components of CICS to allow distributed transaction processing are in order; failure will indicate errors in set up rather than in user programming.

At the start of the program, the operator is prompted to enter the name of the remote system to be attached, and the actual command to be executed on the remote system which is entered just as if it were a local command, for example, CSMT TAS. The program is able to handle both single line output from the remote system and also output which exceeds the terminal page size.

The message received from the remote system is assumed to be in SCS form, that is, containing printable characters and new line symbols only. This is the default output format for LU6 type terminals as produced by CICS supplied programs such as CSFE, CSMT, CSOT, CSST, or CSTT.

SOURCE LISTING OF THE SENDING USER TRANSACTION

```
DFHEISTG DSECT
*
*         STORAGE AREA FOR EIB SESSION AND STATUS FLAGS
*
XDFEIFLG DS    0CL7
DFHSYNC  DS    C                          IF SET, SYNCHPOINT MUST
*                                         BE EXECUTED
DFHFREE  DS    C                          IF SET, TERMINAL / LU
*                                         MUST BE FREED
DFHRECV  DS    C                          IF SET, RECEIVE MUST
*                                         BE EXECUTED
DFHSEND  DS    C                          RESERVED
DFHATT   DS    C                          IF SET, ATTACH HEADER
*                                         DATA EXISTS AND MAY BE
*                                         ACCESSED USING EXTRACT
DFHEOC   DS    C                          IF SET, END-OF-CHAIN
*                                         WAS RECEIVED WITH DATA
DFHFMH   DS    C                          IF SET, DATA PASSED TO
*                                         APPL'N CONTAINS FMH(S)
         COPY  XDFHAI1                      COPY MAP
*
REMDATA  DS    256D
ATCHSESS DS    CL4
CONTROL  DS    0CL60
SBA      DS    CL3
CDATA    DS    CL57
MESSAGE  DS    CL32
INLEN    DS    H
OUTLEN   DS    H
NEWLINE  EQU   X'15'
         DFHEJECT
XDFHAI1A CSECT
  1      EXEC CICS HANDLE CONDITION MAPFAIL(MAPFAIL)
         EXEC CICS HANDLE AID CLEAR(CLEAR)
MAPFAIL  XC    MAPAI(MAPAE-MAPAI),MAPAI    CLEAR MAP
  2      EXEC CICS SEND MAP('MAPA') MAPSET('XDFHAI1')                  *
               ERASE MAPONLY WAIT
```

```
  3       EXEC CICS RECEIVE MAP('MAPA') MAPSET('XDFHAI1')
          LA    8,DATAI
          MVC   DATABL(3+L'DATABO),DATAL
          MVC   OUTLEN,DATAL
  4       EXEC CICS HANDLE CONDITION SYSIDERR(SYSERR)
*
  5       EXEC CICS SEND MAP('MAPB') MAPSET('XDFHAI1') WAIT ERASE
  6       EXEC CICS ALLOCATE SYSID(SYSIDI)
          MVC   ATCHSESS,EIBRSRCE
CONVERSE  DS    0H
          MVC   INLEN,=H'2048'
  7       EXEC CICS CONVERSE                                          *
                SESSION(ATCHSESS)                                     *
                FROM(0(8))                                            *
                FROMLENGTH(OUTLEN)                                    *
                INTO(REMDATA)                                         *
                TOLENGTH(INLEN)
          MVC   XDFEIFLG,EIBSYNC        SAVE EIB VALUES
DATASENT  DS      0H
  8       CLC   INLEN,=H'0'             IF NULL RU SENT THEN
          BE    TESTSYNC                NOTHING TO SEND TO TERMINAL
*
          LH    1,INLEN
          LA    2,REMDATA(1)            ADDR BYTE AFTER DATA
          MVI   0(2),X'13'              INSERT CURSOR HERE
          LA    1,1(,1)
          STH   1,INLEN
          EXEC CICS SEND TEXT FROM(REMDATA) LENGTH(INLEN) ACCUM
TESTSYNC  DS    0H
  9       CLI   DFHSYNC,X'FF'
          BNE   TESTFREE
          EXEC CICS SYNCPOINT
TESTFREE  DS    0H
 10       CLI   DFHFREE,X'FF'
          BNE   TESTRECV
          EXEC CICS SEND PAGE RETAIN
          EXEC CICS RETURN
TESTRECV  DS    0H
 11       CLI   DFHRECV,X'FF'
          BNE   SEND
          MVC   INLEN,=H'2048'
          EXEC CICS RECEIVE SESSION('ATCHSESS') INTO(REMDATA)         *
                LENGTH(INLEN)
          MVC   XDFEIFLG,EIBSYNC        SAVE EIB VALUES
          B     DATASENT
SEND      DS    0H
 12       EXEC CICS SEND PAGE RETAIN
          MVC   OUTLEN,=H'60'
          EXEC CICS RECEIVE INTO(CONTROL) LENGTH(OUTLEN)
          LH    0,OUTLEN
          SH    0,=H'3'                 REDUCE FOR LENGTH OF SBA
          LA    8,CDATA
          B     CONVERSE
SYSERR    DS    0H
 13       CLI   EIBRCODE+1,12
          BE    UNKNOWN
          CLI   EIBRCODE+1,4
          BE    NOTCTSE
NOLINK    DS    0H
 14       MVC   MESSAGE,LINKMSG
          MVC   MESSAGE+28(4).SYSIDI
          B     EXPLAIN
LINKMSG   DC    CL32'UNABLE TO ESTABLISH LINK TO     '
UNKNOWN   DS    0H
 15       MVC   MESSAGE,UNKMSG
          MVC   MESSAGE+12(4),SYSIDI
          B     EXPLAIN
UNKMSG    DC    CL32'SYSTEM NAME      IS NOT KNOWN   '
NOTCTSE   DS    0H
 16       MVC   MESSAGE,TCTMSG
          MVC   MESSAGE(4),SYSIDI
          B     EXPLAIN
TCTMSG    DC    CL32'     IS NOT A SYSTEM NAME'
```

```
EXPLAIN  DS    0H
         EXEC CICS SEND FROM(MESSAGE) LENGTH(=H'32') ERASE WAIT
CLEAR    DS    0H
         END
```

PROGRAM NOTES

1. Set up exit for map errors and clear key.

2. The screen is erased, and the prompting map displayed at the terminal.

3. The remote system name and command to be transmitted are mapped in.

4. Set up exit for the error conditions which may arise whilst establishing connection to the remote system.

5. The screen is erased again and the command entered by the operator is displayed on the top line.

6. A session is now allocated naming the remote system only, and its name is obtained from EIBRSRCE.

7. A CONVERSE command is now issued which sends the data entered by the terminal operator to the remote system which he has specified, then receives the resulting response from that system. To enable the program to determine what action is next expected of it, the contents of the EXEC Interface Block will have to be examined, thus the values therein must be retained. The SYSID option is used since the application is requesting that an alternate facility be made available to it. Note that, although it is permissible to build an attach header and transmit it using the CONVERSE command, this action does not need to be taken in this case since by default CICS/VS will assume that the first four characters of the transmitted data contain the transaction code.

8. If the data length field for the RECEIVE component of the CONVERSE indicates that there is data to be handled, a logical message is built using the BMS TEXT facility for subsequent sending to the screen. To ensure that the terminal cursor is placed on the next available line for any further input, the 'Insert Cursor' control character is appended to the data stream.

9. The session-oriented information transmitted across the LU6 session by the remote transaction must now be examined to determine what action should be taken next. The 'SYNCPOINT required' indicator in the EXEC Interface Block is first tested and if need be the program issues its own SYNCPOINT.

10. If the EXEC Interface Block indicates that the program should now free the session, thereby denoting that the remote transaction has completed successfully and has terminated the conversation, the built logical message is sent to the screen using the RELEASE option of the SEND PAGE command which returns control direct to CICS/VS and thus frees the session.

11. If the EXEC Interface Block indicates that the application is to continue receiving data from across the session, a further RECEIVE command is issued.

12. The indicators SYNCPOINT, FREE session, or RECEIVE, do not apply, thus by default the remote application has requested a further transmission from this program. (In the case of the CICS/VS supplied programs named in the description above this would imply the receipt of a prompting message.) The program therefore sends the logical message built to date, which will include the prompt, to the terminal operator and receives his reply; a second CONVERSE can then be issued across the session. Note that the 'Set Buffer Address' control and the two buffer address bytes received from the terminal must be bypassed before transmission across the link.

13. The SYSID error routine has been entered. To determine the exact cause of the error, EIBRCODE must be examined, and an appropriate informatory message sent to the operator.

14. Some kind of error exists which prevents the link between the two systems from being established.

15. The remote system name given by the operator is not recognized.

16. The system name given is recognized, but is not that for a remote system.

MAP DEFINITION

```
XDFHAI1  DFHMSD TYPE=&SYSPARM,MODE=INOUT,CTRL=FREEKB,TIOAPFX=YES,      *
               LANG=ASM,STORAGE=AUTO
MAPA     DFHMDI SIZE=(12,80)
         DFHMDF POS=(1,1),ATTRB=ASKIP,LENGTH=33,                       *
               INITIAL='TYPE REMOTE SYSTEM ID AND COMMAND'
         DFHMDF POS=(3,1),ATTRB=ASKIP,LENGTH=16,                       *
               INITIAL='REMOTE SYSTEM ID'
SYSID    DFHMDF POS=(3,20),ATTRB=(NORM,IC),LENGTH=4,INITIAL='    '
         DFHMDF POS=(3,25),LENGTH=1
         DFHMDF POS=(4,1),ATTRB=ASKIP,LENGTH=07,                       *
               INITIAL='COMMAND'
DATA     DFHMDF POS=(4,10),ATTRB=(NORM),LENGTH=68,INITIAL=' '
         DFHMDF POS=(6,1),ATTRB=ASKIP,LENGTH=16,                       *
               INITIAL='THEN PRESS ENTER'
MAPB     DFHMDI SIZE=(12,80)
DATAB    DFHMDF POS=(1,1),ATTRB=(NORM),LENGTH=68,INITIAL=' '
         DFHMSD TYPE=FINAL
         END
```

SCREEN LAYOUT

```
TYPE REMOTE SYSTEM ID AND COMMAND

REMOTE SYSTEM ID    XXXX
COMMAND   XXXXXXXXXXXXXXXXXXXXXXXXXXX

THEN PRESS ENTER
```

CICS TO CICS CONVERSATION (SYNCHRONOUS) SAMPLE PROGRAM

DESCRIPTION

This sample is activated with the transaction code 'AISC'.

The sample will consist of a two-part program illustrating the facility of transferring the contents of a temporary storage queue from a local CICS system to another remote CICS system. The corresponding transaction name to be used on the remote system is 'AISR'.

To test the sample, a temporary storage queue may first be created on the local system. The operator may then refer to this queue by name or alternatively, if no name is given, a small 5-record queue will be built by the program for basic test purposes. The terminal operator at the local system is given the opportunity of specifying details of the queue name, and of the remote system including the queue name that the data set is to be given on the remote system, thus the sample is general purpose in nature.

When the transfer of the queue starts, the terminal operator is notified accordingly.

SOURCE LISTING OF USER TRANSACTION

```
DFHEISTG DSECT
*
*         STORAGE AREA FOR EIB SESSION AND STATUS FLAGS
*
XDFEIFLG DS    0CL7
*
DFHSYNC  DS    C                        IF SET, SYNCHPOINT MUST
*                                       BE EXECUTED
DFHFREE  DS    C                        IF SET, TERMINAL / LU
*                                       MUST BE FREED
DFHRECV  DS    C                        IF SET, RECEIVE MUST
*                                       BE EXECUTED
DFHSEND  DS    C                        RESERVED
DFHATT   DS    C                        IF SET, ATTACH HEADER
*                                       DATA EXISTS AND MAY BE
*                                       ACCESSED USING EXTRACT
DFHEOC   DS    C                        IF SET, END-OF-CHAIN
*                                       WAS RECEIVED WITH DATA
DFHFMH   DS    C                        IF SET, DATA PASSED TO
*                                       APPL'N CONTAINS FMH(S)
         COPY  XDFHAI2
RECCOUNT DS    CL3
INLEN    DS    H
QNAME    DS    CL8
ATCHSESS DS    CL4
MESSAGE  DS    CL32
R4       EQU   4
R5       EQU   5
R8       EQU   8
R9       EQU   9
         DFHEJECT
XDFHAI2A CSECT
  1      CLC   EIBTRNID,=C'AISR'   RECEIVING TRANSACTION ?
         BE    INBFMH              YES - BRANCH TO HANDLE
  2      EXEC CICS HANDLE AID CLEAR(CLEAR)
         EXEC CICS HANDLE CONDITION MAPFAIL(MAPFAIL)
MAPFAIL  XC    MAPAI(MAPAE-MAPAI),MAPAI  CLEAR MAP
  3      EXEC CICS SEND MAP('MAPA') MAPSET('XDFHAI2') ERASE MAPONLY    *
               WAIT
  4      EXEC CICS RECEIVE MAP('MAPA') MAPSET('XDFHAI2')
  5      CLI   LOCALQI,0           IF TS Q IS NAMED
         BNE   QNAMED              WE DO NOT CREATE ONE.
         MVC   LOCALQI,=CL8'TSQAISC'
         LA    R4,REC1             ADDRESS 1ST RECORD
         LA    R5,R5               LOOP COUNT
QLOOP    DS    0H
         EXEC CICS WRITEQ TS QUEUE(LOCALQI) FROM(0(R4)) LENGTH(RLEN)
         LA    R4,L'REC1(R4)       ADDRESS NEXT RECORD
         BCT   R5,QLOOP            AND WRITE IT.
QNAMED   DS    0H
```

```
  6       EXEC CICS HANDLE CONDITION ERROR(ABEND) SYSIDERR(SYSERR)
  7       EXEC CICS READQ TS QUEUE(LOCALQI) SET(R9) LENGTH(INLEN)
  8       CLI   REMOTQI,0           REMOTE Q NAME SPECIFIED ?
          BNE   REMQOK              ..YES
          MVC   REMOTQI,=CL8'TSQAISR'  ..NO, GIVE A DEFAULT
REMQOK    DS    0H
  9       EXEC CICS ALLOCATE SYSID(SYSIDI)
          MVC   ATCHSESS,EIBRSRCE
 10       EXEC CICS BUILD ATTACH                                        *
                ATTACHID('REMQ') PROCESS('AISR') QUEUE(REMOTQI)
 11       EXEC CICS SEND SESSION(ATCHSESS) FROM(0(R9))  LENGTH(INLEN)   *
                ATTACHID('REMQ') WAIT
 12       MVC   MTSQ0,LOCALQI       SET UP LOCAL TS Q NAME
          MVC   MSYSID0,SYSIDI      SET UP REMOTE SYSTEM ID
          EXEC CICS SEND MAP('MAPB') MAPSET('XDFHAI2')                  *
                WAIT CURSOR(=H'80') ERASE
 13       ZAP   RECCOUNT,=P'1'      UPDATE RECORD COUNT
 14       EXEC CICS HANDLE CONDITION ITEMERR(QEND)
LOOP      DS    0H
 15       EXEC CICS READQ TS QUEUE(LOCALQI) SET(R9) LENGTH(INLEN)
          EXEC CICS SEND SESSION(ATCHSESS) FROM(0(R9)) LENGTH(INLEN) WAIT
          AP    RECCOUNT,=P'1'      UPDATE RECORD COUNT
          B     LOOP                START READQ/SEND LOOP AGAIN
QEND      DS    0H
 16       UNPK  COUNT0,RECCOUNT
          OI    COUNT0+5,C'0'
          EXEC CICS SEND MAP('MAPC') MAPSET('XDFHAI2') WAIT
 17       EXEC CICS DELETEQ TS QUEUE(LOCALQI)
 18       EXEC CICS SYNCPOINT
          B     CLEAR
*
ABEND     DS    0H
 19       EXEC CICS ASSIGN ABCODE(ABCODE0)
          MVC   RSOURC0,EIBRSRCE
 20       EXEC CICS SEND MAP('MAPD') MAPSET('XDFHAI2') ERASE WAIT
          B     CLEAR
*
SYSERR    DS    0H
 21       CLI   EIBRCODE+1,12
          BE    UNKNOWN
          CLI   EIBRCODE+1,4
          BE    NOTCTSE
NOLINK    DS    0H
 22       MVC   MESSAGE,LINKMSG
          MVC   MESSAGE+28(4),SYSIDI
          B     EXPLAIN
LINKMSG   DC    CL32'UNABLE TO ESTABLISH LINK TO     '
*
UNKNOWN   DS    0H
 23       MVC   MESSAGE,UNKMSG
          MVC   MESSAGE+12(4),SYSIDI
          B     EXPLAIN
UNKMSG    DC    CL32'SYSTEM NAME      IS NOT KNOWN   '
*
NOTCTSE   DS    0H
 24       MVC   MESSAGE,TCTMSG
          MVC   MESSAGE(4),SYSIDI
          B     EXPLAIN
TCTMSG    DC    CL32'     IS NOT A SYSTEM NAME'
*
EXPLAIN   DS    0H
          EXEC CICS SEND FROM(MESSAGE) LENGTH(=H'32') ERASE WAIT
          B     CLEAR
*
REC1      DC    C'DISTRIBUTED TRANSACTION PROCESSING TS Q - RECORD1'
          DC    C'DISTRIBUTED TRANSACTION PROCESSING TS Q - RECORD2'
          DC    C'DISTRIBUTED TRANSACTION PROCESSING TS Q - RECORD3'
          DC    C'DISTRIBUTED TRANSACTION PROCESSING TS Q - RECORD4'
          DC    C'DISTRIBUTED TRANSACTION PROCESSING TS Q - RECORD5'
          DS    0H
RLEN      DC    AL2(L'REC1)
          DFHEJECT
INBFMH    DS    0H                      CODE FOR 'RECEIVING' TRANSACTION
```

```
 25       EXEC CICS RECEIVE SET(9) LENGTH(INLEN)
          MVC   XDFEIFLG,EIBSYNC        SAVE EIB VALUES
 26       EXEC CICS EXTRACT ATTACH QUEUE(QNAME)
INBOUND   DS    0H
 27       CLC   INLEN,=H'0'             IF NO DATA SENT THEN
          BE    TESTSYNC                NOTHING TO WRITE
          EXEC CICS WRITEQ TS QUEUE(QNAME) FROM(0(9)) LENGTH(INLEN)
TESTSYNC  DS    0H
 28       CLI   DFHSYNC,X'FF'
          BNE   TESTFREE
          EXEC CICS SYNCPOINT
TESTFREE  DS    0H
 29       CLI   DFHFREE,X'FF'
          BE    CLEAR
     .    CLI   DFHRECV,X'FF'
          BNE   CLEAR
          EXEC CICS RECEIVE SET(9) LENGTH(INLEN)
          MVC   XDFEIFLG,EIBSYNC        SAVE EIB VALUES
          B     INBOUND
CLEAR     DS    0H
          END
```

PROGRAM NOTES

1. This single program contains both the sending and receiving code and copies should be used simultaneously at both ends of the link. This first test is to determine which of the two functions is being executed.

2. Set up exit for clear key and to resend map on failure.

3. The screen is erased, and the prompting map displayed at the terminal.

4. The remote system name and both local and remote temporary storage queue names are mapped in.

5. The sending program normally expects the user to notify the name of a local temporary storage queue to be transferred. For testing purposes, however, the user may omit the queue name in which case a small, 5-record queue will be built for him and named 'TSQAISC'.

6. Set up exit for error condition which may arise whilst establishing connection to remote system, and for temporary storage queue errors.

7. The first record on the local temporary storage queue is now read.

8. If the remote queue name is omitted, a default of 'TSQAISR' is supplied.

9. A session is now allocated naming the remote system only, and its name is obtained from EIBRSRCE.

10. A transaction is to be initiated on a remote system which needs to know the transaction name and the name to be given to the temporary storage queue which will be created. These are detailed in the attach header which is built at this point.

11. The contents of the first record on the local temporary storage queue are sent across the session. The attach header just built is prefixed onto the SEND command by means of the ATTACHID option.

12. A message is constructed and sent to the terminal operator informing him that the copying of the queue is now taking place.

13. A count is kept of the number of temporary storage queue records transmitted.

14. Set up exit for action to be taken at local temporary storage queue end.

15. Continue reading the local temporary storage queue and sending records across the session. (The attach header is not prefixed to these subsequent records).

16. The count of the number of records transmitted is unpacked and an informatory message sent to the terminal operator.

17. The local temporary storage queue is deleted and the program exits.

18. A syncpoint is taken to indicate end of transmitted data.

19. If an abend code is detected, its value is assigned, together with details of the last resource used.

20. A message warning of the abend is sent to the operator and the program terminates.

21. Some kind of error exists which prevents the link between the two systems from being established.

22. The remote system name given by the operator is not recognized.

23. The system name given is recognized, but is not that for a remote system.

24. This represents the start of the receiving element of the program. The first record transmitted, which will contain the attach header, is read and the contents of the EXEC Interface Block are accessed. No SYSID or SESSION options are required on the READ command since the principal facility is being addressed.

25. The name of the temporary storage queue to be created by this program is contained in the attach header just received; it is extracted at this point.

26. If the data length field for the RECEIVE indicates that there is data to be handled, it is written onto the temporary storage queue.

27. The session-oriented information transmitted across the LU6 session by the sending transaction must now be examined to determine what action should be taken next. The 'SYNCPOINT required' indicator in the EXEC Interface Block is first tested and if necessary the program issues its own SYNCPOINT. The sending program takes a syncpoint to indicate end of data transmitted, and the receiving program must test for the presence of this syncpoint indicator in every message it receives.

28. If the EXEC Interface Block indicates that the program should now free the session, thereby denoting that the remote transaction has completed successfully and has terminated the conversation, this program can itself end normally, when the session will be freed automatically.

29. If the EXEC Interface Block indicates that the application is to continue receiving data from the session, a further RECEIVE command is issued. If neither the SYNCPOINT, FREE session, or RECEIVE indicators apply, the program exits. In practice this situation should never occur since the sending application never sets up a RECEIVE.

MAP DEFINITION

```
XDFHAI2  DFHMSD TYPE=&SYSPARM,MODE=INOUT,CTRL=FREEKB,TIOAPFX=YES,       *
               LANG=ASM,STORAGE=AUTO
MAPA     DFHMDI SIZE=(12,80)
         DFHMDF POS=(1,01),ATTRB=ASKIP,LENGTH=28,                       *
               INITIAL='TS Q TRANSFER DRIVER PROGRAM'
         DFHMDF POS=(5,01),ATTRB=ASKIP,LENGTH=18,                       *
               INITIAL='LOCAL TS Q NAME...'
LOCALQ   DFHMDF POS=(5,30),ATTRB=(NORM,IC),LENGTH=8,INITIAL='        '
         DFHMDF POS=(5,39),LENGTH=1
         DFHMDF POS=(7,01),ATTRB=ASKIP,LENGTH=19,                       *
               INITIAL='REMOTE SYSTEM ID...'
SYSID    DFHMDF POS=(7,30),ATTRB=NORM,LENGTH=4,INITIAL='    '
         DFHMDF POS=(7,35),LENGTH=1
         DFHMDF POS=(9,01),ATTRB=ASKIP,LENGTH=19,                       *
               INITIAL='REMOTE TS Q NAME...'
REMOTQ   DFHMDF POS=(9,30),ATTRB=NORM,LENGTH=8,INITIAL='        '
         DFHMDF POS=(9,39),LENGTH=1
         DFHMDF POS=(11,1),ATTRB=ASKIP,LENGTH=35,                       *
               INITIAL='TYPE VALUES ABOVE, THEN PRESS ENTER'
MAPB     DFHMDI SIZE=(12,80)
         DFHMDF POS=(1,01),ATTRB=ASKIP,LENGTH=13,                       *
               INITIAL='COPYING TS Q '
MTSQ     DFHMDF POS=(1,15),ATTRB=NORM,LENGTH=8,INITIAL='        '
         DFHMDF POS=(1,24),ATTRB=ASKIP,LENGTH=4,INITIAL=' TO '
MSYSID   DFHMDF POS=(1,29),ATTRB=NORM,LENGTH=4,INITIAL='    '
MAPC     DFHMDI SIZE=(12,80)
COUNT    DFHMDF POS=(2,1),ATTRB=NORM,LENGTH=6
         DFHMDF POS=(2,8),ATTRB=ASKIP,LENGTH=32,INITIAL='TS Q ITEMS HAV*
               E BEEN TRANSFERRED'
MAPD     DFHMDI SIZE=(12,80)
ABCODE   DFHMDF POS=(1,1),ATTRB=NORM,LENGTH=4
         DFHMDF POS=(1,6),ATTRB=ASKIP,LENGTH=23,INITIAL='ABEND HAS BEEN*
               RECEIVED'
         DFHMDF POS=(3,1),ATTRB=ASKIP,LENGTH=22,INITIAL='LAST RESOURCE *
               USED WAS'
RSOURC   DFHMDF POS=(3,24),ATTRB=NORM,LENGTH=8
         DFHMSD TYPE=FINAL
         END
```

SCREEN LAYOUT

```
TS Q TRANSFER DRIVER PROGRAM


LOCAL TS Q NAME...              XXXXXXXX
REMOTE SYSTEM ID..              XXXX
REMOTE TS Q NAME..              XXXXXXXX
TYPE VALUES ABOVE, THEN PRESS ENTER
```

CICS TO OTHER SYNCHRONOUS SAMPLE PROGRAM

DESCRIPTION

This sample is activated with the transaction code 'AICO'.

The CICS to other Synchronous sample is intended to illustrate a conversation of a simple nature. It is planned to operate the sample with a program similar to the IMS ECHO application.

At the start of the program, the operator is prompted to enter the name of the remote system and application on that system, together with the input data to be ECHOed back.

The response at the terminal will consist of the reECHOed data only.

The message received from the remote system is assumed to contain printable characters only, and to be in variable length variable block format. Each logical record is treated as representing one screen line, and for the purposes of this sample, may not be greater than 79 characters in length.

SOURCE LISTING OF THE SENDING USER TRANSACTION

```
DFHEISTG DSECT
*
*         STORAGE AREA FOR EIB SESSION AND STATUS FLAGS
*
XDFEIFLG DS     0CL7
*
DFHSYNC  DS     C                       IF SET, SYNCHPOINT MUST
*                                       BE EXECUTED
DFHFREE  DS     C                       IF SET, TERMINAL / LU
*                                       MUST BE FREED
DFHRECV  DS     C                       IF SET, RECEIVE MUST
*                                       BE EXECUTED
DFHSEND  DS     C                       RESERVED
DFHATT   DS     C                       IF SET, ATTACH HEADER
*                                       DATA EXISTS AND MAY BE
*                                       ACCESSED USING EXTRACT
DFHEOC   DS     C                       IF SET, END-OF-CHAIN
*                                       WAS RECEIVED WITH DATA
DFHFMH   DS     C                       IF SET, DATA PASSED TO
*                                       APPL'N CONTAINS FMH(S)
         COPY   XDFHAI4
         COPY   DFHBMSCA             BMS ATTRIBUTES
R4       EQU    4
R5       EQU    5
R6       EQU    6
R7       EQU    7
R8       EQU    8
R9       EQU    9
REMSYS   DS     CL8
ATCHSESS DS     CL4
INLEN    DS     H
         DFHEJECT
XDFHAI4A CSECT
         XC     MAPAI(MAPAE-MAPAI),MAPAI  CLEAR MAP
SENDMAP  DS     0H
  1      EXEC CICS SEND MAP('MAPA') MAPSET('XDFHAI4') ERASE
  2      XC     DATAI,DATAI          RE-CLEAR THE DATA AREA
  3      EXEC CICS RECEIVE MAP('MAPA') MAPSET('XDFHAI4')
  4      CLI    SYSIDI,0             REMOTE SYSTEM NAME GIVEN ?
         BE     REMAP                ..NO, SEND MESSAGE TO OPERATOR
  5      EXEC CICS ALLOCATE SYSID(SYSIDI)
  6      MVC    ATCHSESS,EIBRSRCE
         B      BUILD
REMAP    DS     0H
  7      MVC    ERROIO(L'SYSMSG),SYSMSG SET UP PROMPTING MESSAGE
         MVI    ERROIA,DFHBMBRY      HIGHLIGHT MESSAGE
         B      SENDMAP              AND SEND IT.
BUILD    DS     0H
  8      EXEC CICS BUILD ATTACH ATTACHID('TIMS')  RESOURCE(TRANI)      *
               IUTYPE(=H'1')
  9      EXEC CICS SEND SESSION(ATCHSESS) ATTACHID('TIMS') FROM(DATAI) *
```

```
                LENGTH(DATAL) INVITE
RECV        DS    0H
 10         EXEC CICS RECEIVE SESSION(ATCHSESS) SET(R9) LENGTH(INLEN)
DATASENT    DS    0H
            MVC   XDFEIFLG,EIBSYNC          SAVE EIB VALUES
*
            LA    R4,MAPBI                  START OF OUTPUT MAP
            LR    R6,R4
            LA    R5,MAPBE-MAPBI            LENGTH OF MAP
            XR    R7,R7
            MVCL  R4,R6                     CLEAN UP THE MAP
*
 11         CLC   INLEN,=H'0'               IF NULL RU SENT THEN
            BE    TESTSYNC                  NOTHING TO SEND TO TERMINAL
*
            LA    R7,LINE10                 ADDRESS 1ST OUTPUT LINE
            LH    R4,INLEN                   LENGTH  OF RECEIVED DATA
LRECL       DS    0H
            LH    R5,0(R9)                  LOGICAL RECORD LENGTH
            SR    R4,R5                     REDUCE BLOCK LENGTH
            SH    R5,=H'5'                  PREPARE FOR EX INSTR.
            EX    R5,SETLINE                MOVE LOGICAL RECORD TO MAP
            LTR   R4,R4                     END OF BLOCK REACHED ?
            BZ    SENDMAPB                  ..YES, SEND THE MAP
            AH    R9,0(R9)                  ADVANCE TO NEXT RECORD
            LA    R7,LINE20-LINE10(R7)      ADDRESS NEXT OUTPUT LINE
            B     LRECL                     GO TO MOVE NEXT RECORD
SENDMAPB    DS    0H
            EXEC CICS SEND MAP('MAPB') MAPSET('XDFHAI4') ERASE WAIT       *
                  CURSOR(='1840')
*
TESTSYNC    DS    0H
 12         CLI   DFHSYNC,X'FF'
            BNE   TESTFREE
            EXEC CICS SYNCPOINT
TESTFREE    DS    0H
 13         CLI   DFHFREE,X'FF'
            BE    EXIT
 14         EXEC CICS RECEIVE SET(R8) LENGTH(INLEN)
 15         CLI   DFHRECV,X'FF'
            BE    RECV
 16         EXEC CICS CONVERSE FROMLENGTH(INLEN) SESSION(ATCHSESS)        *
                  SET(R9) TOLENGTH(INLEN) FROM(0(R8))
            B     DATASENT
*
SETLINE     MVC   0(0,R7),4(R9)          MOVE INPUT RECORD TO MAP
SYSMSG      DC    C'MUST SPECIFY REMOTE SYSID'
EXIT        DS    0H
            END
```

PROGRAM NOTES

1. The screen is erased, and the prompting map displayed at the terminal.

2. The data area portion of the map is used to hold any error messages sent to the terminal; this area is cleared before a RECEIVE is issued.

3. The remote system name and data are mapped in.

4. The terminal operator now enters the remote system name.

5. If the remote system name is given, an ALLOCATE is performed on that system, and

6. The name of the actual session allocated is found in the EIBRSRCE field.

7. Use the input data area of the map to advise the operator to try again.

8. A transaction is to be initiated on a remote system which needs to know the transaction name. This is detailed in the attach header which is built at this point. For IMS, the "transaction name" must be entered as the resource name; the processing name being reserved for an attached system process (when used). Also, since IMS requires single-chain input, the IUTYPE option is set to binary halfword '1'.

9. The data entered by the terminal operator is now sent across the

acquired session together with the previously built attach header. The presence of the INVITE option indicates that a RECEIVE will directly follow this SEND and improves performance across the session.

10. A RECEIVE is issued against the remote system to read back the echoed data. To enable the program to determine what action is next expected of it, the contents of the EXEC Interface Block will have to be retained.

11. If the data length field for the previous RECEIVE indicates that there is data to be handled, it is sent to the requesting terminal.

12. The session-oriented information transmitted across the LU6 session by the remote transaction must now be examined to determine what action should be taken next. The SYNCPOINT required indicator in the EXEC Interface Block is first tested and if necessary the program issues its own SYNCPOINT.

13. If the EXEC Interface Block indicates that the program should now free the session, thereby denoting that the remote transaction has completed successfully and has terminated the conversation, the program now exits causing an automatic freeing of the session.

14. The program receives further input data from the terminal operator. This allows for the remote program to send, for example, a request for further input. For simple autopaging through an output file, pressing ENTER is all that is required.

15. If the EXEC Interface Block indicates that the application is to continue receiving data from the session, a further RECEIVE command is issued.

16. A CONVERSE command is now issued which sends the data entered by the terminal operator to the remote system which he has specified, then receives the resulting response from that system. To enable the program to determine what action is next expected of it, the contents of the EXEC Interface Block must again be requested for the RECEIVE.

MAP DEFINITION

```
XDFHAI4  DFHMSD TYPE=&SYSPARM,MODE=INOUT,CTRL=FREEKB,TIOAPFX=YES,       *
               LANG=ASM,STORAGE=AUTO
MAPA     DFHMDI SIZE=(24,80)
         DFHMDF POS=(1,01),ATTRB=ASKIP,LENGTH=26,                      *
               INITIAL='INVOKE RETURN APPLICATION'
         DFHMDF POS=(3,01),ATTRB=ASKIP,LENGTH=25,                      *
               INITIAL='SUPPLY VALUES AS REQUIRED'
         DFHMDF POS=(4,01),ATTRB=(BRT,ASKIP),LENGTH=30,                *
               INITIAL='REMOTE TRANSACTION NAME......'
TRAN     DFHMDF POS=(4,32),ATTRB=(NORM,IC),LENGTH=8,INITIAL= ' '
         DFHMDF POS=(4,41),LENGTH=1
         DFHMDF POS=(5,01),ATTRB=(BRT,ASKIP),LENGTH=30,                *
               INITIAL='REMOTE SYSTEM ID.............'
SYSID    DFHMDF POS=(5,32),ATTRB=NORM,LENGTH=4,INITIAL='    '
         DFHMDF POS=(5,37),LENGTH=1
         DFHMDF POS=(8,01),ATTRB=(BRT,ASKIP),LENGTH=34,                *
               INITIAL='AND RETURN TRANSACTION INPUT DATA'
         DFHMDF POS=(8,36),LENGTH=1
DATA     DFHMDF POS=(10,1),ATTRB=NORM,LENGTH=79,INITIAL=' '
ERROI    DFHMDF POS=(11,1),ATTRB=NORM,LENGTH=79
MAPB     DFHMDI SIZE=(24,80)
LINE1    DFHMDF POS=(1,1),ATTRB=NORM,LENGTH=79
LINE2    DFHMDF POS=(2,1),ATTRB=NORM,LENGTH=79
LINE3    DFHMDF POS=(3,1),ATTRB=NORM,LENGTH=79
LINE4    DFHMDF POS=(4,1),ATTRB=NORM,LENGTH=79
LINE5    DFHMDF POS=(5,1),ATTRB=NORM,LENGTH=79
LINE6    DFHMDF POS=(6,1),ATTRB=NORM,LENGTH=79
LINE7    DFHMDF POS=(7,1),ATTRB=NORM,LENGTH=79
LINE8    DFHMDF POS=(8,1),ATTRB=NORM,LENGTH=79
LINE9    DFHMDF POS=(9,1),ATTRB=NORM,LENGTH=79
LINE10   DFHMDF POS=(10,1),ATTRB=NORM,LENGTH=79
MAPE     DFHMDI SIZE=(24,80)
ERROR    DFHMDF POS=(1,1),ATTRB=NORM,LENGTH=36
MAPP     DFHMDI SIZE=(24,80)
OLP      DFHMDF POS=(24,1),ATTRB=NORM,LENGTH=6,INITIAL=' '
         DFHMSD TYPE=FINAL
         END
```

SCREEN LAYOUT

```
INVOKE REMOTE APPLICATION

SUPPLY VALUES AS REQUIRED
REMOTE TRANSACTION NAME.....   XXXXXXXX
REMOTE SYSTEM ID............   XXXX

AND REMOTE TRANSACTION INPUT DATA

XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
```

DESCRIPTION

The following sample fulfills the same requirements as that above except that provision is made for the operator to read IMS Demand Paged Output using Operator Logical Paging.

The sample automatically reads the first logical page of the IMS output and it is then the responsibility of the terminal operator to signify which logical page of the output message he now requires to see.

The message received from the remote system is assumed to contain printable characters only, and to be in VLVB (variable length variable block) format. Each logical record is treated as representing one screen line, and for the purposes of this example, may not be greater than 79 characters in length.

This sample is activated with the transaction code 'AICD'.

The following functions, available to the operator, are supported by the sample; they should be preceded by "P/" as if normal CICS/VS BMS paging were being performed.

- N = display the next logical page.
- P = display the previous logical page.
- C = redisplay the current logical page.
- Enter =n, =nn or =nnn to display a specific logical page of the message.
- +n, +nn or +nnn to display the nth logical page past the current position.
- -n, -nn or -nnn to display the nth logical page before the current position.
- Press CLEAR to delete the current message from the IMS system and CLEAR the user's screen.

SOURCE LISTING OF THE SENDING USER TRANSACTION

```
DFHEISTG DSECT
         COPY  XDFHAI4
*
*        STORAGE AREA FOR EIB SESSION AND STATUS FLAGS
*
XDFEIFLG DS    0CL7
*
DFHSYNC  DS    C                    IF SET, SYNCPOINT MUST
*                                   BE EXECUTED
DFHFREE  DS    C                    IF SET, TERMINAL / LU
*                                   MUST BE FREED
DFHRECV  DS    C                    IF SET, RECEIVE MUST
*                                   BE EXECUTED
DFHSEND  DS    C                    RESERVED
*
DFHATT   DS    C                    IF SET, ATTACH HEADER
*                                   DATA EXISTS AND MAY BE
*                                   ACCESSED USING EXTRACT
DFHEOC   DS    C                    IF SET, END-OF-CHAIN
*                                   WAS RECEIVED WITH DATA
DFHFMH   DS    C                    IF SET, DATA PASSED TO
*                                   APPL'N CONTAINS FMH(S)
         COPY  DFHBMSCA             BMS ATTRIBUTES
R4       EQU   4
R5       EQU   5
R6       EQU   6
R7       EQU   7
DPAGEREG EQU   8
R9       EQU   9
*
DBLWORD  DS    D
INLEN    DS    H
REMSYS   DS    CL8
ATCHSESS DS    CL4
*
OLP      DSECT
OLPCODE  DS    CL2
OLPVAL   DS    CL3
```

```
         DFHEJECT
XDFHAI4B CSECT
  1      XC    MAPAI(MAPAE-MAPAI),MAPAI  CLEAR MAP
SENDMAP  DS    0C
         EXEC CICS SEND MAP('MAPA') MAPSET('XDFHAI4') ERASE WAIT
  2      EXEC CICS RECEIVE MAP('MAPA') MAPSET('XDFHAI4')
         CLI   SYSIDI,0              REMOTE SYSTEM NAME GIVEN ?
         BE    REMAP                 ..NO, SEND MSG TO OPERATOR
  3      EXEC CICS ALLOCATE SYSID(SYSIDI)
  4      MVC   ATCHSESS,EIBRSRCE
         B     BUILD
REMAP    DS    0H
  5      MVC   ERROIO(L'SYSMSG),SYSMSG SET UP PROMPTING MSG
         MVI   ERROIA,DFHBMBRY       HIGHLIGHT MESSAGE
  6      XC    MAPAI(MAPAE-MAPAI),MAPAI  RE-CLEAR MAP
         B     SENDMAP               AND SEND IT.
BUILD    DS    0H
  7      EXEC CICS IGNORE CONDITION INBFMH.
  8      EXEC CICS HANDLE AID CLEAR(CLEAR)
  9      EXEC CICS BUILD ATTACH                                          *
               ATTACHID('TIMS')  RESOURCE(TRANI) IUTYPE(=H'1')
 10      EXEC CICS SEND SESSION(ATCHSESS) ATTACHID('TIMS') FROM(DATAI) *
               LENGTH(DATAL) INVITE
         EXEC CICS SYNCPOINT
 11      EXEC CICS RECEIVE SESSION(ATCHSESS)                             *
               SET(R9) LENGTH(INLEN)
 12      MVC   XDFEIFLG,EIBSYNC      SAVE EIB VALUES
 13      CLI   DFHATT,X'FF'          IF NO HEADER SENT,
         BNE   ABEND                 REMOTE SYSTEM ERROR.
 14      EXEC CICS EXTRACT ATTACH SESSION(ATCHSESS)                      *
               QUEUE(QGETQNAM).      GET REMOTE QUEUE NAME.
 15      MVC   QGETNQNM,QGETQNAM
         MVC   QPURGENM,QGETQNAM
 16      EXEC CICS BUILD ATTACH                                          *
               ATTACHID('QMOD') PROCESS(QMODEL)  IUTYPE(=H'1')
RECV     DS    0H
 17      LA    DPAGEREG,1            1ST LOGICAL PAGE.
 18      EXEC CICS CONVERSE SESSION(ATCHSESS) FROM(QGETN)                *
               FROMLENGTH(QGETNLEN) TOLENGTH(INLEN) SET(R9)            *
               ATTACHID('QMOD') FMH
UNPICK   DS    0H
         MVC   XDFEIFLG,EIBSYNC
 19      LA    R4,MAPBI              START OF OUTPUT MAP
         LR    R6,R4
         LA    R5,ERRORL-MAPBI       LENGTH OF MAP
         XR    R7,R7
         MVCL  R4,R6                 CLEAN UP THE MAP
*
         LH    R4,INLEN              LENGTH OF REC'D DATA
*
         XR    R5,R5
CATFMH   DS    0H
         IC    R5,0(R9)              FMH LENGTH.
         LR    R6,R9
 20      AR    R9,R5                 POINT BEYOND FMH.
         SR    R4,R5                 LENGTH OF ACTUAL DATA.
 21      BZ    QSTATUS               QSTATUS IF NO DATA.
         TM    1(R6),X'80'           ANY CONCATENATED FMHS ?
         BO    CATFMH                YES - AGAIN.
*
         LA    R7,LINE10             ADDRESS 1ST OUTPUT LINE
LRECL    DS    0H
 22      LH    R5,0(R9)              LOGICAL RECORD LENGTH
         SR    R4,R5                 REDUCE BLOCK LENGTH
         SH     R5,=H'3'               PREPARE FOR EX INSTR.
 23      EX    R5,SETLINE            MOVE LREC TO MAP
         LTR   R4,R4                 END OF BLOCK REACHED ?
         BZ    SENDMAPB              ..YES, SEND THE MAP
         AH    R9,0(R9)              ADVANCE TO NEXT RECORD
         LA    R7,LINE20-LINE10(R7)  ADDR NEXT OUTPUT LINE
         B     LRECL                 GO TO MOVE NEXT REC
SENDMAPB DS    0H
 24      EXEC CICS SEND MAP('MAPB') MAPSET('XDFHAI4') ERASE WAIT         *
```

```
                CURSOR(=H'1841')
          XC    ERRORO,ERRORO                 CLEAR ERROR LINE.
*
TESTSYNC  DS    0H
 25       CLI   DFHSYNC,X'FF'
          BNE   TESTFREE
          EXEC CICS SYNCPOINT
TESTFREE  DS    0H
 26       CLI   DFHFREE,X'FF'
          BE    EXIT
 27       CLI   DFHRECV,X'FF'
          BE    ABEND
*
GETOLP    DS    0H
 28       EXEC CICS RECEIVE SET(R9) LENGTH(INLEN)
*
          LA    R9,3(R9)                      BYPASS SBA BYTES.
          USING OLP,R9
 29       CLC   OLPCODE,=C'P/'                'P/' REQUIRED TO START.
          BNE   OLPERR
 30       CLI   OLPVAL,C'C'                   CURRENT PAGE AGAIN ?
          BE    READQ                         YES.
 31       CLI   OLPVAL,C'N'                   NEXT PAGE REQUIRED ?
          BNE   TESTPREV                      NO.
          LA    DPAGEREG,1(DPAGEREG)          YES, SET TS ITEM NO.
          B     READQ
*
TESTPREV  DS    0H
 32       CLI   OLPVAL,C'P'                   PREVIOUS PAGE REQ'D ?
          BNE   CODETEST                      NO.
          BCTR  DPAGEREG,0                    YES, SET TS ITEM NO.
          B     READQ
*
CODETEST  DS    0H
 33       LH    R4,INLEN
          SH    R4,=H'6'                      SBA +  3 CHARS.
          BM    OLPERR
*
          TM    OLPVAL,C'0'                   IF FIRST CHAR. IS A DIGIT,     *
                                              NO SIGN HAS BEEN GIVEN.
          BO    NOSIGN
          BCTR  R4,0                          REDUCE LENGTH BY ONE.
          LA    R5,OLPVAL+1                   ADDRESS 1ST DIGIT.
          B     PACKINST                      CONVERT TO TS ITEM NO.
NOSIGN    DS    0H
          LA    R5,OLPVAL                     ADDRESS 1ST DIGIT.
PACKINST  DS    0H
          EX    R4,OC                         ENSURE NO. IS NUMERIC.
          EX    R4,PACK                       PACK PAGE NO. AND
          CVB   R5,DBLWORD                    CONVERT TO BINARY VALUE.
          TM    OLPVAL,C'0'                   IF 1ST CHAR. IS NOT A DIGIT,   *
                                              IT MUST BE A SIGN.
          BNO   TMINUS
          LR    DPAGEREG,R5                   RESET TS ITEM NO.
          B     READQ
*
TMINUS    DS    0H
          CLI   OLPVAL,C'-'
          BNE   TPLUS                         FOR '+'
          LNR   R5,R5                         REDUCE CURRENT PAGE NO.
          B     NEWDPAGE
*
TPLUS     DS    0H
          CLI   OLPVAL,C'+'
          BNE   OLPERR
*
NEWDPAGE  DS    0H
          AR    DPAGEREG,R5                   SET TS ITEM NO.
*
READQ     DS    0H
          LTR   DPAGEREG,DPAGEREG             IF PAGE NO. IS ZERO
          BZ    OLPERR                        THIS IS AN ERROR.
 34       STCM  DPAGEREG,3,DPAGENO            STORE QUEUE RECORD NO.
```

```
         EXEC CICS CONVERSE SESSION(ATCHSESS) FROM(QGET)                *
               FROMLENGTH(QGETLEN) TOLENGTH(INLEN) SET(R9)  FMH
 35      CLC   INLEN,=H'0'              IF NULL RU SENT,THEN
         BNE   UNPICK                   ANALYZE INPUT
         MVC   XDFEIFLG,EIBSYNC
         B     TESTSYNC                 NOTHING TO SEND
*
OC       OC    0(0,R5),=C'000'          ENSURE NUMERIC.
PACK     PACK  DBLWORD,0(0,R5)
*
OLPERR   DS    0H
 36      MVC   ERRORO(L'OLPERMSG),OLPERMSG  SET UP MSG.
         B     SENDMAPB
*
QSTATUS  DS    0H
 37      MVC   ERRORO(L'QSTAMSG),QSTAMSG    SET UP MSG.
         LA    DPAGEREG,1
         EXEC CICS CONVERSE SESSION(ATCHSESS) FROM(QGETN)               *
               FROMLENGTH(QGETNLEN) TOLENGTH(INLEN) SET(R9) FMH
         B     UNPICK
*
CLEAR    DS    0H
 38      EXEC CICS CONVERSE SESSION(ATCHSESS) FROM(QPURGE)              *
               FROMLENGTH(QPURGELN) TOLENGTH(INLEN) SET(R9)  FMH
         B     EXIT
*
ABEND    DS    0H
         MVC   ERRORO(L'ABENDMSG),ABENDMSG SET UP ERROR MSG.
         EXEC CICS SEND MAP('MAPB') MAPSET('XDFHAI4')                   *
               WAIT.
         B     EXIT
*
SETLINE  MVC   0(0,R7), 2(R9)            MOVE LOG.REC. TO MAP.
*
SYSMSG   DC    C'MUST SPECIFY REMOTE SYSID'
OLPERMSG DC    C'OPERATOR LOGICAL PAGING ERROR - RE-TYPE'
ABENDMSG DC    C'PROCESSING ERROR IN REMOTE SYSTEM'
QSTAMSG  DC    C'PAGE NO. EXCEEDS QUEUE SIZE'
*
*        QGETN
*
QGETN    DS    0H
         DC    X'10060A1000010208'
QGETQNAM DC    CL8' '
*
QGETNLEN DC    AL2(*-QGETN)        LENGTH.
*
*        QGET
*
QGET     DS    0H
         DC    X'13060A04000102'
         DC    X'08'
QGETNQNM DC    CL8' '
         DC    X'02'
DPAGENO  DS    CL2
*
QGETLEN  DC    AL2(*-QGET)         LENGTH.
*
*        QPURGE
*
QPURGE   DS    0H
         DC    X'10060A06000102'
         DC    X'08'
QPURGENM DC    CL8' '
*
QPURGELN DC    AL2(*-QPURGE)       LENGTH.
*
QMODEL   DS    0CL8
         DC    X'03'
         DC    CL7' '
EXIT     DS    0H
         END
```

PROGRAM NOTES

1. The screen is erased, and the prompting map displayed at the terminal.

2. The remote system name and data are mapped in.

3. If the remote system name is given, an ALLOCATE is performed on that system, and

4. The name of the actual session allocated is found in the EIBRSRCE field.

5. Use the input data area of the map to advise the operator to reenter his data, correctly naming the remote system.

6. The map is recleared to ensure that all three fields are correctly reentered.

7. When pages are returned by IMS, they are preceded by a QXFR FMH; in this instance this FMH need not be examined, but the INBFMH condition will be raised by CICS/VS and should be ignored.

8. Pressing CLEAR will cause the sample to instruct IMS to delete the demand paging queue after which the program terminates.

9. A transaction is to be initiated on a remote system; the name of the transaction on that system is supplied via the attach FMH, built at this point.

10. The data entered by the terminal operator is now sent across the acquired session together with the previously built attach header. The presence of the INVITE option indicates that a RECEIVE will directly follow this SEND and improves performance across the session.

11. A RECEIVE is issued against the remote system to read back the IMS reply. IMS will initially transmit an attach FMH to signify that the output will now be sent at the request of the CICS/VS terminal operator; this header will be examined to enable the name of the IMS demand paged output queue to be found.

12. To enable the program to determine what action should next be performed on the session, the contents of the EXEC Interface Block, set by RECEIVE, will have to be retained for future reference.

13. For IMS demand paging output queues, IMS sends as its initial output an attach header. The absence of this header indicates an error on the remote system.

14. The IMS queue name is extracted. (The SESSION option is required in this instance, since the EXTRACT relates to data sent by this sample's alternate facility; without this option, the principal facility, that is, the operator terminal, would be addressed.)

15. The sample will issue three types of request to the IMS queue -

    a) Get Next
    b) Get (specific)
    c) Purge

    The queue model FMHs required to perform these functions must be completed so as to contain the appropriate IMS queue name.

16. Preceding each queue model FMH, IMS needs an attach FMH, which must contain the queue model function as its destination process name. The FMH is built at this point and will be used in conjunction with all remaining commands across the session.

17. The program has to keep a note of the page number of the logical page being currently accessed on the IMS queue; this is to enable the new page number to be correctly calculated each time a logical paging command is entered by the operator. To do this, the register 'DPAGEREG' is used to hold the current number.

18. In order to open the IMS queue for output a GET NEXT command has first to be issued; this will cause the first logical page of the message to be returned to CICS/VS. Thereafter, GET commands will be issued. It will be seen that the command is sent as a text string containing an attach FMH together with the queue model FMH. The use of the ATTACHID and FMH options should be noted.

19. The record sent by IMS (that is, a logical page) is now prepared for writing to the operator's terminal.

20. The output record received from IMS will contain the requested page record preceded by a QXFR FMH; this FMH is not required in this sample and is bypassed.

21. The presence of a FMH but no accompanying data in the message

returned from IMS indicates that a request has been made for a logical page outside the dimensions of the queue size. In such instances, IMS sends a QSTATUS FMH with the QINVCUR (invalid cursor) flag set.

22. The output from IMS is in VLVB format; the IMS mapping function sets one screen output line as one logical record. The following lines of code unpack the physical record received to obtain single logical records for transmission to the terminal via BMS.

23. One logical record is inserted and a check made for further logical records.

24. The whole logical message is now sent.

25. The session-oriented information transmitted across the LU6 session by the remote transaction must now be examined to determine what action should be taken next. The syncpoint required indicator in the EXEC Interface Block is tested and if necessary the program issues its own SYNCPOINT.

26. If the EXEC Interface Block indicates that the program should now free the session, thereby denoting that the remote transaction has completed successfully and has terminated the conversation, the program now exits causing an automatic freeing of the session.

27. If the EXEC Interface Block indicates that a further RECEIVE should be made over the session, some kind of error has occurred, since normally IMS will be awaiting a paging command at this point; thus the program should be in 'send' state.

28. The operator now enters a paging command as if this were a normal CICS/VS application.

29. The command must begin with 'P/'.

30. If the current page ('P/C') is required again, go back to reuse the current page number in the GET command.

31. If the next page ('P/N') is required, add 1 to 'DPAGEREG' and issue the GET command.

32. If the previous page ('P/P') is required, subtract 1 from 'DPAGEREG' and issue the GET command.

33. The presence of a '+' or '-' sign is now detected, in which case the increment or decrement is found and either added to or subtracted from the logical page number 'DPAGEREG'. If no sign is found, the actual value typed in is the new logical page number required.

34. The page number of the logical record to be read next, held in 'DPAGEREG' is stored into DPAGENO and a GET command issued.

35. If the data length field indicates that no data has been sent, the session status must be tested to determine what to do next; otherwise, the new data will be unpacked.

36. If any error is detected in the paging command entered by the operator, an error message is sent to him to prompt for the command to be reentered correctly.

37. The QSTATUS FMH indicates that IMS has detected an invalid paging request. Having sent the QSTATUS, IMS relocks the queue in question, and it is the responsibility of the queue owner, in this case the sample program, to open the queue again for further processing; this is done by issuing the original GET NEXT which will unlock the queue and resend the first logical page.

38. A queue purge request is sent to IMS to cause it to delete the demand paging queue. This command is sent using CONVERSE since IMS will respond to the purge request by returning a QSTATUS FMH and the program must allow for its receipt.

ADDITIONS TO TABLES FOR THE SAMPLE PROGRAMS

The following entries are required in the using system's PPT and PCT to enable these samples to be run:

PPT

```
DFHPPT TYPE=ENTRY,PROGRAM=XDFHAMC
DFHPPT TYPE=ENTRY,PROGRAM=XDFHAI1
DFHPPT TYPE=ENTRY,PROGRAM=XDFHAI2
DFHPPT TYPE=ENTRY,PROGRAM=XDFHAI4
DFHPPT TYPE=ENTRY,PROGRAM=DFHXAIBL
DFHPPT TYPE=ENTRY,PROGRAM=DFHXAIBR
DFHPPT TYPE=ENTRY,PROGRAM=DFHXAI1A
DFHPPT TYPE=ENTRY,PROGRAM=DFHXAI2A
DFHPPT TYPE=ENTRY,PROGRAM=DFHXAI4A
DFHPPT TYPE=ENTRY,PROGRAM=DFHXAI4B
```

PCT

```
DFHPCT TYPE=ENTRY,TRANSID=AIBL,PROGRAM=DFHXAIBL,                    *
      SPURGE=YES,TPURGE=YES,INBFMH=ALL
DFHPCT TYPE=ENTRY,TRANSID=AIBR,PROGRAM=DFHXAIBR,                    *
      SPURGE=YES,TPURGE=YES
DFHPCT TYPE=ENTRY,TRANSID=AICC,PROGRAM=DFHXAI1A,                    *
      SPURGE=YES,TPURGE=YES
DFHPCT TYPE=ENTRY,TRANSID=AISC,PROGRAM=DFHXAI2A,                    *
      SPURGE=YES,TPURGE=YES
DFHPCT TYPE=ENTRY,TRANSID=AISR,PROGRAM=DFHXAI2A,                    *
      SPURGE=YES,TPURGE=YES,INBFMH=ALL,EXTRACT=ATTACH
DFHPCT TYPE=ENTRY,TRANSID=AICO,PROGRAM=DFHXAI4A,                    *
      SPURGE=YES,TPURGE=YES,INBFMH=ALL,EXTRACT=ATTACH
DFHPCT TYPE=ENTRY,TRANSID=AICD,PROGRAM=DFHXAI4B,                    *
      SPURGE=YES,TPURGE=YES,INBFMH=ALL,EXTRACT=ATTACH
```

Bibliography

For further information about the Customer Information Control System/Virtual Storage (CICS/VS), refer to the following IBM CICS/VS publications:

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

General Information, GC33-0066

System/Application Design Guide, SC33-0068

System Programmer's Reference Manual, SC33-0069

System Programmer's Guide (DOS/VS), SC33-0070

System Programmer's Guide (OS/VS), SC33-0071

Application Programmer's Reference Manual, (Macro Level), SC33-0079

Application Programmer's Reference Manual, (RPG II), SC33-0085

IBM 3270 Guide, SC33-0096

IBM 3600/3630 Guide, SC33-0072

IBM 3650/3680 Guide, SC33-0073

IBM 3767/3770/6670 Guide, SC33-0074

IBM 3790/3730 Guide, SC33-0075

Operator's Guide, SC33-0080

Messages and Codes, SC33-0081

Entry-Level System User's Guide (DOS/VS), SC33-0086

Problem Determination Guide, SC33-0089

Diagnosis Reference, LC33-0105

Data Areas (DOS/VS), LY33-6033

Data Areas (OS/VS), LY33-6035

Application Programmer's Reference Summary (Command Level), GX33-6012

Master Terminal Operator's Reference Summary, SX33-6011

Programming Debugging Reference Summary, SX33-6010

Master Index, SC33-0095

The reader of this manual may also want to refer to the following IBM publications:

Systems Network Architecture (SNA)

Logical Unit Types, GC20-1868

Types of Logical Unit to Logical Unit Sessions, GC20-1869

Distributed Processing Programming Executive/Distributed Presentation Services (DPPX/DPS)

DPPX/Distributed Presentation Services Version 2 System Programming Guide, SC33-0117

Screen Definition Facility/CICS SDF/CICS Program Reference Manual, SH19-6077

AVAILABILITY OF PUBLICATIONS

The availability of a publication is indicated by its use key, which is the first letter in the order number. The use keys and their meanings are:

G -- Generally available: Provided to users of IBM systems, products, and services without charge, in quantities to meet their normal requirements. Can also be purchased by anyone through IBM branch offices.

S -- Sold: Can be purchased by anyone through IBM offices.

L -- Licensed material, property of IBM: Available only to licensees of the related program products under the terms of the license agreements.

Index

Special Characters

A

B

D

G

H

I

J

K

L

M

Q

R

S

T

U

V

W

X

2

3

7

8

Customer Information Control System/Virtual Storage (CICS/VS)
Application Programmer's Reference Manual (Command Level)

SC33-0077-3

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. You may use this form to communicate your comments about this publication, its organization, or subject matter, with the understanding that IBM may use or distribute whatever information you supply in any way it believes appropriate without incurring any obligation to you.

Your comments will be sent to the author's department for whatever review and action, if any, are deemed appropriate. Comments may be written in your own language; English is not required.

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 _____

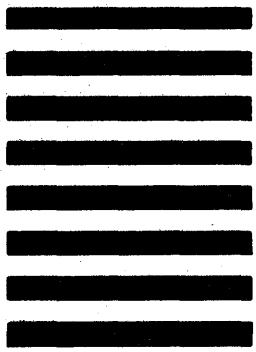
Thank you for your cooperation. No postage stamp necessary if mailed in the U.S.A. (Elsewhere, an IBM office or representative will be happy to forward your comments or you may mail directly to either address in the Edition Notice on the back of the title page.)

SC33-0077-3

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 the 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 Application Programmer's Reference Manual (Command Level) Printed in U.S.A. SC33-0077-3

CICS/VS Application Programmer's Reference Manual (Command Level) Printed in U.S.A. SC33-0077-3

SC33-0077-3