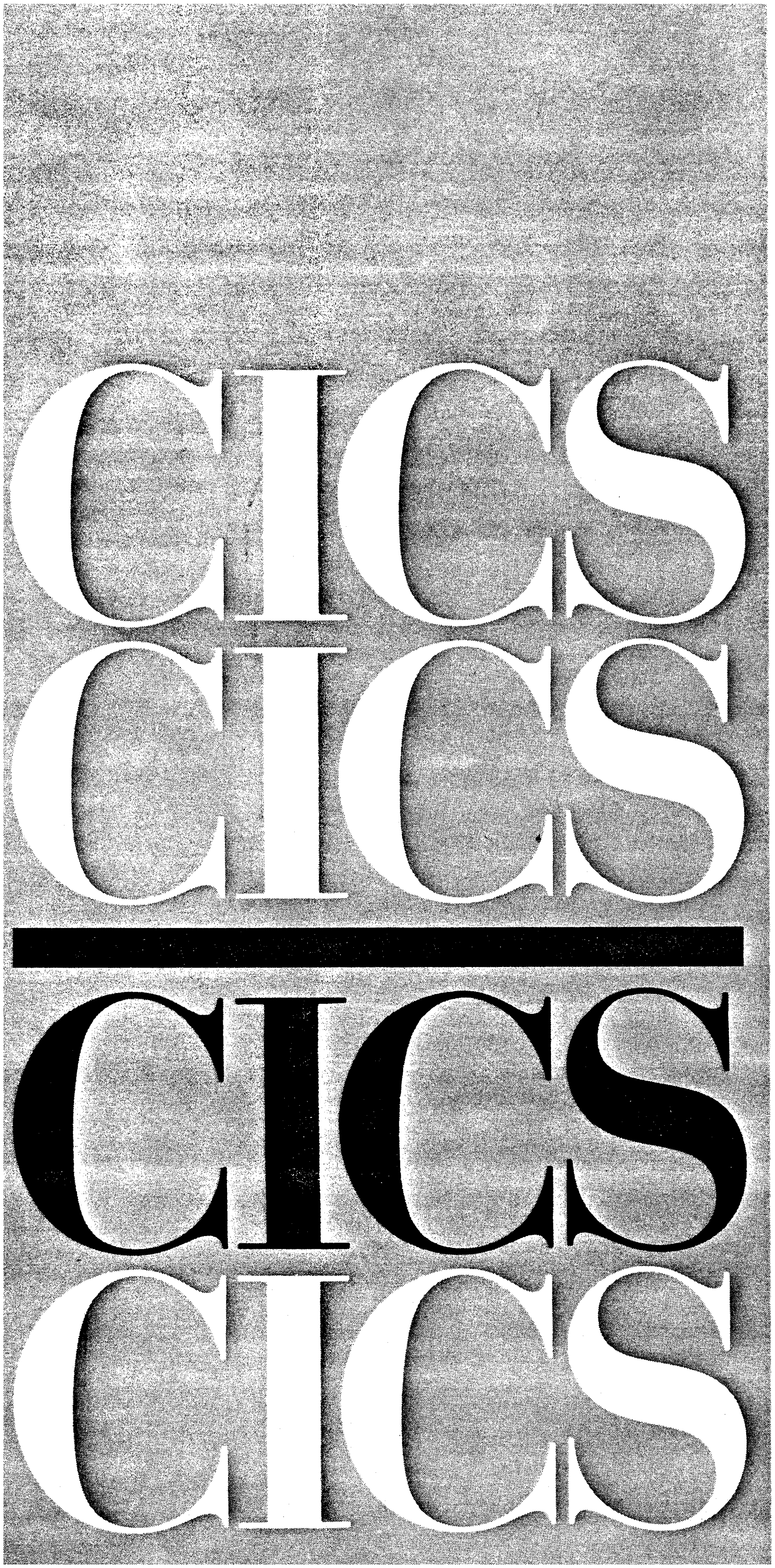
IBM



Customer
Information
Control
System
CICS/VS

Licensed Program
Version 1.7

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

Application Programmer's Reference Manual (Macro Level)

SC33-0079-5

Sixth Edition (July 1987)

This edition applies to Version 1 Release 7 (Version 1.7) of the IBM licensed program Customer Information Control System/Virtual Storage (CICS/VS), program numbers 5740-XX1 (CICS/OS/VS) and 5746-XX3 (CICS/DOS/VS), and to all subsequent releases and modifications until otherwise indicated in new editions or technical newsletters.

This edition is based on SC33-0079-3, which applied to CICS/OS/VS Version 1 Release 6 and to CICS/DOS/VS Version 1 Release 6. Changes from that edition are indicated by vertical lines to the left of the changes. The 1.6 edition remains applicable and current for CICS/VS 1.6 users, and can now be ordered by using the temporary order number ST00-1070.

Changes are made periodically to the information herein; before using this publication in connection with the operation of IBM systems, consult the latest IBM System/370, 30xx, and 4300 Processors Bibliography, GC20-0001, for the editions that are applicable and current.

PREFACE

This publication describes the IBM Customer Information Control System/Virtual Storage (CICS/VS) macro level application programming interface; it contains introductory and reference information necessary to prepare assembler language, COBOL, and PL/I application programs, using CICS macros, to execute under either of two IBM licensed programs: CICS/OS/VS (5740-XX1) or CICS/DOS/VS (5746-XX3).

The publication 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) General Information manual is required.

From CICS/VS Version 1 Release 6 onwards, enhancements to the CICS licensed program for application program use will, in general, only be provided at the command level interface. This interface is simpler and more usable than the macro level interface and can help increase application programmer productivity and reduce coding errors.

However, for 1.7, the manual is issued with the minimum number of changes to make it usable by both CICS/OS/VS and CICS/DOS/VS users who have migrated to 1.7. The opportunity has also been taken to correct errors and incorporate readers' comments.

This publication consists of eight parts, the first seven comprising one or more chapters and the eighth containing appendixes. Each of the first seven parts (except Part 1) contains information on a particular topic, both procedural and reference. In general, each chapter consists of the following:

- A brief introduction to the facilities available by specifying the macros that are described in detail in the remainder of the chapter.
- The syntax of each macro in the standard form (described in Chapter 1.2).
- The operands, in alphabetical order, that can be specified with the macros.

Where appropriate, examples in the three programming languages (assembler language, COBOL, and PL/I) that can be used with CICS have been included.

Part 1 is an introduction to macro level application programming. It compares the CICS DB/DC system with the conventional batch system of data processing. It also describes the general format of a CICS macro and explains the syntax notation used throughout the publication.

Part 2 describes symbolic storage definition. This, together with addressability, must be specified in the application program to enable the application program to be executed under CICS. The preparation of an application program for execution is described in the appropriate CICS Installation and Operations Guide.

Part 3 describes files and data bases: file control (including browsing) and DL/I services.

Part 4 describes data communication operations: terminal control, basic mapping support (BMS), and batch data interchange. You should refer to the appropriate CICS Application Programmer's Reference Manual (Command level) for descriptions of the additional BMS attributes that are new for this release.

Part 5 describes control operations: interval control, task control, program control, storage control, transient data control, and temporary storage control.

Part 6 describes built-in functions: table search, phonetic conversion, data field verification, data field edit, bit manipulation, input formatting, and weighted retrieval.

Part 7 describes error handling and debugging, trace services, dump services, journal services, and recovery/restart services.

Part 8 consists of appendixes. These include sample programs, BMS examples, fields that make up the application programming interface, and translate tables.

Experience in writing programs in assembler language, COBOL, or in PL/I is assumed. (Note: in some places in the publication, ASM is used as the abbreviation for assembler language.)

In this publication, the term VTAM refers to the record interface of ACF/TCAM (CICS/OS/VS only), and to ACF/VTAM and ACF/VTAME (CICS/DOS/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).

The term DAM refers to BDAM (CICS/OS/VS only) and to DAM (CICS/DOS/VS only).

For more information about CICS and related subjects discussed in this publication, the reader is referred to the publications listed in the bibliography, particularly to the appropriate Facilities and Planning Guide, which provides a good overall description of CICS.

Details of system requirements and a glossary of terms applicable to CICS are provided in the CICS/VS General Information manual.

CONTENTS

FIGURES

SUMMARY OF AMENDMENTS

SUMMARY OF AMENDMENTS FOR VERSION 1 RELEASE 7

This sixth edition (SC33-0079-5) provides limited information about the enhanced features introduced by Version 1 Release 7 for both CICS/OS/VS and CICS/DOS/VS. This edition also contains maintenance and editorial updates.

SUMMARY OF AMENDMENTS FOR VERSION 1 RELEASE 7 (CICS/OS/VS ONLY)

The fifth edition (SC33-0079-4) provided information about the enhanced features introduced by CICS/OS/VS Version 1 Release 7, in the area of file control. This edition also contained maintenance and editorial updates.

SUMMARY OF AMENDMENTS FOR VERSION 1 RELEASE 6

The fourth edition (SC33-0079-3) contained maintenance and editorial updates only. The opportunity was taken to change the format to double column, a change that involved some reformatting.

Questionnaire

Application Programmer's Reference Manual (ML)

(CICS/VS Version 1 Release 7)

To help us produce books that meet your needs, please fill in this questionnaire. It would help us if you provide your name and address in case we need to clarify any of the points you raise. Please understand that IBM may use or distribute whatever information you supply in any way it believes appropriate without incurring any obligation to you.

1. Please rate the book on the points shown below

   The book is:

accurate	1	2	3	4	5	inaccurate
readable	1	2	3	4	5	unreadable
well laid out	1	2	3	4	5	badly laid out
well organized	1	2	3	4	5	badly organized
easy to understand	1	2	3	4	5	incomprehensible
adequately illustrated	1	2	3	4	5	inadequately illustrated
has enough examples	1	2	3	4	5	has too few examples

   And the book as a whole?

excellent	1	2	3	4	5	poor

2. When using this book, did you find what you were looking for? ______

   What were you looking for? ______

   What led you to this book? ______

   Did you come straight to this book? ______

3. Which topics does the book handle well?

4. And which does it handle badly?

5. How could the book be improved? ______

6. How often do you use this book?

   Less than once a month? ☐ Monthly? ☐ Weekly? ☐ Daily? ☐

7. What sort of work do you use CICS for? ______

8. How long have you been using CICS? _____ years/months

9. Have you any other comments to make?

Thank you for your time and effort. No postage stamp necessary if mailed in the USA. (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.)

Please use pressure-sensitive or other gummed tape to seal this form.

SC33-0079-5

Questionnaire

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 NO. 40 ARMONK, N.Y.

POSTAGE WILL BE PAID BY ADDRESSEE:

International Business Machines Corporation
Department 6R1H,
180 Kost Road,
Mechanicsburg, PA 17055, USA

Fold and tape Please Do Not Staple Fold and tape

IBM®

Name ....

Job Title .... Company ....

Address ....

.... Zip ....

PART 1. INTRODUCTION

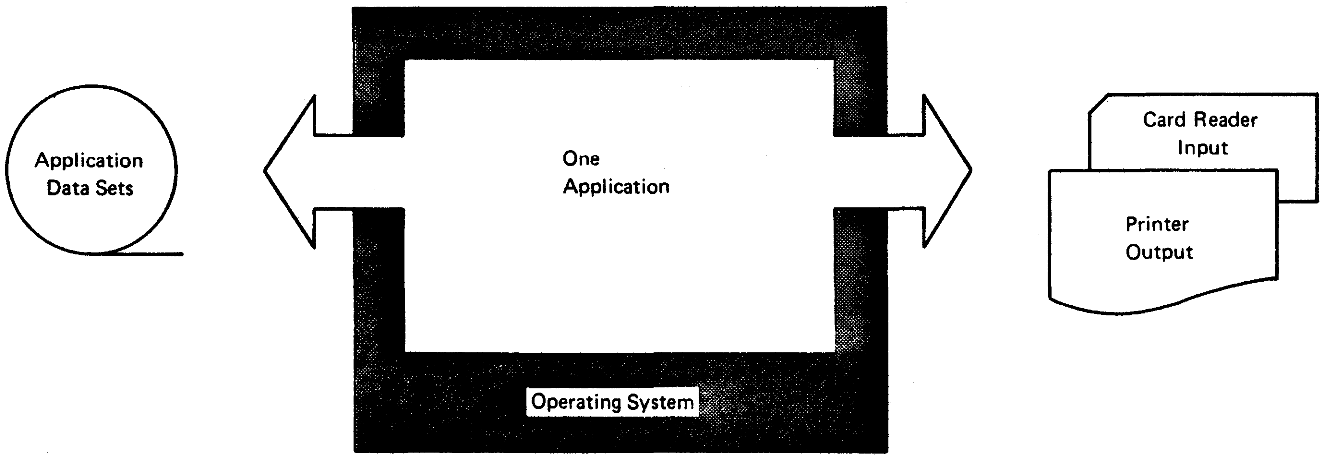


Figure 1. Conventional Batch Processing

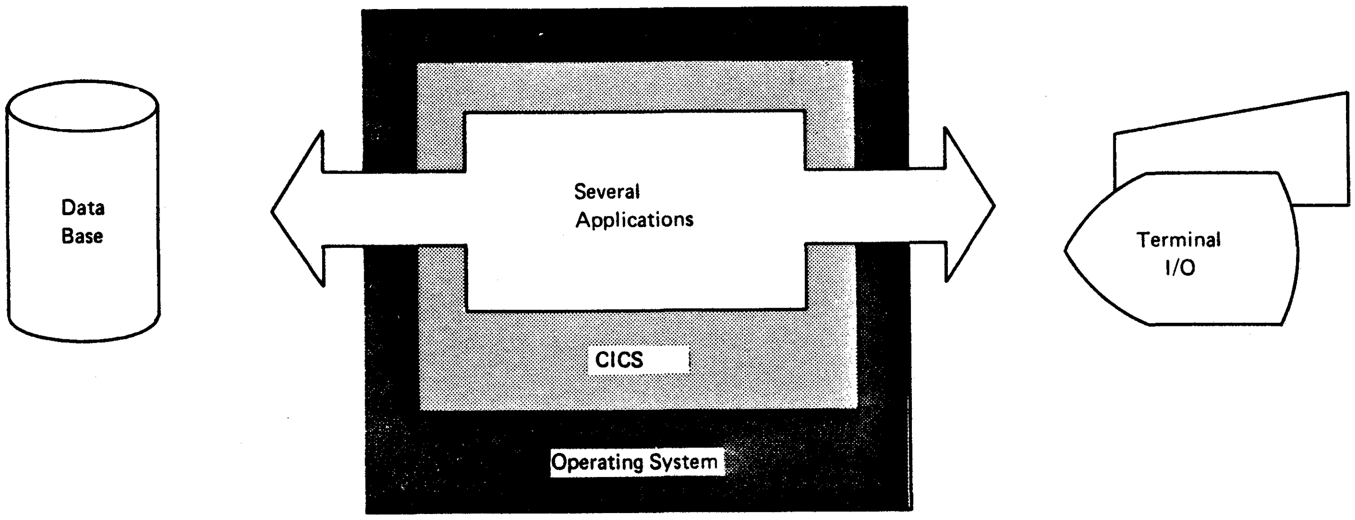


Figure 2. Transaction Processing of CICS

CHAPTER 1.1. MACRO-LEVEL APPLICATION PROGRAMMING

The IBM Customer Information Control System/Virtual Storage (CICS/VS) is a transaction-oriented data base/data communication (DB/DC) system. It can be applied to most online IBM System/370 systems, since it offers terminal facilities for many applications: message switching, inquiry, data collection, order entry, and conversational and batched data entry.

CICS works with either the Operating System/Virtual Storage (MVS) or with VSE/Advanced Functions. It can be thought of as an extension of the operating system or as an interface between the operating system and the user's application programs. The system is modular: at system generation or initialization, an installation can select the components it needs to tailor a CICS system for a given application.

In conventional batch processing (see Figure 1 on page 2), similar transactions are grouped for processing, and the application programmer plans a series of runs to edit input transactions, update data sets, or write output reports. Because the programmer concentrates on manipulating data for most efficient handling of each transaction type, the data in batch processing becomes closely tied to the program logic and has little value for other applications.

A realtime DB/DC system differs from batch processing primarily in the number and types of activities taking place in the system at the same time. A batch-processing system schedules each application independently and provides data base support unique to each application. A DB/DC system controls many random nonscheduled transactions for many applications and provides one integrated data base supporting all the applications on the system (see Figure 2 on page 2).

The CICS licensed program (either CICS/OS/VS or CICS/DOS/VS) performs many functions essential to success in realtime DB/DC systems. Its major functions can be summarized as follows:

- Provision of rapid response to simultaneously active online terminals
- Control of a telecommunication network of mixed devices
- Management of a wide mixture of transactions being serviced by a variety of application programs at the same time
- Control of access to data files or to a data base
- Management of system resources, such as main storage, to keep the system in continuous operation
- Assignment of priorities to optimize use of the processor.

With these functions assumed by CICS, application programmers can concentrate on their particular applications. Programming takes less time, debugging is easier, and implementation time and costs are reduced accordingly.

A key consideration in selecting a DB/DC system is its adaptability to present and future needs. CICS is a family of systems that provides a DB/DC interface to IBM System/370 at most levels of the product line, offering a clear path for growth or migration of an installation.

Figure 3 on page 4 indicates how CICS supports the information needs of multiple applications, independently and concurrently.

Although application programmers need not be concerned with details of CICS structure or performance, they should have a general understanding of how CICS components interact to perform essential processing steps. CICS consists of six major components, explained in greater detail in the appropriate *CICS Diagnosis Reference* manual, as follows:

- System management
- System services
- System monitoring
- System reliability
- System support
- Application services.

Each of these **components** is divided into **functions** which provide **services** to CICS users. The components that most directly affect the application programmer are system management, system monitoring, and system reliability. To help the application programmer understand some of the ways in which CICS assists him, the system management functions are summarized below.

- Terminal management - provides for communication between terminals and user-written application programs through the terminal control

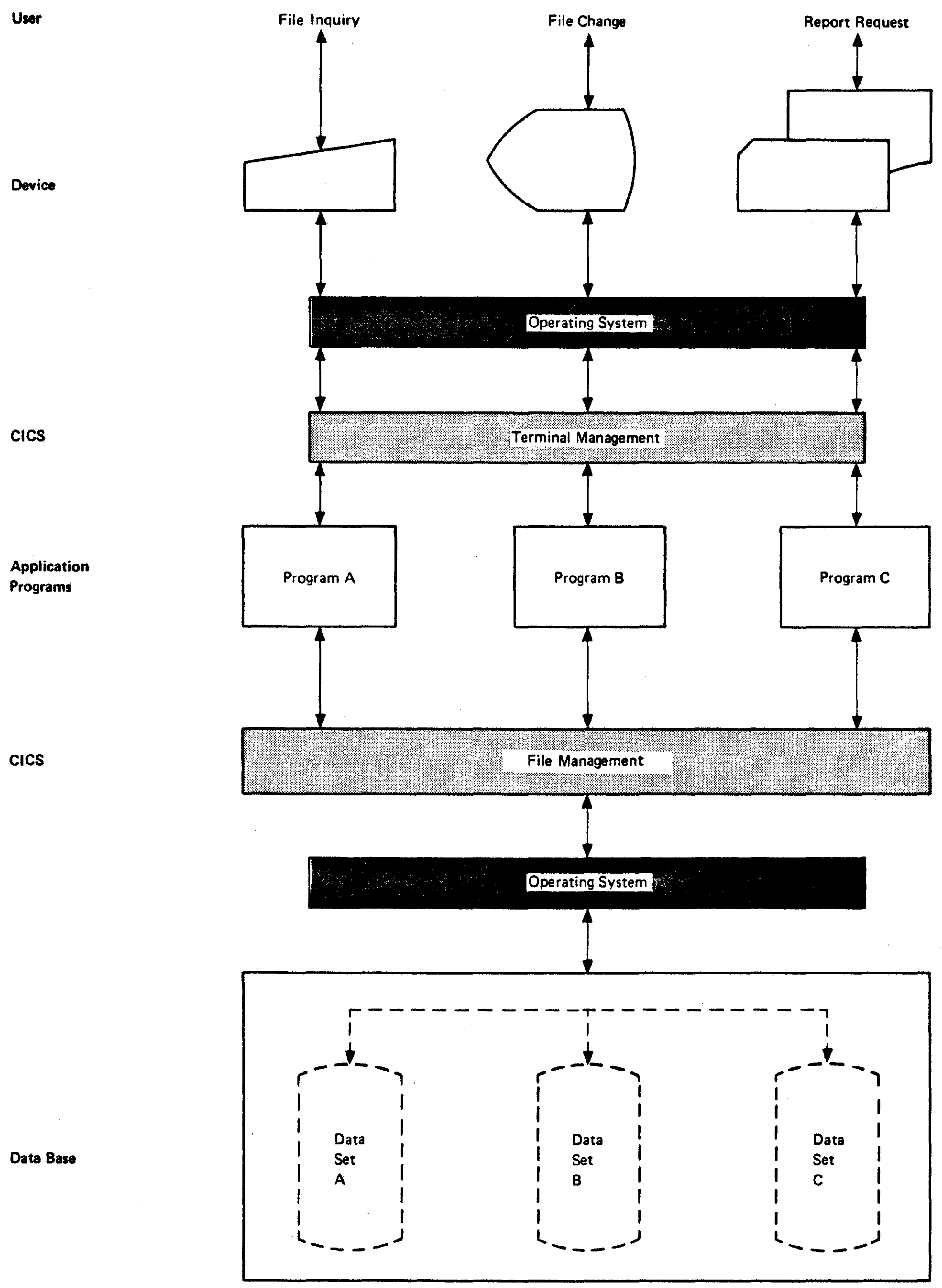


Figure 3. CICS Processing of Multiple Concurrent Transactions

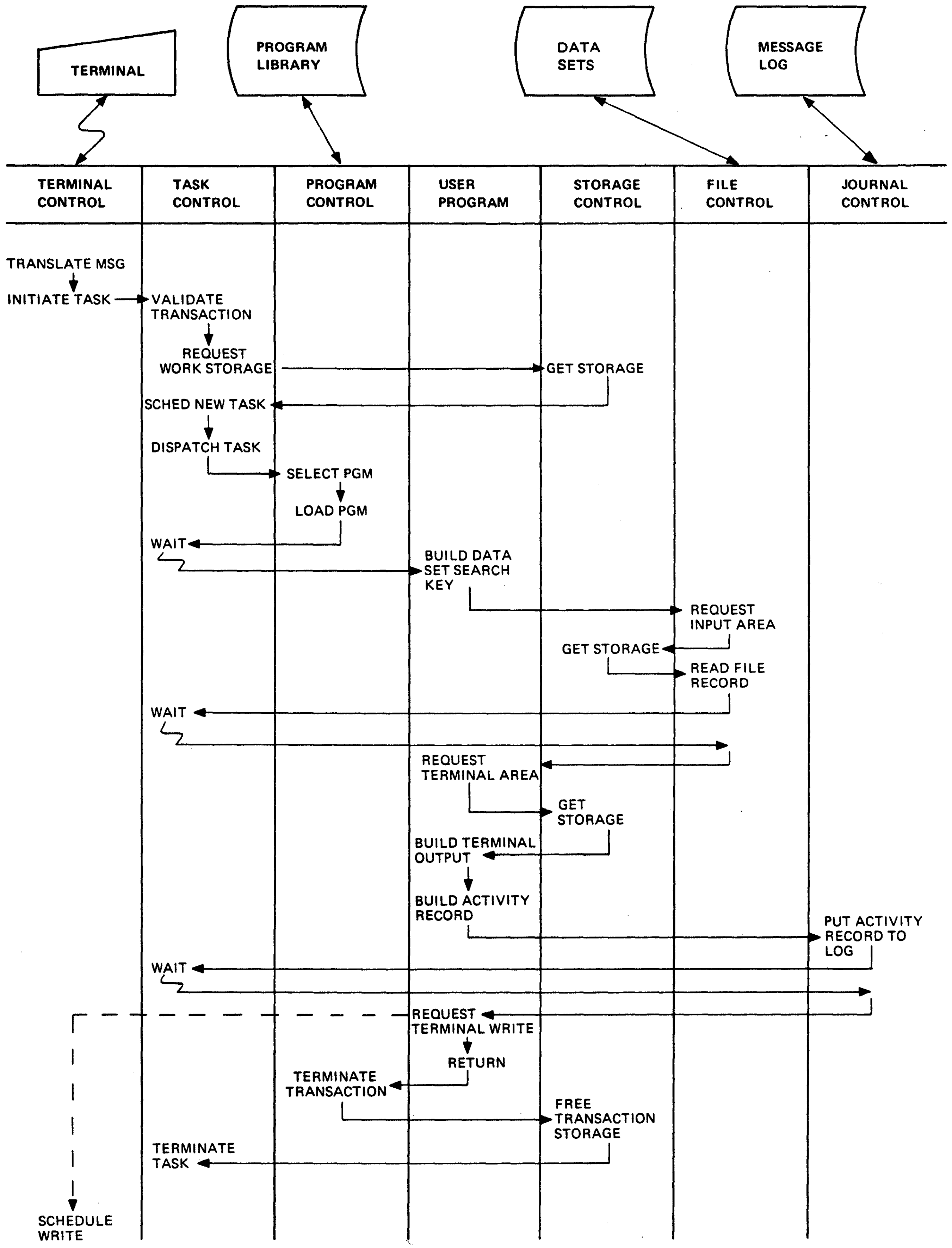


Figure 4. CICS Transaction Flow

program. This function supports automatic task initiation (ATI) to process new transactions. The testing of application programs is accommodated by the simulation of terminals by sequential devices such as card readers, line printers, tape units, or disk storage units.

- File management - provides for the addition, update, direct retrieval, and selective retrieval (browsing) of data on VSAM and BDAM data sets. Additional capabilities provided only for VSAM data sets include record deletion, skip-sequential processing, key-ordered mass insertion, relative byte addressing, search key high or equal, generic key, and locate mode processing for read-only requests.

  Optional access to the DL/I facility of the IBM Information Management System/Virtual Storage (IMS/VS) is provided under CICS/OS/VS. Such use of DL/I requires installation of the IBM licensed program IMS/VS Data Base System.

  Note: Users of CICS/DOS/VS can interface with the IBM licensed program DOS/VS DL/I through DOS/VS DL/I CALLs, but CICS file control macros cannot be used.

- Transient data management - provides for optional queuing of data in transit between user-defined destinations. This function facilitates message switching and data collection.

- Temporary storage management - provides an optional general-purpose "scratch pad" function intended for video display paging, broadcasting, data collection suspension, conservation of main storage, retention of control information, and similar. Where multiple records are used and random access to those records is required, this function also provides a queuing facility.

- Storage management - provides control of main storage allocated to CICS. Storage acquisition, disposition, initialization, and request queuing are among the services performed by this function.

- Program management - provides a multiprogramming capability through dynamic program management while offering a realtime program fetch capability.

- Time management - provides control of various task functions (for example, runaway task control, task synchronization, and system stall detection) based on specified intervals of time or the time of day.

- Task management - provides dynamic multitasking necessary for effective, concurrent transaction processing, such as priority scheduling, transaction synchronization, and control of serially reusable resources. This function controls activities within the CICS partition or region and is in addition to the multitasking or multiprocessing capabilities of the host operating system.

- Journal management - provides for the creation and management of special-purpose sequential data sets, called journals, during realtime execution of CICS. Journals are intended for recording (in chronological order) data that the user may need in subsequent reconstruction of data or events. Examples of such data sets are an audit trail, a change-file of data base updates and additions, and a record of system transaction activity (often called a log).

- Sync point management - works in conjunction with other CICS functions such as transient data management and file management, to provide for an emergency restart of CICS after abnormal termination. The CICS transaction backout program (DFHTBP) or a user-written application program can make changes to data base data sets or transient data intrapartition queues for tasks "in-flight" at time of failure based on information recorded on a system log during online execution of CICS.

CICS also provides dump management and trace management, which are used in program debugging. CICS basic mapping support (BMS) facilitates information display on a wide variety of terminals and provides device independence, terminal paging, and message routing. A number of built-in functions are available for use by application programs. CICS also provides system service programming to identify terminal operators, to give control of the entire system to a master terminal, to display realtime system statistics, to intercept abnormal conditions not handled directly by the operating system, and to end operation by collecting statistics, closing data sets, and returning control to the operating system.

To provide rapid response to terminal users, CICS executes in a multitasking mode of operation within its own partition or region. Such multitasking within a partition or region is analogous to multiprogramming within the total operating-system environment. Generally, tasks are initiated as a result of transactions entered at terminals. Whenever a task is forced to wait for completion of an I/O operation, availability of a resource, or some other cause, processing of another task within the system is initiated or continued.

The processing of a typical transaction is shown in Figure 4 on page 5. Some general characteristics of application programs to be run under CICS and the use of other functions that it provides are explained in subsequent parts of this publication.

CHAPTER 1.2. MACRO FORMAT AND SYNTAX NOTATION

Application programs to be executed under CICS can be written at the macro level in assembler language, COBOL, or PL/I. Regardless of the language used, it is strongly recommended that CICS is allowed to perform all supervisory and data management services for applications. Such services can be invoked by using CICS macros. CICS macros can also be used to request dump and trace facilities when testing or debugging an application program.

Although an application program is not precluded from direct communication with the operating system, the results of such action are unpredictable and performance may be affected. Such action also has a limiting effect on migration from CICS/DOS/VS to CICS/OS/VS, a growth path that may become highly advisable for the CICS/DOS/VS user.

CICS macros are written in a format similar to assembler language macros, as follows:

Name	Operation	Operands	Comments
Blank or Symbol	DFHxxxxx	One or more operands separated by commas	Program comments

The name field must not contain a label if the macro is used in a COBOL or PL/I program; however, if a label is desired for the macro, it may be placed on the line preceding the macro. For COBOL programs, the first six positions may contain a sequence number.

The operation field must begin before column 16 and must contain the three-character combination "DFH" in the first three positions of the operation field. Up to five additional characters can be appended to DFH to complete this symbolic name for the appropriate program or table. Since DFH is reserved for CICS macros, no other line may begin with this three-character combination.

The operands field is used to specify the services and options to be performed.

Note that, throughout the remainder of the manual, the displays of syntax are reduced to the essentials. The name and comments fields are not always shown, neither is the continuation character, which if needed, must appear in column 72.

The following general rules apply to the macros described in this manual:

1. Operands that are written in uppercase letters (for example, TYPE=INITIAL) are to be coded exactly as shown.

2. Operands that are written as a combination of uppercase and lowercase letters separated by an equal sign are to be coded with the keyword on the left as it appears and an appropriate substitution for the general class of elements on the right. For example, if the format description contains NORESP=symb-addr, the user may code NORESP=NORMROUT.

3. Commas and parentheses are coded as shown. However, the parentheses are required only when more than one operand is specified. For example, the following coding is correct:

   ```
   TYPE=READ
   TYPE=(READ,WAIT)
   ```

   The commas are used as separators, but no comma should precede the first operand entry or follow the last one inside parentheses. Similarly, no comma should follow the last operand coded for a particular macro.

4. Since a blank character indicates the end of the operand field, the operand field must not contain blanks except after a comma on a line to be continued or after the last operand of the macro. The first operand on a continuation line must begin in column 16.

5. When a CICS macro is coded on more than one line, each line containing part of the macro (except the last one) must contain a nonblank character (for example, an asterisk) in column 72 indicating that the macro has been continued on the next line.

6. If a macro that has positional operands is coded with an invalid operand, the operand will be ignored. An error message will not be issued.

7. If a keyword is spelled incorrectly, the operand **may** be treated as an invalid positional operand as in point 6.

8. The rules for writing CICS macro operands are the same as those for assembler language macros.

SYNTAX NOTATION

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

1. Braces { } are used to delimit parameters from which choices are made. For example,

   ARGTYP={<u>KEY</u>|RBA}

   which indicates that the coding ARGTYP= must be followed either by the keyword KEY or by the keyword RBA.

2. The "or" symbol | indicates that a choice is to be made. For example,

   [,INTRVAL={numeric value|YES}]

   means that either "numeric value" or "YES", but not both, can be specified in the macro.

3. Square brackets [ ] denote options. Anything enclosed in square brackets may or may not be coded, depending on whether or not the associated option is desired. For example,

   MODE=[{<u>MOVE</u>|LOCATE}]

   If a default value is assumed by CICS in the case of an omitted operand, that default value is indicated by underlining.

4. An ellipsis ... (three dots) denotes that the immediately preceding unit may appear one or more times in succession in the macro.

CHAPTER 1.3. PROGRAMMING TECHNIQUES AND RESTRICTIONS

Application programs to be run under CICS may be coded at the macro level in assembler language, COBOL, or PL/I. Writing a program to be run under CICS is not significantly different from writing a program to be run on any of numerous computing systems. However, the CICS user should be aware that CICS is an online system and that programs running under CICS operate in an online environment. Some of the basic differences between online systems and the traditional batch processing environment are summarized in Figure 5 on page 12.

Single threading is the execution of a program to process inputs to completion, sequentially. Processing of one input is completed before another input is acted upon.

In contrast, multithreading is the capability of using various sections of a single program concurrently. Under CICS, for example, the first section of an application program may be executing to process one transaction. When that section is completed (in general, signaled by the execution of a CICS macro that causes a wait), processing of another transaction using a different section of code in the same program may ensue.

Just as there is not usually one clearly superior, correct way to solve a problem, so there is not usually one correct way to write a program to implement that solution. Nevertheless, there are good and bad techniques of programming under CICS. How much time and thought should be given to programming style when writing a program? The answer depends largely on the expected usage of the program. Will it be used once a day or once a year? When used, will it run for two minutes or five hours? The frequency and length of use are important factors to consider when deciding how much time to spend on programming techniques (that is, to devising the optimum solution to a problem).

Some of the basic characteristics of application programs to be run under CICS are summarized below. These characteristics should be viewed as essential to successful operation under CICS (although some are not mandatory, they are highly advisable).

1. Programs must be quasi-reenterable. See "Quasi-Reenterability" on page 14.
2. CICS macros (rather than programming language statements such as GET, PUT, READ, or WRITE) are included to control the functions required in application programs. (See "Chapter 1.2. Macro Format and Syntax Notation" on page 9.)
3. Input/output areas, temporary storage areas, and work areas are not included in an application program. All or portions of these areas are defined outside of application programs. The application programmer must work with CICS system programmers in defining these areas by means of tables within CICS. (See "Storage Definition" on page 14, and in Part 2.)
4. Files are not defined within application programs. As in item 3, the application programmer works with CICS system programmers in establishing these definitions. (See the appropriate <u>CICS Resource Definition</u> manuals and the applicable operating system publications.)
5. The application programmer must establish addressability in his program to CICS storage areas accessed by his program.
6. Working storage should not be tied up, for example, awaiting a reply from a terminal user.
7. Programs should be as efficient as possible, to work with CICS in providing rapid response to terminals.
8. Any feature, option, or statement that will transfer control to the operating system should not be used in a CICS program.
9. Assembler language programmers should be aware that the program mask is undefined by CICS on entry to a program. It is the user's responsibility to set the program mask for any module that requires a specific value for the mask. CICS does not preserve the mask value across the interface to other called programs, for example, when the DFHPC TYPE=LINK or DFHPC TYPE=XCTL macro is used.
10. Care must be taken if a program involves lengthy calculations; since an application program retains control from one CICS macro to the next, processing of other

	Batch Processing	Online Application
Input	Generally sequential from cards, tape, or a direct access storage device (DASD); submitted as groups of related data, edited, and verified	Random, multiple, concurrent but unrelated entries from terminals; immediate edit and verification of each entry
Processing	Sequential, generally single-thread, with updating of sequential master files	Random, multithreading, as one aspect of multitasking within a partition or region; for inquiry or updating purposes or both
Output	Generally in the form of updated master files and printed reports	Messages to terminals updated files, and system log of activities
Sequence of operations	Start program Read transaction Read master Process	System is initialized then transactions are processed as they occur, with data rather than program as driver
End of job	Signaled by last transaction	Generally, end of shift or day
Amount of activity	Predictable, known before run	Not predictable, can fluctuate widely
Master files/ data sets	Applications "own" master files on tape or DASD; placed online when required for run	Files accessible to multiple, authorized applications; must be online; are on DASD
Response time	Varies widely; usually involves manual procedures	Measured in seconds; generally occurs as message to terminal

Figure 5. Comparison of Batch and Online Environments

transactions is completely excluded. However, the DFHKC TYPE=CHAP macro can be used to allow other transaction processing to proceed, refer to "Chapter 5.3. Task Control (DFHKC Macro)" on page 221 for details.

The general structure of a CICS application program can be summarized as follows:

- Storage definition statements
- Program initialization statements
- Processing statements
- Termination statements.

No attempt is made in this text to teach the use of typical programming language statements or general programming techniques within assembler language, COBOL, or PL/I. Documentation for these languages should be consulted for such information (see the Bibliography).

CICS operates in a virtual storage environment. The key objective of programming in this type of environment is the reduction of page faults (those cases in which a program refers to an instruction or data that does not reside in real storage). When this occurs, the page in virtual storage that contains the referenced instruction or data must be transferred (paged) into real storage. The more paging required, the lower the overall system performance.

The application programmer who writes programs to be run in a virtual storage environment should understand the following concepts:

- **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
- **Validity of reference** - the consistent reference to valid data. This ensures that few storage references retrieve useless data
- **Working set** - the number and combination of pages of a program needed for satisfactory performance (low paging rate) during a given period.

In general, the application program should use techniques to improve the locality and validity of reference and to minimize the size of the working set at any time during execution of the program, as follows:

1. To achieve locality of reference, processing should be sequential for both code and data, as far as possible.

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

    b. Define new data items as close as possible to the items that use them.

    c. Define arrays or other data structures in the order in which they will be referred to; refer to elements within structures in the order in which they are stored, for example, by row rather than by column when using PL/I.

    d. Separate error-handling or unusual-situation routines from the main section of a program; they should be subprograms.

    e. Subprograms that are short and used only once or twice (other than those in "d" above) should be coded inline in the calling program.

2. To achieve validity of reference.

    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.

3. To reduce the size of the working set, the amount of storage that a program refers to in a given period should be as low as possible.

    a. Write modular programs and then structure the modules according to frequency and anticipated time of reference.

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

When all page frames in a real storage environment are filled and another page must be loaded into storage, a page replacement operation is required. The operating system replaces first those pages that have not been referred to for the longest period of time. If a page to be replaced has been modified, that page must be paged out onto virtual storage before the required page can be read in. The more page-out operations required, the lower the overall performance of the system.

To avoid the necessity for page-out operations, the application program should be coded so that page-out operations are not required when a page containing a portion of the program must be replaced in real storage. The program need only avoid modifying instructions or data within itself. A program in which neither instructions nor data are modified is said to be reenterable. As noted earlier, programs to be run in a CICS environment must be quasi-reenterable. For performance reasons, it may be wise to make them truly reenterable programs.

The application program should not attempt to use overlays, that is, to incorporate paging techniques. System paging is automatic and generally more efficient.

APPLICATION PROGRAM PACKAGING

The design of an application program for a virtual environment is similar to the design of an application program in 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 less efficient than making the programs resident since a dynamic program will not share unused space on a page with another program.

The reference pattern of the application should touch the fewest concurrent pages during its execution.

1. The main line execution should be as straight a line as possible. The ideal program executes sequentially with no branch logic referencing beyond a small range of address space.

2. Literals and subroutines should be coded as close to their use as possible. This would include LTORG statements at appropriate locations in the program. Place constants that are used only once near to the place where they are used. Executed instructions should be near the EX instruction. Perform and GOTO routines should be placed near the caller.

3. Avoid use of COBOL EXAMINE or VARIABLE MOVE operations since these expand into subroutine executions.

4. Do not alter anything within the program module. An unchanged module is reenterable and is not paged out.

5. Use the TWA for changeable data during execution, that is counters, switches, parameter passing, basic mapping support output area (use BMS SAVE).

6. Do few or no user GETMAINs to minimize the task's reference pattern.

7. Avoid LINKs since it will cause a GETMAIN for an RSA and will search the PPT.

8. Try to keep the execution path straight line by using XCTL.

9. If specifying data for a CICS service request by explicitly assigning a value to a CICS field (for example, in the task control area), assign the value immediately prior to issuing the service request, with no other service requests intervening. Also, reassign the value immediately before issuing any subsequent request that needs it.

QUASI-REENTERABILITY

Application programs must be coded so that they are "serially reusable" between entry and exit points of the program. A serially reusable portion of an application program is executed by only one transaction at a time, and must initialize and/or restore any instructions or data that it alters within itself during execution. (It is recommended, however, that all applications be truly reenterable to minimize paging.) Entry and exit points coincide with the use of CICS macros, since an application program loses control only upon execution of a CICS macro.

This required quality of application programs written to run under CICS is called "quasi-reenterability," since the programs need not meet System/370 specifications for true reenterability. Quasi-reenterability allows a single copy of a user-written application program to be used to process several transactions concurrently, thereby reducing the number of copies of a program that must be in main storage.

Intermediate exits may be taken during execution of an application program. Such exits constitute a transfer of control from the program. All switches, data, and intermediate results needed upon subsequent return to the program must be retained in a unique storage area such as the transaction work area (TWA). The application programmer must provide that unique intermediate storage area by symbolically defining it in his program (as described in Part 2).

A serially reusable application program that has no intermediate exits also has the quality of quasi-reenterability.

STORAGE DEFINITION

The macro library supplied with CICS contains symbolic storage definitions of CICS control areas, work areas, and I/O areas. It is strongly recommended that the application programmer use these definitions rather than develop actual or direct displacements in his program. This protects the application program in the event of any relocation of CICS.

The assembler language programmer includes symbolic storage definitions in his program by means of assembler language COPY statements. For the PL/I programmer, the macro library contains numerous BASED structures, in the form of dummy control sections (DSECTS), that describe CICS control areas. These DSECTs are available to the user through the use of %INCLUDE statements. The COBOL programmer uses similar definitions through COPY statements in the linkage section of the data division of his application program. These definitions are discussed in Part 2.

PROGRAM INITIALIZATION

In the initialization section of the application program, the assembler language programmer must establish a symbolic base address for his program, because this is not done by CICS prior to entry. In doing so, he identifies a base register. R12 is reserved by CICS for the address of the task control area (TCA) for this task. R13 is reserved for the address of the common system area (CSA). Both these registers are initialized by CICS prior to entry and their contents must be preserved throughout execution of the program. For COBOL and PL/I, this preservation of registers is done automatically by CICS.

R15 through R11 are available to the user and their contents are preserved when a CICS macro is executed; the contents of R14 are destroyed whenever a CICS macro is executed. The contents of R1 are destroyed if parameters are specified on a DL/I call.

The different types of the DFHPC macro that can be issued to transfer control from or to an application program are listed in the left-hand column of Figure 6 on page 15. The status of all

At program entry because of:	Registers			
	15, and 0-11	12	13	14
Initial Program Entry	Unknown	TCA	CSA	User—program address
LINK	Registers of program issuing the LINK	TCA	CSA	User—program address
XCTL	Registers of program issuing the XCTL	TCA	CSA	User—program address
Following execution of:				
LOAD	Unchanged	TCA	CSA	Next sequential instruction
RETURN (issued by a linked-to program)	Unchanged (from point-of-view of program issuing the LINK)	TCA	CSA	Next sequential instruction

Figure 6. Register Usage under CICS

registers upon program entry or upon return to a program is as shown in the remaining columns.

Although R14 contains the program entry address, it is not advisable to use R14 as the base register since it is used by CICS to service requests for CICS supervisory and data management services.

RESTRICTIONS

There are language and other restrictions that the application programmer should be aware of when writing programs to be run under CICS.

ASM

The use of CICS macros in an assembler language application program precludes the use of the assembler instruction COM (define blank common control section).

COBOL

| Note that VS COBOL II is not supported at macro level.

The use of CICS macros in a COBOL application program precludes the use of the following:

1. The environment and data division entries normally associated with data management services.

2. The file section of the data division.

3. The special features:

   ACCEPT
   DISPLAY
   EXHIBIT
   REPORT WRITER
   SEGMENTATION
   SORT
   TRACE
   UNSTRING

   Any feature that requires an operating system GETMAIN (CICS/OS/VS only).

4. The COBOL compiler options (CICS/OS/VS only):

   COUNT
   DYNAM
   ENDJOB
   FLOW
   STATE
   SYMDMP
   SYST
   TEST

   The COBOL compiler options (CICS/DOS/VS only):

   COUNT
   FLOW
   STATE
   STXIT
   SYMDMP

   Any option that requires operating system services.

5. The COBOL statements:

   CLOSE
   OPEN
   READ
   WRITE

6. The QUOTE option, which signifies that literals are to be delineated by quotation marks (for example, "74"). Because CICS macros generate COBOL code using apostrophes to delineate literals (for example, '74'), the APOST option must be in effect.

7. The OPTIMIZE option of DOS Full COBOL Version 3 (5736-CB2)

SERVICE RELOAD statements must be coded in programs compiled under the following compilers when the OPTIMIZE option is active:

- OS Full COBOL Version 4 (5734-CB2)
- OS/VS COBOL Release 1 (5740-CB1)
- DOS/VS COBOL (5746-CB1)

If the NOOPTIMIZE option is used, SERVICE RELOAD can, but need not, be used. Further details of SERVICE RELOAD appear in "Additional Guidelines" in "Chapter 2.3. Storage Definition - COBOL" on page 35.

CICS macros should not be coded within a COBOL statement, since each COBOL statement generated by a CICS macro is terminated by a period.

CICS macros generate COBOL statements which use an apostrophe (') to delineate literals. Code written by the application programmer cannot utilize quotes (") to delineate literals.

Duplicate names may not be used. This requirement is a result of preprocessing by the translator before COBOL statements are generated.

Any COBOL program that is to run under CICS must contain at least one CICS macro (for example, DFHPC TYPE=RETURN) for proper operation.

Floating-point operations can be used, but CICS does not dump the contents of floating-point registers, and programmers should be aware that a floating-point interrupt will cause the task to be abnormally terminated.

Users of the OS/VS COBOL Release 2 compiler must specify LANGLVL(1), and must not use the INSPECT or USE FOR DEBUGGING statements. Note that all the COBOL examples throughout this manual have been written to language level 68 (specify LANGLVL(1)).

The macro level interface will not support a COBOL program with a TGT larger than 4K. If a program generates a TGT greater than 4K the command level interface must be used.

PL/I

| The use of CICS macros in a PL/I application program precludes the use of the following:

1. The PL/I multitasking built-in functions:

   COMPLETION
   PRIORITY
   STATUS

2. The PL/I multitasking options:

   EVENT
   PRIORITY
   TASK

3. The PL/I statements:

   CLOSE
   DELAY
   DELETE
   DISPLAY
   EXIT
   FETCH
   GET
   HALT
   LOCATE
   OPEN
   PUT
   READ
   RELEASE
   REWRITE
   STOP
   UNLOCK
   WRITE

4. PL/I Sort/merge.

5. PL/I error handling.

6. A declaration for a nonstring element variable with the attributes STATIC EXTERNAL but without the INITIAL attribute. (This declaration will generate a common CSECT that cannot be handled by CICS).

7. The PL/I compiler options:

   FLOW
   GONUMBER
   GOSTMT
   REPORT

Refer to the appropriate <u>PL/I Optimizing Compiler Programmer's Guide</u> for more information on the applicable restrictions.

An application program written in PL/I must consist of an external (MAIN) procedure. Procedure CALLs (both internal and external) are allowed in a PL/I program to be run under CICS.

Floating-point operations can be used, but CICS does not dump the contents of floating-point registers, and programmers should be aware that a floating-point interrupt will cause the task to be abnormally terminated.

Any CICS macro operand which defines a name or label of a storage area or routine should comply with the assembler language restrictions of 8 characters or less. This requirement is a result of preprocessing by the assembler before PL/I statements are generated.

LINK-EDITING

Separate COBOL routines cannot be link-edited together. Neither can separate PL/I routines. Assembler language routines may be link-edited, but routines invoked by CALL statements must conform to CICS application program requirements. Facilities comparable to link-editing are provided under CICS through DFHPC TYPE=LINK and DFHPC TYPE=XCTL (transfer control) macros, which can be used to set up communication between programs. For details of the job control required to compile and link-edit application programs refer to the appropriate CICS Installation and Operations Guide.

OBJECT PROGRAM SIZE

The object module resulting from any application program must not occupy more than 524152 bytes of main storage.

ENTRY POINT ADDRESS

For all programs, the entry point address must be less than 32768 bytes from the load point.

BMS MAP SIZE

The load module of a BMS map that is loaded dynamically using the DFHPC TYPE=LOAD macro must not exceed 65520 bytes.

ASSEMBLY-TIME SERVICE (DFHCOVER MACRO)

In addition to knowing the execution-time considerations discussed in this chapter, the application programmer should be aware of an assembly-time (or compile-time) service available under CICS: the DFHCOVER macro. This macro specifies that the assembler or compiler in use print a cover page on two consecutive pages, which ensures that the application program listing can be torn off with one of the cover pages face up. Useful information (program name, date, time of assembly, remarks, and so on) may then be written on the cover page.

The DFHCOVER macro requires no operands and nothing else should appear on the same coding line.

If the DFHCOVER macro is coded as part of an assembler language application program, it should be coded as the first instruction in the program. If desired, however, this macro may be coded after anything that is not vital to the listing (such as the TITLE line).

The first line of a PL/I source program is printed as a header on each page of the source listing. This means that when the DFHCOVER macro is part of a PL/I application program, the first line should be a comment containing information that the application programmer wants printed as a header. The second line should contain the DFHCOVER macro. The actual PL/I code should begin with the third line.

Since column 1 is used by the DFHCOVER macro for line and page spacing under PL/I, column 1 must be defined as reserved for control characters and columns 2-72 must be defined as available for data. For information concerning PL/I compile-time services, see the appropriate PL/I Optimizing Compiler Programmer's Guide.

The example in Appendix A shows how the DFHCOVER macro is used.

TESTING RESPONSES TO MACROS

As a result of issuing CICS macros, certain error conditions may be raised. A programmer can test for these conditions in any of the following ways:

- Code the appropriate operands in the macro being issued. Each macro syntax display lists the operands available.
- Code a DFHXX TYPE=CHECK macro immediately following the particular macro by which the service is requested.
- Code instructions, following the macro by which the service was requested, that test the contents of certain CICS control areas. The relevant control areas and the meaning of the returning bit patterns are discussed in each chapter that describes the services.

If the programmer does not check the response to a request, program flow continues with the next sequential instruction.

PART 2. STORAGE DEFINITION

CHAPTER 2.1. INTRODUCTION TO STORAGE DEFINITION

CICS provides symbolic storage definitions in the form of dummy sections (DSECTs) that describe the layouts of a number of storage areas. These storage definitions are contained in the CICS libraries and can be copied into application programs where, in combination with user-defined layouts of the user's sections of the storage areas, they provide symbolic addressing (addressability) to the storage areas.

CICS STORAGE AREAS

The storage areas for which symbolic storage definitions are provided consist of control areas, for example the Common System Area (CSA), work areas, for example the File Work Area (FWA), and input/output areas, for example the Terminal Input/Output Area (TIOA). CICS storage areas are summarized in Figure 7 on page 22. CICS system sections are shown in Figure 8 on page 23.

Some of these storage areas are acquired by CICS during system initialization, others are acquired and released during execution of the system. Some areas are acquired by CICS; some by the application program; and some by either CICS or the application program.

All CICS storage areas, with the exception of the journal control area (JCA) and VSAM work areas (VSWAs), consist of two sections. The first is the **system section**, used primarily by CICS; the second is the **user section**, defined and used exclusively by the application program. This division exists whether the storage areas are acquired during system initialization (for example, the CSA) or acquired during execution (for example, a TIOA)

You should consult the appropriate CICS Data Areas manual (which is the authoritative source) for the sizes of these storage areas. These sizes differ according to the release level of your CICS system.

All CICS pointers (areas containing addresses) are 4 bytes in length.

A storage accounting field comprising 8 bytes preceding and 8 bytes following each storage area is built by CICS for every storage area acquired for the user. If this field is altered or destroyed, CICS may be abnormally terminated.

The common system area (CSA) and the task control area (TCA) must be defined in every application program; other areas are defined as needed. It is the user's responsibility to define the CSA and TCA as well as other storage areas required by the application program.

Identifiers such as CSA and TCA, used in this manual, are also used in symbolic names, or labels, within CICS modules and must be used by the application programmer to refer to the data that they represent. Names of fields within a storage area generally begin with the characters of the label for that area. For example, TCA stands for Task Control Area, TCAFCAAA is a field in the TCA that points to a Facility Control Area, TCASCSA is a field in the TCA that points to a Storage Control Storage Area, and so on.

The letters A through G in Figure 7 on page 22 denote the following:

A Assembler language only.

B The TCAFCAAA may also point to the address of a DCT entry or to the address of an automatic initiate descriptor.

C COBOL equivalent:

```
01 DFHTCTTE COPY DFHTCTTE.
MOVE TCAFCAAA TO TCTTEAR.
```

PL/I equivalent:

```
%INCLUDE DFHTCTTE;
```

D EOB = End of block.

E TCAFCAA, TCATSDA, and TCATDAA: The same location within the TCA is used for these 3 pointers, only one of which is current at any given time.

F TCASCSA may also point to an area to be released by a DFHSC TYPE=FREEMAIN macro.

G After a DFHPC TYPE=LOAD macro, TCAPCLA points to the beginning address of the loaded program.

Throughout Figure 7, the characters "LLbb" represent a 4-byte field in which the first 2 bytes define the length of the area ("bb" represent 2 blank characters).

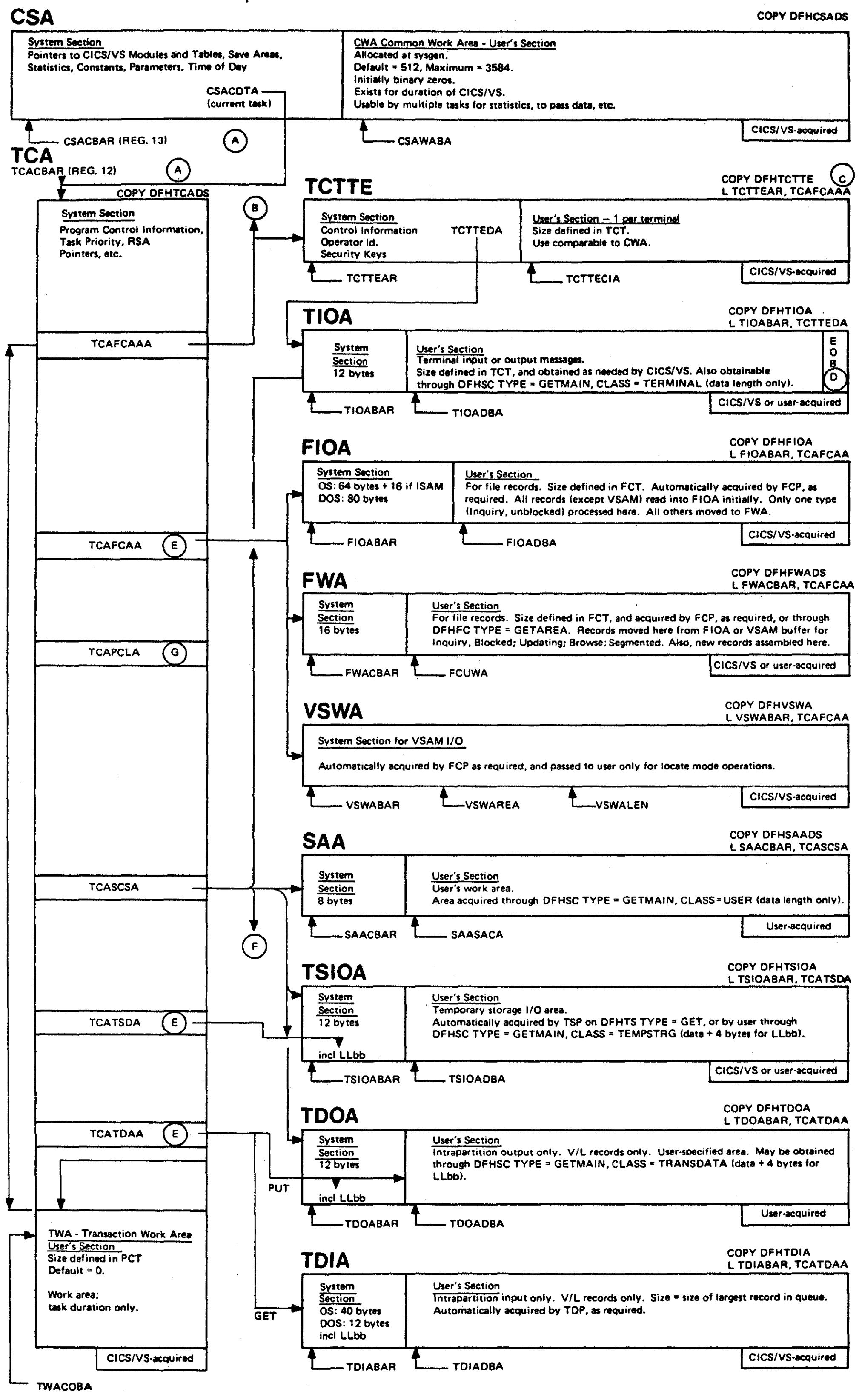
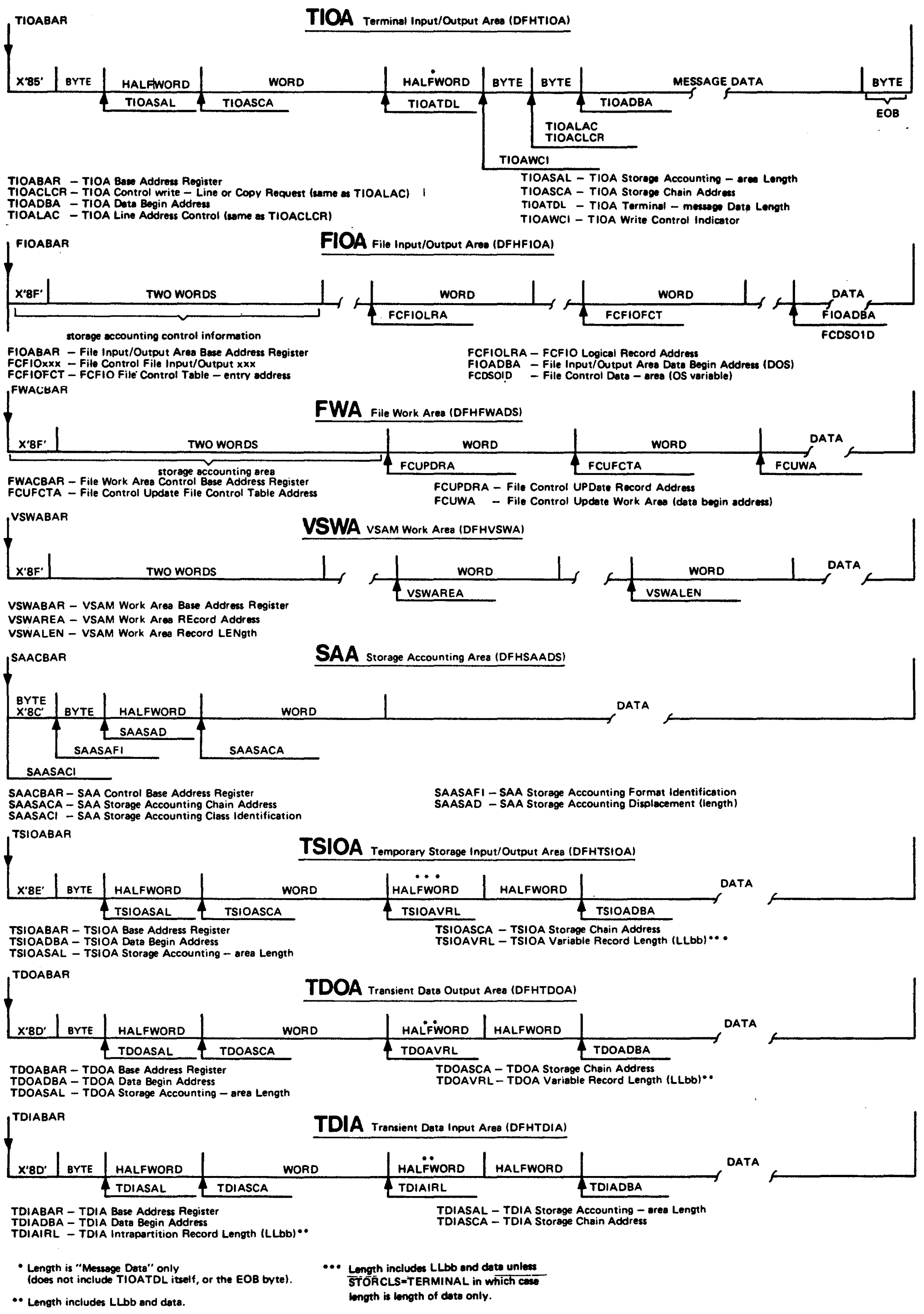

CSA
COPY DFHCSADS
System Section
Pointers to CICS/VS Modules and Tables, Save Areas, Statistics, Constants, Parameters, Time of Day
CSACDTA (current task)
CWA Common Work Area - User's Section
Allocated at sysgen.
Default = 512, Maximum = 3584.
Initially binary zeros.
Exists for duration of CICS/VS.
Usable by multiple tasks for statistics, to pass data, etc.
CICS/VS-acquired
CSACBAR (REG. 13)
A
CSAWABA
TCA
TCACBAR (REG. 12)
A
COPY DFHTCADS
System Section
Program Control Information, Task Priority, RSA Pointers, etc.
B
TCTTE
COPY DFHTCTTE
L TCTTEAR, TCAFCAAA
C
System Section
Control Information
Operator Id.
Security Keys
TCTTEDA
User's Section – 1 per terminal
Size defined in TCT.
Use comparable to CWA.
CICS/VS-acquired
TCTTEAR
TCTTECIA
TIOA
COPY DFHTIOA
L TIOABAR, TCTTEDA
TCAFCAAA
System Section 12 bytes
User's Section
Terminal input or output messages.
Size defined in TCT, and obtained as needed by CICS/VS. Also obtainable through DFHSC TYPE = GETMAIN, CLASS = TERMINAL (data length only).
E O B
D
CICS/VS or user-acquired
TIOABAR
TIOADBA
FIOA
COPY DFHFIOA
L FIOABAR, TCAFCAA
System Section
OS: 64 bytes + 16 if ISAM
DOS: 80 bytes
User's Section
For file records. Size defined in FCT. Automatically acquired by FCP, as required. All records (except VSAM) read into FIOA initially. Only one type (Inquiry, unblocked) processed here. All others moved to FWA.
CICS/VS-acquired
FIOABAR
FIOADBA
TCAFCAA
E
FWA
COPY DFHFWADS
L FWACBAR, TCAFCAA
System Section 16 bytes
User's Section
For file records. Size defined in FCT, and acquired by FCP, as required, or through DFHFC TYPE = GETAREA. Records moved here from FIOA or VSAM buffer for Inquiry, Blocked; Updating; Browse; Segmented. Also, new records assembled here.
CICS/VS or user-acquired
TCAPCLA
G
FWACBAR
FCUWA
VSWA
COPY DFHVSWA
L VSWABAR, TCAFCAA
System Section for VSAM I/O
Automatically acquired by FCP as required, and passed to user only for locate mode operations.
CICS/VS-acquired
VSWABAR
VSWAREA
VSWALEN
SAA
COPY DFHSAADS
L SAACBAR, TCASCSA
TCASCSA
System Section 8 bytes
User's Section
User's work area.
Area acquired through DFHSC TYPE = GETMAIN, CLASS=USER (data length only).
User-acquired
F
SAACBAR
SAASACA
TSIOA
COPY DFHTSIOA
L TSIOABAR, TCATSDA
System Section 12 bytes
incl LLbb
User's Section
Temporary storage I/O area.
Automatically acquired by TSP on DFHTS TYPE = GET, or by user through DFHSC TYPE = GETMAIN, CLASS = TEMPSTRG (data + 4 bytes for LLbb).
TCATSDA
E
CICS/VS or user-acquired
TSIOABAR
TSIOADBA
TDOA
COPY DFHTDOA
L TDOABAR, TCATDAA
TCATDAA
E
System Section 12 bytes
incl LLbb
PUT
User's Section
Intrapartition output only. V/L records only. User-specified area. May be obtained through DFHSC TYPE = GETMAIN, CLASS = TRANSDATA (data + 4 bytes for LLbb).
User-acquired
TDOABAR
TDOADBA
TWA - Transaction Work Area
User's Section
Size defined in PCT
Default = 0.
Work area;
task duration only.
CICS/VS-acquired
TWACOBA
TDIA
COPY DFHTDIA
L TDIABAR, TCATDAA
GET
System Section
OS: 40 bytes
DOS: 12 bytes
incl LLbb
User's Section
Intrapartition input only. V/L records only. Size = size of largest record in queue.
Automatically acquired by TDP, as required.
CICS/VS-acquired
TDIABAR
TDIADBA

Figure 7. Summary of CICS Storage Areas



TIOABAR – TIOA Base Address Register
TIOACLCR – TIOA Control write – Line or Copy Request (same as TIOALAC)
TIOADBA – TIOA Data Begin Address
TIOALAC – TIOA Line Address Control (same as TIOACLCR)
TIOASAL – TIOA Storage Accounting – area Length
TIOASCA – TIOA Storage Chain Address
TIOATDL – TIOA Terminal – message Data Length
TIOAWCI – TIOA Write Control Indicator

FIOABAR – File Input/Output Area Base Address Register
FCFIOxxx – File Control File Input/Output xxx
FCFIOFCT – FCFIO File Control Table – entry address
FCFIOLRA – FCFIO Logical Record Address
FIOADBA – File Input/Output Area Data Begin Address (DOS)
FCDSOID – File Control Data – area (OS variable)

FWACBAR – File Work Area Control Base Address Register
FCUFCTA – File Control Update File Control Table Address
FCUPDRA – File Control UPDate Record Address
FCUWA – File Control Update Work Area (data begin address)

VSWABAR – VSAM Work Area Base Address Register
VSWAREA – VSAM Work Area REcord Address
VSWALEN – VSAM Work Area Record LENgth

SAACBAR – SAA Control Base Address Register
SAASACA – SAA Storage Accounting Chain Address
SAASACI – SAA Storage Accounting Class Identification
SAASAFI – SAA Storage Accounting Format Identification
SAASAD – SAA Storage Accounting Displacement (length)

TSIOABAR – TSIOA Base Address Register
TSIOADBA – TSIOA Data Begin Address
TSIOASAL – TSIOA Storage Accounting – area Length
TSIOASCA – TSIOA Storage Chain Address
TSIOAVRL – TSIOA Variable Record Length (LLbb)***

TDOABAR – TDOA Base Address Register
TDOADBA – TDOA Data Begin Address
TDOASAL – TDOA Storage Accounting – area Length
TDOASCA – TDOA Storage Chain Address
TDOAVRL – TDOA Variable Record Length (LLbb)**

TDIABAR – TDIA Base Address Register
TDIADBA – TDIA Data Begin Address
TDIAIRL – TDIA Intrapartition Record Length (LLbb)**
TDIASAL – TDIA Storage Accounting – area Length
TDIASCA – TDIA Storage Chain Address

* Length is "Message Data" only (does not include TIOATDL itself, or the EOB byte).

** Length includes LLbb and data.

*** Length includes LLbb and data unless STORCLS=TERMINAL in which case length is length of data only.

Figure 8. CICS System Sections

Copying Symbolic Storage Definitions

Depending on the programming language used, a statement of one of the forms shown below is required to copy a symbolic storage definition into an application program.

1. Assembler language COPY statement of the form:

   ```
   COPY name
   ```

2. COBOL COPY statement of the form:

   ```
   01 name COPY name.
   ```

   specified in the linkage section of the data division.

3. PL/I preprocessor statement of the form:

   ```
   %INCLUDE library(member);
         or
   %INCLUDE member;
   ```

For example, assume that one or more TIOAs are to be acquired during program execution. One of the statements below must be included:

```
ASM:    COPY DFHTIOA

COBOL:  01 DFHTIOA COPY DFHTIOA.

PL/I:   %INCLUDE DFHTIOA;
```

This statement copies the storage definition as a description or map of the storage area into the application program, but does not acquire storage for it. As pointed out above, sometimes CICS acquires the area; in other cases, the user acquires it.

Addressability

The storage definition that has been copied into the application program must be mapped over the storage area acquired. This is done by moving the address of the area (stored in a particular location by CICS) into a base locator for that area. Addressability through this base locator is limited to 4096 (0 through 4095) bytes. Depending on the programming language, a statement of one of the following forms will normally be used to establish addressability to the area:

1. Assembler language statement of the form:

   ```
   L  base-locator,
        location-containing-address
   ```

2. COBOL statement of the form:

   ```
   MOVE location-containing-address
   TO base-locator.
   ```

3. PL/I based pointer assignment of the form:

   ```
   base-locator=
       location-containing-address;
   ```

For example, assume that a TIOA has been acquired during program execution. TCASCSA is a 4-byte field in the TCA that contains the address of the storage area that has been acquired. TIOABAR is the TIOA base address register. One of the statements below must be executed:

```
ASM:    L TIOABAR,TCASCSA

COBOL:  MOVE TCASCSA TO TIOABAR.

PL/I:   TIOABAR=TCASCSA;
```

Figure 9 on page 25 contains the names used in copying CICS-provided symbolic storage definitions into an application program and the names that represent base addresses used in establishing addressability. These symbolic names are used in Figure 7 on page 22 and Figure 8 on page 23, which show how the areas are related and give a summary of the contents of each area.

Chaining of CICS Storage Areas

Storage acquired by the application program through CICS storage management is controlled by chaining together all storage associated with a task. This chaining allows CICS to release all storage associated with the task, either upon request from the user or when the task is terminated, normally or abnormally.

The CSA, whose address is provided by CICS, points to the TCA which in turn points to the other storage areas required by the task. The TCA is the head of the chain of storage associated with each task, except for TIOAs, which are chained from the TCTTE. Figure 10 on page 26 illustrates the chaining of CICS storage areas and indicates the symbolic base address used to locate each storage area.

Required Storage Areas

At least two storage area definitions, namely, those for the CSA and the TCA, are required in every application program to be run under CICS. The following sections describe these areas. Services performed by CICS components are mentioned as necessary. Some tables that are basic to CICS operation are also mentioned. These tables are explained in greater detail in the appropriate <u>CICS Resource Definition</u> manual.

Storage Area	Symbolic Name for Defined Storage	Base Locator or Base Address Register	General Purpose Register Assignment
Common System Area (CSA)	DFHCSADS	CSACBAR	13
Task Control Area (TCA)	DFHTCADS	TCACBAR	12
Terminal Control Table Terminal Entry (TCTTE)	DFHTCTTE	TCTTEAR	*
Terminal Input/Output Area (TIOA)	DFHTIOA	TIOABAR	*
File Input/Output Area (FIOA)	DFHFIOA	FIOABAR	*
File Work Area (FWA)	DFHFWADS	FWACBAR	*
VSAM Work Area (VSWA)	DFHVSWA	VSWABAR	*
Storage Accounting Area (SAA)	DFHSAADS	SAACBAR	*
Temporary Storage Input/Output Area (TSIOA)	DFHTSIOA	TSIOABAR	*
Transient Data Output Area (TDOA)	DFHTDOA	TDOABAR	*
Transient Data Input Area (TDIA)	DFHTDIA	TDIABAR	*
Journal Control Area (JCA)	DFHJCADS	JCABAR	*
* Any register except 12, 13, or 14 (which are used by CICS) or 0 (which cannot be used as a base or index register)			

Figure 9. Symbolic Names and Base Addresses of CICS Storage Areas

COMMON SYSTEM AREA (CSA)

The Common System Area (CSA) contains areas and data required for the operation of CICS. It can be extended to include a user-defined common work area (CWA) that can be referred to by application programs.

Data in the CSA that is required for the operation of CICS includes:

- CICS save areas
- Addresses of CICS management programs
- Control system and user statistics accumulators
- Addresses of CICS system control tables
- Common system constants
- System control parameters.

Common Work Area (CWA)

The Common Work Area (CWA) is an area within the CSA that can be used by application programs for user data that needs to be accessed by any task in the system. This area is acquired during system initialization and its size is determined by the system programmer at system generation. It is initially set to binary zeros. Its contents can be accessed and altered by any task during CICS operation.

Addressability for the CWA is provided when copying the CICS storage definition for the CSA. However, addressability is limited to a combined total of 4096 (0 through 4095) bytes for the CSA and CWA. Addressability for any portion of the CWA extending beyond the 4096-byte limit is the responsibility of the user.

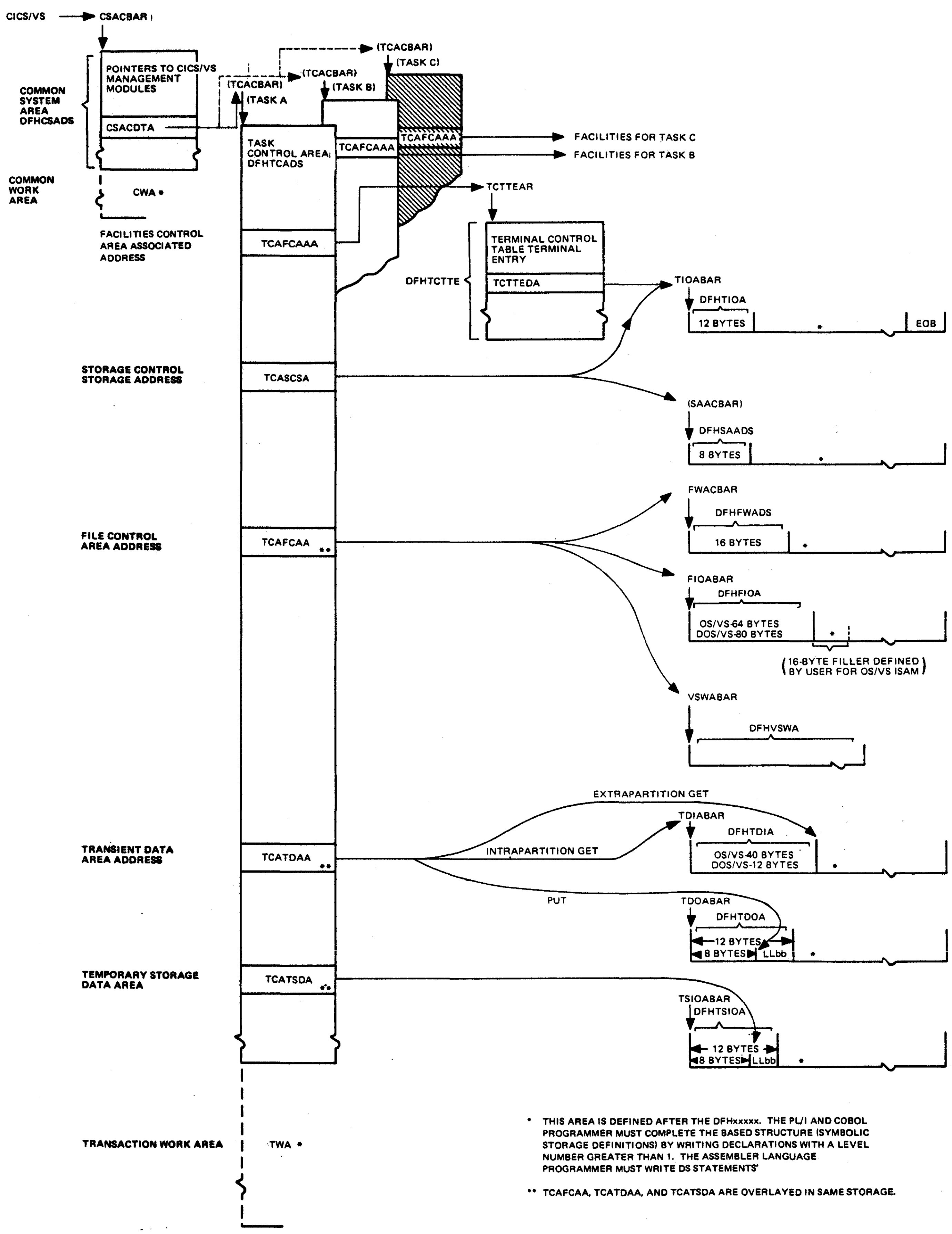
Since the CWA is available to any task while it has control of the system, it is not advisable for an application program to use this area for retention of data when requesting CICS services; instead, it would be better to use the transaction work area (TWA) which has been designed to be used by individual tasks for their own purposes. The TWA is described later in the chapter.

TASK CONTROL AREA (TCA)

The Task Control Area (TCA) is an area of main storage acquired by CICS when a task is initiated by the task control program. Once acquired, the TCA exists until the task is terminated. It contains the current status of the task, its relative dispatching priority, and parameters and information being passed between CICS and the application program. During execution of the task, the user can change the priority through task management services; further processing of the task is scheduled accordingly.

The TCA provides the following items for its associated task:

- Register save areas
- Unique fields (parameter areas) for communicating requests to CICS
- Address of the related facility control area (FCA)



* THIS AREA IS DEFINED AFTER THE DFHxxxxx. THE PL/I AND COBOL PROGRAMMER MUST COMPLETE THE BASED STRUCTURE (SYMBOLIC STORAGE DEFINITIONS) BY WRITING DECLARATIONS WITH A LEVEL NUMBER GREATER THAN 1. THE ASSEMBLER LANGUAGE PROGRAMMER MUST WRITE DS STATEMENTS'

** TCAFCAA, TCATDAA, AND TCATSDA ARE OVERLAYED IN SAME STORAGE.

Figure 10. Chaining of CICS Storage Areas

- Task storage chain addresses.

The TCA makes no provision for residual data such as statistics. However, the TCA can be extended to include a transaction work area (TWA), the size of which is determined by the user to meet the needs of the transaction. (See "Transaction Work Area", below.)

The TCA consists of 3 parts:

- CICS system section
- Communication section
- Optional transaction work area (TWA).

The CICS system section contains addresses and data needed by CICS to control the task. Access to this section is limited to CICS management and service programs.

The communication section is used by CICS and by user-written application programs for communication between the application program and CICS management and service programs. CICS functions sometimes overwrite some of the fields in the communication section of the TCA. The assignment of required fields in the TCA for a particular CICS request must therefore be done immediately prior to issuing the request, with no other requests intervening.

The optional transaction work area is reserved for the exclusive use of the application program.

In those cases in which a task is initiated from a terminal (nearly always the case), CICS places into the TCA the address of the terminal control table terminal entry (TCTTE) associated with the terminal. The TCTTE, in turn, contains the address of the TIOA.

Transaction Work Area (TWA)

The Transaction Work Area (TWA) is an extension of the TCA and is created at the option of the user to provide a work area for a given task. The TWA can be used for the accumulation of data and intermediate results during the execution of the task. It can also be used when the amount of working storage for a task is relatively static, when data must be passed between user-written application programs, or when data must be accessed by different programs during transaction processing. During multiple entries of data for a transaction, the application programs might retain the data in the TWA. This approach cannot be used for multiple transactions; the TWA is released automatically at task termination.

The size of the TWA for the task must be determined by the application designer and must be specified in the PCT by the system programmer at system generation. The TWA must be defined immediately following the definition of the TCA in the application program. The sizes of TWAs within the system vary according to the needs of the transaction. The TWA is initially set to binary zeros. For a discussion about establishing the TWA, see the explanation of the program control table in the appropriate CICS Resource Definition manual.

Addressability of the TWA is provided when copying the CICS storage definition for the TCA. However, addressability is limited to a combined total of 4096 (0 through 4095) bytes for the TCA and TWA. Addressability for any portion of the TWA extending beyond the 4095-byte limit is the responsibility of the user.

CHAPTER 2.2. STORAGE DEFINITION - ASSEMBLER LANGUAGE

The assembler language programmer must define storage for the CICS control areas and any other storage areas required for the processing of the application program. This is done by using the assembler language COPY statement to (1) copy the appropriate symbolic storage definitions into the application program and (2) specify the names of the storage areas being defined. All registers are available, except R12, R13, and R14 (which are used by CICS).

All application programs must contain statements to copy the symbolic storage definitions for the CSA and the TCA. If a terminal is to be used, the storage definition of a TCTTE must be copied also. The expansions of the CICS macros used in an application program refer to fields within these areas, so their locations must be identified. Whether additional definitions must be copied depends on the processing requirements (storage areas and macros used) of the application program.

STORAGE DEFINED DURING INITIALIZATION

During CICS initialization, the CSA is allocated as part of the CICS nucleus. For each terminal that is to be used, a terminal control table terminal entry (TCTTE) must be included in the TCT.

COMMON SYSTEM AREA (CSA)

The statement

```
  COPY DFHCSADS
```

copies the symbolic storage definition for the CSA and assigns R13 as its base register.

If CICS is generated to include a CWA, a symbolic definition for that area must be included immediately following the COPY DFHCSADS statement. In the following example, a total of 16 bytes of storage are defined by the three DS statements. It is assumed that a CWA of at least 16 bytes has been defined.

```
           COPY  DFHCSADS
  BUCKET1  DS    F
  BUCKET2  DS    F
  TEMPNAME DS    CL8
```

TERMINAL CONTROL TABLE TERMINAL ENTRY (TCTTE)

The statement

```
  COPY DFHTCTTE
```

copies the symbolic storage definition for the TCTTE. This definition can be used to obtain the address of the current TIOA (the current terminal control table terminal entry data address, or TCTTEDA) or to request a terminal control service via the DFHTC macro. An EQU statement must be included to set up a base register for the TCTTE, equating the label TCTTEAR to a general-purpose register. Addressability must also be established for the TCTTE by loading the address at TCAFCAAA into TCTTEAR. The following is an example of the coding required:

```
  TCTTEAR EQU  5
          COPY DFHTCTTE
          .
          .
          .
          L    TCTTEAR,TCAFCAAA
```

STORAGE DEFINED DURING EXECUTION

During execution of a task, the TCA, TIOA, and other storage areas required by the task are allocated by CICS storage management upon request from either the application program or CICS. The application program must include symbolic storage definitions for these storage areas by using COPY statements as described below.

TASK CONTROL AREA (TCA)

The statement

```
  COPY DFHTCADS
```

copies the symbolic storage definition for the communication section only of the TCA and assigns R12 as the base register for the whole of the TCA. If the application program requires the use of a TWA, DS statements for the TWA must immediately follow the COPY statement. The following is an example of the coding required to symbolically define storage for both the TCA and TWA. In the example, a total of 53 bytes of storage are defined by the four DS statements. It is assumed that a TWA of at least 53 bytes has been defined in the PCT for the transaction.

```
          COPY DFHTCADS
  NAME    DS   CL20
  STREET  DS   CL20
  CITY    DS   CL10
  STATE   DS   CL3
```

TERMINAL INPUT/OUTPUT AREA (TIOA)

The statement

```
  COPY DFHTIOA
```

copies the symbolic storage definition for the CICS system section of the TIOA. This storage definition should precede the user's definition of a terminal input or output message. The user must code an EQU statement to set up a base register for the TIOA, equating the label TIOABAR to a general-purpose register. Any action that requires a TIOA can then be specified. For example, a DFHSC TYPE=GETMAIN macro requesting CICS storage control to obtain dynamic storage for a TIOA for the program can be specified, as follows:

```
  TIOABAR EQU   9
          COPY  DFHTIOA
  NAME    DS    CL20
  STREET  DS    CL20
          DS    CL5
          .
          .
          .
          DFHSC TYPE=GETMAIN,NUMBYTE=45
                ,CLASS=TERMINAL
          L     TIOABAR,TCASCSA
```

For additional information about obtaining storage, see "Obtain and Initialize Main Storage (TYPE=GETMAIN)" in "Chapter 5.5. Storage Control (DFHSC Macro)" on page 241.

FILE INPUT/OUTPUT AREA (FIOA)

The statement

```
  COPY DFHFIOA
```

copies the symbolic storage definition for the CICS system section of the FIOA. This storage definition should precede the user's defined layout of an FIOA when reading an unblocked record without updating, or when reading DAM blocked records without deblocking. If desired, the user can identify that the area returned in response to a user file request is an FIOA, rather than an FWA or VSWA, by testing label FIOAIND for a mixed condition using mask FIOAM.

The user must code an EQU statement to set up a base register for the FIOA, equating the label FIOABAR to a general-purpose register. The FIOA is automatically acquired by CICS file management whenever a request is made by the user to access a data base data set. For CICS/OS/VS only, if data is | retrieved using an existing ISAM | application in ISAM compatibility mode, the FIOA must include a 16-byte filler prior to the user's data definition. The user must establish addressability for an FIOA acquired in response to a DFHFC macro before referring to the FIOA. The following is an example of the coding required; it includes the optional test (TM and BM instructions) for FIOA identification.

```
  FIOABAR EQU   7
          COPY  DFHFIOA
  NAME    DS    CL20
  STREET  DS    CL5
          .
          .
          .
          L     FIOABAR,TCAFCAA
          TM    FIOAIND,FIOAM
          BM    GOTFIOA
```

FILE WORK AREA (FWA)

The statement

```
  COPY DFHFWADS
```

copies the symbolic storage definition for the CICS system section of the FWA. This storage definition should precede the user's defined layout of a file record area when reading or updating an existing blocked record, when adding a new record to a file, or when retrieving records using the browse feature. If desired, the user can identify that the area returned in response to a user file request is an FWA, rather than an FIOA or VSWA, by testing label FWAIND for a ones condition using mask FWAM.

The user must code an EQU statement to set up a base register for the FWA, equating the label FWACBAR to a general-purpose register. The user must also establish addressability for an FWA acquired in response to a DFHFC macro prior to any reference to the FWA. The following is an example of the coding required; it includes the optional test (TM and BO instructions) for FWA identification:

```
  FWACBAR EQU   7
          COPY  DFHFWADS
  NAME    DS    CL20
  STREET  DS    CL30
  ZIPCODE DS    CL5
          .
          .
          .
          L     FWACBAR,TCAFCAA
          TM    FWAIND,FWAM
          BO    GOTFWA
```

VSAM WORK AREA (VSWA)

The statement

```
  COPY DFHVSWA
```

copies the symbolic storage definition for the CICS system section of the VSAM work area (VSWA) and must be present in all programs using locate mode I/O. See "Direct Retrieval (VSAM Locate Mode)" in

"Chapter 3.2. File Control (DFHFC Macro)" on page 51. If desired, the user can identify that the area returned in response to a user file request is a VSWA, rather than an FIOA or FWA, by testing label VSWAID for a zero condition using mask VSWAM.

The user must code an EQU statement to set up a base register for the VSWA, equating the label VSWABAR to a general-purpose register. After a VSWA is acquired by CICS in response to a DFHFC macro using locate mode I/O, the user must establish addressability for the VSWA prior to any reference to that area. The following is an example of the coding required; it includes the optional test (TM and BZ instructions) for VSWA identification:

```
VSWABAR EQU    7
        COPY   DFHVSWA
        .
        .
        .
        L      VSWABAR,TCAFCAA
        TM     VSWAID,VSWAM
        BZ     GOTVSWA
```

TRANSIENT DATA INPUT AREA (TDIA)

The statement

```
COPY DFHTDIA
```

copies the symbolic storage definition for the CICS system section of the intrapartition TDIA. This storage definition should precede the user's defined layout of the message area used for data obtained from an intrapartition destination by means of a DFHTD TYPE=GET macro. (See "Acquire Queued Data (TYPE=GET)" in "Chapter 5.6. Transient Data Control (DFHTD Macro)" on page 245.) The user must code an EQU statement to set up a base register for the TDIA, equating the label TDIABAR to a general-purpose register. The user must also establish addressability for the TDIA following a DFHTD macro. The following is an example of the coding required:

```
TDIABAR EQU    9
        COPY   DFHTDIA
NAME    DS     CL20
STREET  DS     CL20
        .
        .
        .
        L      TDIABAR,TCATDAA
```

TRANSIENT DATA OUTPUT AREA (TDOA)

The statement

```
COPY DFHTDOA
```

copies the symbolic storage definition for the CICS system section of the intrapartition TDOA. This storage definition should precede the user's defined layout of the message area for transient data to be directed to an intrapartition or extrapartition destination by means of a DFHTD TYPE=PUT macro. (See "Dispose of Data (TYPE=PUT)" in "Chapter 5.6. Transient Data Control (DFHTD Macro)" on page 245.)

The user must code an EQU statement to set up a base register for the TDOA, equating the label TDOABAR to a general-purpose register. The address of the data to be output (including the four-byte length field in the case of variable-length records) must be given to transient data control either through the TDADDR operand of the DFHTD macro or by placing it in TCATDAA. The following is an example of the coding required:

```
TDOABAR EQU    9
        COPY   DFHTDOA
TIME    DS     CL4
DATE    DS     PL3
INTERM  DS     CL4
OUTTERM DS     CL4
        .
        .
        .
        DFHSC  TYPE=GETMAIN
               ,CLASS=TRANSDATA
               ,NUMBYTE=19
        L      TDOABAR,TCASCSA
        .
        .
        .
        DFHTD  TYPE=PUT,DESTID=POST
               ,TDADDR=TDOAVRL
```

TDOAVRL is a name associated with the first byte of the output message (LLbb for variable length records).

TEMPORARY STORAGE INPUT/OUTPUT AREA (TSIOA)

The statement

```
COPY DFHTSIOA
```

copies the symbolic storage definition for the CICS system section of the TSIOA. This storage definition should precede the user's defined data fields. The user must code an EQU statement to set up a base register for the TSIOA, equating the label TSIOABAR to a general-purpose register. The address of the data, which always includes a length field (LLbb) for temporary storage must be given to temporary storage control either through the TSDADDR operand of the DFHTS macro or by placing it in TCATSDA. The following is an example of the coding required:

```
  TSIOABAR EQU   6
           COPY  DFHTSIOA
  PAGENO   DS    PL2
  TITLE    DS    CL30
  LINE1    DS    CL70
           .
           .
           .
           DFHTS TYPE=GET
           L     TSIOABAR,TCATSDA
           SH    TSIOABAR,=H'8'
```

Upon execution of the DFHTS TYPE=GET macro, CICS returns the address of the data portion (LLbb field) of the temporary storage record which is read in TCATSDA. To establish addressability to the TSIOA (that is, to use the DFHTSIOA DSECT), the application program must subtract eight from this address to point to the storage accounting field of the storage area acquired by CICS. If the TSDADDR operand is included in the DFHTS TYPE=GET macro, this is not required.

STORAGE ACCOUNTING AREA (SAA)

The statement

```
  COPY DFHSAADS
```

copies the symbolic storage definition for the SAA. This storage definition should precede the user's defined layout of a unique work area that is used within the application program. The user must code an EQU statement to set up a base register for the SAA, equating the label SAACBAR to a general-purpose register. The following is an example of the coding required:

```
  SAACBAR  EQU   9
           COPY  DFHSAADS
  SYMBLA   EQU   *
  NAME     DS    CL50
  STREET   DS    CL15
  SYMBLB   EQU   *-SYMBLA
           .
           .
           DFHSC TYPE=GETMAIN,INITIMG=00
           ,NUMBYTE=SYMBLB,CLASS=USER
           .
           .
           L     SAACBAR,TCASCSA
           .
           .
```

Having copied the symbolic storage definition for the SAA, the application program can specify a DFHSC TYPE=GETMAIN macro requesting CICS storage control to obtain main storage for use by the program. The address returned by CICS in TCASCSA should be moved to SAACBAR, the base address register for the SAA.

JOURNAL CONTROL AREA (JCA)

The statement

```
  COPY DFHJCADS
```

copies the symbolic storage definition for the CICS system section of the journal control area (JCA) and must be present in all programs requesting journal services. (See "Journal Control", "Chapter 7.5. Journal Control (DFHJC Macro)" on page 305.) The user must code an EQU statement to set up a base register for the JCA, equating the label JCABAR to a general-purpose register. The following is an example of the coding required:

```
  JCABAR   EQU   9
           COPY  DFHJCADS
```

A JCA is acquired by means of a DFHJC TYPE=GETJCA macro. Addressability to the JCA is automatically provided through the macro expansion, which loads the JCA address into JCABAR.

EXAMPLE OF CICS ASSEMBLER LANGUAGE APPLICATION PROGRAM

The following example is an assembler language program written to run under CICS. The program asks a question of the terminal operator, receives a reply, dynamically acquires some storage, and sends the operator's message back to the terminal. In effect, an echo test is performed. (The line numbers refer to the following notes.)

```
01  BASEREG EQU    2
02  TCTTEAR EQU    11
03  TIOABAR EQU    10
04          COPY   DFHCSADS
05          COPY   DFHTCADS
06  LENGTH  DS     H
07  MESSAGE DS     CL32
08          COPY   DFHTCTTE
09          COPY   DFHTIOA
10  MESSG   DS     CL32
11          CSECT
12          BALR   BASEREG,0
13          USING  *,BASEREG
14          L      TCTTEAR,TCAFCAAA
15          L      TIOABAR,TCTTEDA
16          MVC    MESSG,=C'ENTER
                   MSG TO BE ECHOED'
17          MVC    TIOATDL,=H'26'
18          DFHTC  TYPE=(WRITE,READ,
                   WAIT,ERASE)
19          L      TIOABAR,TCTTEDA
20          MVC    LENGTH,TIOATDL
21          MVC    MESSAGE,MESSG
22          DFHSC  TYPE=GETMAIN,
23                 CLASS=TERMINAL,
24                 NUMBYTE=32
25          L      TIOABAR,TCASCSA
26          ST     TIOABAR,TCTTEDA
27          MVC    MESSG,MESSAGE
28          MVC    TIOATDL,LENGTH
29          DFHTC  TYPE=WRITE
30          DFHPC  TYPE=RETURN
31          END
```

Line	**Description**
01	Assigns base register for program.
02-03	Assigns base register for TCTTE and TIOA symbolic storage definitions.
04-05	Copies CSA and TCA symbolic storage definitions.
06-07	Defines fields in TWA as save areas to provide for quasi-reenterability.
08-09	Copies TCTTE and TIOA symbolic storage definitions.
10	Defines message area in TIOA.
11-13	Begins program; establishes addressability for program.
14	Establishes addressability for TCTTE
15	Establishes addressability for TIOA.
16	Moves message to output area of TIOA.
17	Moves length of message to data length field of TIOA.
18	CICS macro that writes message to terminal, waits for operator's reply, and reads operator's reply.
19	Establishes addressability for new TIOA, using address in TCTTE.
20-21	Saves the message and the length of the message in the TWA save area.
22-24	CICS macro that requests 32 bytes of terminal type storage.
25	Establishes addressability for new TIOA (address of newly acquired storage area is in TCASCSA field of the TCA).
26	Places address of new TIOA in TCTTE.
27	Moves the message from TWA save area to new TIOA.
28	Moves the message length to data length field of new TIOA.
29	CICS macro that writes message to terminal.
30	CICS macro that returns control to CICS and terminates this task.
31	Required for assembler language.

CHAPTER 2.3. STORAGE DEFINITION - COBOL

The COBOL programmer must define storage for the CICS control areas and any other storage areas required for the processing of the application program. This is done by using (1) the COPY statement in the linkage section of the data division to copy the symbolic storage definitions into the program and specify the names of the storage areas being defined, and (2) the MOVE statement in the procedure division to establish addressability by moving symbolic storage addresses from one location to another.

The working storage section of a COBOL program should contain only data constants. Variable data should be placed in a TWA or in an area of storage acquired by a DFHSC TYPE=GETMAIN macro. (See "Obtain and Initialize Main Storage (TYPE=GETMAIN)" in "Chapter 5.5. Storage Control (DFHSC Macro)" on page 241.) Note that all COBOL examples in this manual are written to language level 68 (LANGLVL(1)).

The statement

```
01 DFHBLLDS COPY DFHBLLDS.
```

must be the first statement in the linkage section of the data division of a COBOL program that is run under CICS. This statement copies the symbolic storage definition for the linkage section base locator (BLL), which provides the means by which a COBOL program can address dynamically acquired CICS storage areas. Included in this definition are the symbolic base addresses for the common system area (CSA), common system area optional features list (CSAOPFL), and task control area (TCA). Symbolic storage definitions for these areas must be copied into every COBOL program.

If other CICS storage areas are needed, the COPY statement for the BLL must be followed immediately by statements of the form:

```
02 name PIC S9(8) COMP.
```

where "name" is the symbolic base address used to locate a specific storage area. There must be one of these statements for each additional type of storage needed by the application program. Furthermore, these 02-level statements must be coded in the same order as the corresponding 01-level COPY statements coded subsequently to copy the symbolic storage definitions for the areas into the application program.

If the user is going to communicate with the system by means of a terminal, a terminal input/output area (TIOA) and a terminal control table terminal entry (TCTTE) are needed. Assuming that only the required control areas (CSA and TCA), a TIOA, and a TCTTE are needed for an application, the following example shows the coding required in the linkage section of the data division:

```
01 DFHBLLDS COPY DFHBLLDS.
   02 TCTTEAR PIC S9(8) COMP.
   02 TIOABAR PIC S9(8) COMP.
01 DFHCSADS COPY DFHCSADS.
01 DFHTCADS COPY DFHTCADS.
01 DFHTCTTE COPY DFHTCTTE.
01 DFHTIOA COPY DFHTIOA.
```

STORAGE DEFINED DURING INITIALIZATION

During CICS initialization, the CSA is allocated as part of the CICS nucleus. For each terminal that is to be used, TCTTE must be included in the TCT. The COBOL programmer must provide symbolic storage definitions for the CSA and TCTTE (if needed) as follows.

COMMON SYSTEM AREA (CSA)

The statement

```
01 DFHCSADS COPY DFHCSADS.
```

copies the symbolic storage definition for the CSA. Addressability for the CSA is included.

If CICS is generated to include a CWA, a symbolic definition of that area must be included immediately following the COPY statement in the linkage section of the application program. The following is an example of the coding required:

```
01 DFHCSADS COPY DFHCSADS.
   02 CWA.
      03 FIELD1 PIC X(4).
      .
      .
      .
```

TERMINAL CONTROL TABLE TERMINAL ENTRY (TCTTE)

The statement

```
01 DFHTCTTE COPY DFHTCTTE.
```

copies the symbolic storage definition for the TCTTE and must be present in all programs requesting communication with a terminal. The user must code the statement

```
  MOVE TCAFCAAA TO TCTTEAR.
```

in the appropriate place in the procedure division to establish addressability for the TCTTE. TCAFCAAA contains the address of the facility that initiated the transaction. TCTTEAR is the terminal control table terminal entry address register.

STORAGE DEFINED DURING EXECUTION

During the execution of a task, the TCA, the TIOA , and other storage areas required by the task are allocated by CICS storage management upon request from either the application program or CICS. Symbolic storage definitions for these storage areas must be provided as follows.

TASK CONTROL AREA (TCA)

The statement

```
  01 DFHTCADS COPY DFHTCADS.
```

copies the symbolic storage definitions for the CSA optional features list and the TCA. The user must code the statement

```
  MOVE CSACDTA TO TCACBAR.
```

and can optionally code the statement

```
  MOVE CSAOPFLA TO CSAOPBAR.
```

at the appropriate place in the procedure division to establish addressability for the TCA and the CSA optional features list. CSACDTA contains the address of the storage area obtained for the TCA (the common system area currently dispatched task address). This address is stored in TCACBAR, the TCA control base address register.

If the application program requires the use of a TWA, the record layout of the TWA must be defined immediately following the COPY statement in the linkage section of the application program. The following is an example of the coding required:

```
  01 DFHTCADS COPY DFHTCADS.
     02 TWA PIC X(40).
         .
         .
         .
```

TERMINAL INPUT/OUTPUT AREA (TIOA)

The statement

```
  01 DFHTIOA COPY DFHTIOA.
```

copies the symbolic storage definition for the CICS system section of the TIOA and must be present in all programs that use terminal input records or that provide output records to a terminal. The following is an example of the coding required to define the record(s) in the TIOA:

```
  01 DFHTIOA COPY DFHTIOA.
     02 TRANSID PIC XXXX.
     02 TIOAMSG PIC X(20).
         .
         .
         .
```

The user must establish addressability for the TIOA in the procedure division by coding in the appropriate place either the statement

```
  MOVE TCTTEDA TO TIOABAR.
```

or the statement

```
  MOVE TCASCSA TO TIOABAR.
```

The former statement is used to establish addressability to a TIOA acquired by CICS during execution for data entered from a terminal. The latter statement is used to establish addressability for a new TIOA acquired by a DFHSC TYPE=GETMAIN macro and should be coded immediately following that macro.

FILE INPUT/OUTPUT AREA (FIOA)

The statement

```
  01 DFHFIOA COPY DFHFIOA.
```

copies the symbolic storage definition for the CICS system section of the FIOA and must be present in all programs requesting a read of an unblocked record without updating, or a read of blocked records without deblocking. If desired, the user can identify that the area returned in response to a file request is an FIOA, rather than an FWA or VSWA, by testing FIOAM. For CICS/OS/VS only, | if data is retrieved using an existing | ISAM application in ISAM compatibility | mode, the FIOA must include a 16-byte filler prior to the user's data definition. The following is an example of the coding required to define records in the FIOA:

```
  01 DFHFIOA COPY DFHFIOA.
     02 KEYF PIC X(6).
     02 NAME PIC X(20).
     02 FIOAREC PIC X(74).
         .
         .
         .
```

The user must code the statement

```
  MOVE TCAFCAA TO FIOABAR.
```

prior to any reference to the FIOA following a DFHFC macro in the procedure division to establish addressability for the FIOA.

To identify the area returned as an FIOA, the following instruction can be used:

```
  IF FIOAM
  THEN GO TO GOTFIOA.
```

FILE WORK AREA (FWA)

The statement

```
  01 DFHFWADS COPY DFHFWADS.
```

copies the symbolic storage definition for the CICS system section of the FWA and must be present in all programs performing file operations with the exception of a "read without update" from an unblocked data set. If desired, the user can identify the area returned in response to a file request as an FWA, rather than an FIOA or VSWA, by testing FWAM. The following is an example of the coding required to define records in the FWA:

```
  01 DFHFWADS COPY DFHFWADS.
     02 KEYF PIC X(6).
     02 NAME PIC X(20).
     02 FWAREC PIC X(24).
     .
     .
     .
```

The user must code the statement

```
  MOVE TCAFCAA TO FWACBAR.
```

prior to any reference to the FWA following a DFHFC macro in the procedure division to establish addressability for the FWA.

To identify the area returned as an FWA, the following instruction can be used:

```
  IF FWAM
  THEN GO TO GOTFWA.
```

VSAM WORK AREA (VSWA)

The statement

```
  01 DFHVSWA COPY DFHVSWA.
```

copies the symbolic storage definition for the CICS system section of the VSAM work area and must be present in all programs using VSAM locate mode I/O. See "Direct Retrieval (VSAM Locate Mode)' in "Chapter 3.2. File Control (DFHFC Macro)" on page 51. If desired, the user can identify that the area returned in response to a file request is a VSWA, rather than an FIOA or FWA, by testing VSWAM. The user must code the statement.

```
  MOVE TCAFCAA TO VSWABAR.
```

prior to any reference to the VSWA acquired by CICS in response to a DFHFC macro using locate mode I/O.

To identify the area returned as a VSWA, the following instruction can be used:

```
  IF VSWAM
  THEN GO TO GOTVSWA.
```

TRANSIENT DATA INPUT AREA (TDIA)

The statement

```
  01 DFHTDIA COPY DFHTDIA.
```

copies the symbolic storage definition for the CICS system section of the intrapartition TDIA and must be present in all programs requiring a message area for transient data obtained by issuing a DFHTD TYPE=GET macro that refers to an intrapartition destination. (See "Acquire Queued Data (TYPE=GET)" in in "Chapter 5.6. Transient Data Control (DFHTD Macro)" on page 245.) The following is an example of the coding required to define records in the TDIA:

```
  01 DFHTDIA COPY DFHTDIA.
     02 MESSAGE PIC X(25).
```

The user must code the statement

```
  MOVE TCATDAA TO TDIABAR.
```

prior to any reference to the TDIA following a DFHTD macro in the Procedure Division to establish addressability for the TDIA.

TRANSIENT DATA OUTPUT AREA (TDOA)

The statement

```
  01 DFHTDOA COPY DFHTDOA.
```

copies the symbolic storage definition for the CICS system section of the intrapartition TDOA and should be present in all programs issuing a DFHTD TYPE=PUT macro to provide transient data as output. (See "Dispose of Data (TYPE=PUT)" in "Chapter 5.6. Transient Data Control (DFHTD Macro)" on page 245.) The following is an example of the coding required to define records in the TDOA:

```
  01 DFHTDOA COPY DFHTDOA.
     02 MESSAGE PIC X(20).
```

The user must code the statement

```
  MOVE TCASCSA TO TDOABAR.
```

prior to any reference to the TDOA following a DFHSC macro in the procedure division to establish addressability for the TDOA.

TEMPORARY STORAGE INPUT/OUTPUT AREA (TSIOA)

The statement

```
  01 DFHTSIOA COPY DFHTSIOA.
```

copies the symbolic storage definition for the CICS system section of the TSIOA and should be present in all programs using temporary storage. The following is an example of the coding required to define records in the TSIOA:

```
  01 DFHTSIOA COPY DFHTSIOA.
      02 DATA PIC X(10).
```

To establish addressability for the TSIOA, the user must code the statements

```
  MOVE TCATSDA TO TSIOABAR.
  SUBTRACT 8 FROM TSIOABAR.
```

if the request is a GET or GETQ from temporary storage and the TSDADDR operand is not specified. The subtraction of 8 bytes ensures that TSIOABAR points to the storage accounting field (that is, to the beginning) of the storage area acquired by CICS. The user must code the statement

```
  MOVE TCASCSA TO TSIOABAR.
```

if an I/O area has been acquired during execution. In the case of a PUT or PUTQ, the symbolic address of the data is located at TSIOAVRL. Either statement must appear in the appropriate place in the Procedure Division of the COBOL program.

STORAGE ACCOUNTING AREA (SAA)

The statement

```
  01 DFHSAADS COPY DFHSAADS.
```

copies the symbolic storage definition for the SAA. This storage definition should precede the definition of user storage acquired through the DFHSC TYPE=GETMAIN,CLASS=USER macro. The following is an example of the coding required to define records in the SAA:

```
  01 DFHSAADS COPY DFHSAADS.
      02 NAME PIC X(20).
      02 SAAREC PIC X(10).
      .
      .
      .
```

The user must code the statement

```
  MOVE TCASCSA TO SAACBAR.
```

prior to any reference to the SAA following a DFHSC macro in the Procedure Division to establish addressability for the SAA.

JOURNAL CONTROL AREA (JCA)

The statement

```
  01 DFHJCADS COPY DFHJCADS.
```

copies the symbolic storage definition for the CICS system section of the journal control area (JCA) and must be present in all programs requesting journal services. (See "Journal Control", "Chapter 7.5. Journal Control (DFHJC Macro)" on page 305.)

A JCA is acquired by means of a DFHJC TYPE=GETJCA macro. Addressability to the JCA is provided automatically through the macro expansion, which loads the address of the area into JCABAR.

ADDITIONAL GUIDELINES

If the object of an OCCURS DEPENDING ON clause is defined in the linkage section, special consideration 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 referencing DATA.

```
  LINKAGE SECTION.
  01 DFHFWADS COPY DFSFWADS.
      .
      .
      .
      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.
      .
      .
      .
      DFHFC TYPE=GET, etc.
      MOVE TCAFCAA TO FWACBAR.
      MOVE FIELD-COUNTER
      TO FIELD-COUNTER.
      MOVE DATA TO TWA-FIELD.
```

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

If an area greater than 4096 bytes is defined in the linkage section, special considerations arise. An additional 02-level statement under DFHBLLDS and an ADD statement following the MOVE statement to establish addressability to the area are required for each additional 4096 bytes. For example, if a file work area (FWA) exceeds 4096 bytes, the following code can be used.

```
LINKAGE SECTION.
01 DFHBLLDS COPY DFHBLLDS.
   .
   .
   .
   02 FWACBAR PIC S9(8) COMP
   02 FWABR1 PIC S9(8) COMP
   .
   .
   .
01 DFHFWADS COPY DFHFWADS.
   02 FIELD1 PIC X(4000).
   02 FIELD2 PIC X(1000).
   02 FIELD3 PIC X(400).
   .
   .
   .
PROCEDURE DIVISION.
   .
   .
   .
   DFHFC TYPE=GET,
   .
   .
   .
   MOVE TCAFCAA TO FWACBAR.
   ADD 4096 TO FWACBAR GIVING FWABR1.
```

If the size of the COBOL working storage is close to, or greater than 64K, execution errors may occur.

If an application program is to be compiled for execution under CICS/OS/VS using the full COBOL V4 Compiler (5734-CB2), the OS/VS COBOL Compiler (5740-CB1) with the optimization feature, or the DOS/VS COBOL Compiler (5746-CB1) with the optimization feature, a special translator 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, and is also defined in an 01-level statement in the linkage section. The first four statements of the Procedure Division must be:

```
SERVICE RELOAD DFHBLLDS.
SERVICE RELOAD DFHCSADS.
MOVE CSAOPFLA TO CSAOPBAR.
SERVICE RELOAD CSAOPFL.
```

Statements such as:

```
MOVE TCAFCAAA TO TCTTEAR.
SERVICE RELOAD DFHTCTTE.
```

or

```
SUBTRACT 8 FROM TCASCSA
GIVING TSIOABAR.
SERVICE RELOAD DFHTSIOA.
```

can be used to establish addressability for a particular storage area. (Note that the SERVICE RELOAD statement must be used following each statement which modifies addressability to an area defined in the linkage section, that is, whenever an address is moved to a field named in an 02-level statement under 01 DFHBLLDS or the address in the 02-level statement is changed in any way.)

To establish addressability to the TCA, the following statements must be coded:

```
MOVE CSACDTA TO TCACBAR.
SERVICE RELOAD DFHTCA.
```

Note that the SERVICE RELOAD statement specifies DFHTCA, not DFHTCADS.

If areas larger than 4096 bytes are addressed, a SERVICE RELOAD statement must be issued by the user after the primary BLL is updated, but before the secondary BLLs are updated.

Certain COBOL features cannot be used in an application program to be run under CICS. Generally, these features are replaced by CICS services. They are identified under "Restrictions" on page 15.

<u>EXAMPLE OF CICS COBOL APPLICATION PROGRAM</u>

The following example is a COBOL program written to run under CICS. The program asks a question of the terminal operator, receives a reply, acquires storage, and sends the operator's message back to the terminal. In effect, an echo test is performed. (The line numbers refer to the following notes.)

```
01  IDENTIFICATION DIVISION.
02  PROGRAM-ID.
03      'CBLSPRB'.
04  ENVIRONMENT DIVISION.
05  DATA DIVISION.
06  LINKAGE SECTION.
07  01 DFHBLLDS COPY DFHBLLDS.
08      02 TCTTEAR PIC S9(8) COMP.
09      02 TIOABAR PIC S9(8) COMP.
10  01 DFHCSADS COPY DFHCSADS.
11  01 DFHTCADS COPY DFHTCADS.
12      02 SAVE-LENGTH PIC S9(8) COMP.
13      02 SAVE-MESSAGE PIC X(36).
14  01 DFHTCTTE COPY DFHTCTTE.
15  01 DFHTIOA COPY DFHTIOA.
16      02 TIOAMSG PIC X(36).
17  PROCEDURE DIVISION.
18      MOVE CSACDTA TO TCACBAR.
19      MOVE CSAOPFLA TO CSAOPBAR.
20      MOVE TCAFCAAA TO TCTTEAR.
21      MOVE TCTTEDA TO TIOABAR.
22      MOVE 'ENTER MESSAGE TO BE
        ECHOED' TO TIOAMSG.
23      MOVE 26 TO TIOATDL.
24  DFHTC TYPE=(WRITE,READ,WAIT)
25      MOVE TCTTEDA TO TIOABAR.
26      MOVE TIOATDL TO SAVE-LENGTH.
27      MOVE TIOAMSG TO SAVE-MESSAGE.
28  DFHSC TYPE=GETMAIN,
29          NUMBYTE=36,
30          CLASS=TERMINAL
31      MOVE TCASCSA TO TIOABAR.
32      MOVE TIOABAR TO TCTTEDA.
33      MOVE SAVE-MESSAGE TO TIOAMSG.
34      MOVE SAVE-LENGTH TO TIOATDL.
35  DFHTC TYPE=WRITE
36  DFHPC TYPE=RETURN
37      GOBACK.
```

Line	**Description**
01-05	Required for COBOL.
06	Start of linkage section.
07	Copies symbolic storage definition for BLL; contains addresses of CICS storage areas.
08-09	Adds addresses for TCTTE and TIOA (required for statements 14 and 15).
10	Copies symbolic storage definition for CSA.
11	Copies symbolic storage definitions for TCA and CSA optional features list.
12-13	Defines save areas in TWA to ensure quasi-reenterability (SAVE-LENGTH and SAVE-MESSAGE are used to save operator's reply).
14	Copies symbolic storage definition for TCTTE.
15	Copies symbolic storage definition for TIOA.
16	Defines message area in TIOA.
17	Required for COBOL (start of Procedure Division).
18-21	Establishes addressability for TCA, CSA optional features list, TCTTE, and TIOA (CICS establishes addressability for BLL and CSA).
22	Moves message to output area of TIOA.
23	Moves length of message to data length field of TIOA.
24	CICS macro that writes message to terminal, waits for operator's reply, and reads operator's reply.
25	Establishes addressability for new TIOA using address in TCTTE.
26	Saves length of message in TWA.
27	Saves message in TWA.
28-30	CICS macro that requests 36 bytes of terminal storage (terminal storage is chained to terminal control table).
31	Establishes addressability for new TIOA (address of newly acquired storage area is in TCASCSA field of the TCA).
32	Places address of new TIOA in terminal control table.
33	Moves message to output area (TIOA).
34	Moves length of message to output area (TIOA).
35	CICS macro that writes message to terminal.
36	CICS macro that returns control to CICS.
37	COBOL statement that marks the end of the program.

CHAPTER 2.4. STORAGE DEFINITION - PL/I

The PL/I programmer must define storage for the CICS control areas and other storage areas required for the processing of the application program. This is done by using a statement of the form:

```
  %INCLUDE library(member);
      or
  %INCLUDE member;
```

to (1) copy the appropriate symbolic storage definition into the application program at the place where the %INCLUDE statement appears, and (2) specify the name of the storage area being defined.

The PL/I source code provided by CICS in response to %INCLUDE statements is in the form of based structures. These structures describe the attributes of the storage areas and include pointer variables that provide the addresses of the actual locations in storage that the structures describe.

All application programs must contain statements to copy the symbolic storage definitions for the common system area (CSA) and task control area (TCA). The expansions of the CICS macros used in an application program refer to fields within these areas, so their locations must be identified. Whether additional storage definitions must be copied depends on the processing requirements (storage areas and macros used) of the application program. The statements to copy the symbolic storage definitions must be in the order CSA, TCA, TCTTE, TIOA; this is because addressability for the last three areas mentioned depends on the previous area already having been copied.

A PL/I program to be run under CICS must contain the REENTRANT option in the first PROCEDURE statement to satisfy the CICS requirement that code be quasi-reenterable. See "Programming Techniques and Restrictions" in Part 1 for a list of PL/I features that cannot be used.

STORAGE DEFINED DURING INITIALIZATION

During CICS initialization, the CSA is allocated as part of the CICS nucleus. For each terminal that is to be used, a terminal control table terminal entry (TCTTE) must be included in the terminal control table (TCT). The PL/I programmer must provide symbolic storage definitions for the CSA and TCTTE (if needed) as follows.

COMMON SYSTEM AREA (CSA)

The statement

```
  %INCLUDE DFHCSADS;
```

copies the based structures that symbolically define the CSA and the CSA optional features list. Addressability for both areas is included.

If CICS is generated to support a common work area (CWA), coding such as the following must be provided immediately following the %INCLUDE DFHCSADS macro:

```
  DCL 1 DFHCSAWK BASED(CSACBAR),
      2 CSAFILL CHAR(512),
      2 USERLBL1 attributes,
      .
      .
      .
      2 USERLBLn attributes;
```

TERMINAL CONTROL TABLE TERMINAL ENTRY (TCTTE)

The statement

```
  %INCLUDE DFHTCTTE;
```

copies the based structure that symbolically defines the TCTTE and must be present in all programs requesting communication with a terminal. Addressability for the TCTTE is included.

STORAGE DEFINED DURING EXECUTION

During execution of a task, the task control area (TCA), terminal input/output area (TIOA), and other storage areas required by the task are allocated by CICS storage management upon request from either the application program or CICS. Symbolic definitions for these areas must be provided, as follows.

TASK CONTROL AREA (TCA)

The statement

```
  %INCLUDE DFHTCADS;
```

copies the based structure that defines the TCA and establishes addressability.

The latter part of the based structure consists of a DECLARE statement that is not terminated by a semicolon. The declaration of the TCA structure must be completed by supplying an ending (for example, a semicolon) or, if a transaction work area (TWA) is desired, by supplying further declaration. The following is an example of the coding required:

```
  %INCLUDE DFHTCADS;
     2 TWA CHAR(40);
     .
     .
     .
```

TERMINAL INPUT/OUTPUT AREA (TIOA)

The statement

```
  %INCLUDE DFHTIOA;
```

copies the based structure that defines the CICS system section of the TIOA and establishes addressability. This statement must be present in all programs that use terminal input records or that write output records to a terminal. The declaration of the TIOA structure must be completed by supplying further declaration of the input/output area, which could be merely a dummy element. An action that requires a TIOA can be requested. For example, a DFHSC TYPE=GETMAIN macro to obtain storage for a TIOA for the application program. The following is an example of the coding required:

```
  %INCLUDE DFHTIOA;
     2 NAME CHAR(20),
     2 STREET CHAR(20);
     .
     .
     .
  DFHSC TYPE=GETMAIN,
        NUMBYTE=40,
        CLASS=TERMINAL
  TIOABAR=TCASCSA;
  /* TCASCSA FIELD OF TCA CONTAINS
  ADDRESS OF NEWLY ACQUIRED STORAGE */
     .
     .
     .
```

For additional information about GETMAIN, see "Obtain and Initialize Main Storage (TYPE=GETMAIN)" in "Chapter 5.5. Storage Control (DFHSC Macro)" on page 241.

FILE INPUT/OUTPUT AREA (FIOA)

The statement

```
  %INCLUDE DFHFIOA;
```

copies the based structure that defines the CICS system section of the FIOA and must be present in all programs requesting a read of an unblocked record without updating, or a read of blocked records without deblocking. If desired, the user can identify that the area returned in response to a file request is an FIOA, rather than an FWA or VSWA, by testing FIOAIND for a bit value of 01. The declaration of the FIOA must be completed, and addressability must be established for the FIOA using the statement

```
  FIOABAR=TCAFCAA;
```

following the DFHFC macro. For CICS/OS/VS only, if data is retrieved using an existing ISAM application in ISAM compatibility mode, the FIOA must include a 16-byte filler prior to the user's data definition. The following is an example of the coding required; it includes the optional coding for FIOA identification:

```
  %INCLUDE DFHFIOA;
     2 NAME CHAR(20),
     2 ADDR CHAR(20);
     .
     .
     .
  FIOABAR=TCAFCAA;
  IF FIOAIND='01'B THEN GO TO GOTFIOA;
     .
     .
     .
```

FILE WORK AREA (FWA)

The statement

```
  %INCLUDE DFHFWADS;
```

copies the based structure that defines the CICS system section of the FWA. This statement should precede a user-declared file record area when reading or updating an existing blocked record, when adding a new record to a data set, or when retrieving records using the browse technique. If desired, the user can identify that the area returned in response to a file request is an FWA, rather than an FIOA or VSWA, by testing FWAIND for a bit value of 11. The declaration of the FWA must be completed, and addressability must be established for the FWA using the statement

```
  FWACBAR=TCAFCAA;
```

following a DFHFC macro. The following is an example of the coding required; it includes the optional test for FWA identification:

```
  %INCLUDE DFHFWADS;
     2 NAME CHAR(20),
     2 ADDR CHAR (20);
     .
     .
     .
  FWACBAR=TCAFCAA;
  IF FWAIND='11'B THEN GO TO GOTFWA;
     .
     .
     .
```

VSAM WORK AREA (VSWA)

The statement

```
  %INCLUDE DFHVSWA;
```

copies the based structure that defines the CICS system section of the VSAM work area and must be present in all programs using locate mode I/O. See "Direct Retrieval (VSAM Locate Mode)" in "Chapter 3.2. File Control (DFHFC Macro)" on page 51. If desired, the user can identify that the area returned in response to a file request is a VSWA, rather than an FIOA or FWA, by testing VSWAID for a bit value of 00000000. Addressability must be established for the VSWA using the statement

```
  VSWABAR=TCAFCAA;
```

following the DFHFC macro using locate mode I/O which causes CICS to acquire the VSWA.

To identify the area returned as a VSWA, the following instruction can be used:

```
  IF VSWAID='0'B THEN GO TO GOTVSWA;
```

TRANSIENT DATA INPUT AREA (TDIA)

The statement

```
  %INCLUDE DFHTDIA;
```

copies the based structure that defines the CICS system section of the intrapartition TDIA and must be present in all programs requiring a message area for transient data obtained by issuing a DFHTD TYPE=GET macro that references an intrapartition destination. (See "Acquire Queued Data (TYPE=GET)" in "Chapter 5.6. Transient Data Control (DFHTD Macro)" on page 245.) The declaration of the TDIA must be completed, and addressability must be established for the TDIA using the statement

```
  TDIABAR=TCATDAA;
```

following a DFHTD macro. The following is an example of the coding required:

```
  %INCLUDE (DFHTDIA);
     2 MSG CHAR(40);
     .
     .
     .
  TDIABAR=TCATDAA;
     .
     .
     .
```

TRANSIENT DATA OUTPUT AREA (TDOA)

The statement

```
  %INCLUDE DFHTDOA;
```

copies the based structure that defines the CICS system section of the intrapartition TDOA and should be present in all programs issuing a DFHTC TYPE=PUT macro to provide transient data as output. (See "Dispose of Data (TYPE=PUT)" in "Chapter 5.6. Transient Data Control (DFHTD Macro)" on page 245.) The declaration of the TDOA must be completed, and addressability must be established for the TDOA using the statement

```
  TDOABAR=TCASCSA;
```

following a DFHSC macro. The following is an example of the coding required:

```
  %INCLUDE DFHTDOA;
     2 TIME CHAR(2),
     2 DATA CHAR(3),
     2 INTERM CHAR(4),
     2 OUTTERM CHAR(4);
     .
     .
     .
  DFHSC TYPE=GETMAIN,
        NUMBYTE=XX,
        CLASS=USER
  TDOABAR=TCASCSA;
     .
     .
     .
```

TEMPORARY STORAGE INPUT/OUTPUT AREA (TSIOA)

The statement

```
  %INCLUDE DFHTSIOA;
```

copies the based structure that defines the CICS system section of the TSIOA and must be present in all programs using temporary storage. The declaration for the TSIOA must be completed. If the request is a GET or GETQ from temporary storage and the TSDADDR operand is not specified, addressability must be established for the TSIOA using coding such as:

```
  DCL TSIOABAA FIXED BIN(31)
      BASED(TSIOABAB);
      TSIOABAR=TCATSDA;
      TSIOABAB=ADDR(TSIOABAR);
      TSIOABAA=TSIOABAA-8;
```

The subtraction of eight ensures that TSIOABAA points to the storage accounting field (that is, to the beginning) of the storage area acquired by CICS. The statement

```
  TSIOABAR=TCASCSA;
```

must be coded if the I/O area has been acquired during execution. In the case of a PUT or PUTQ, the symbolic address of the data is located at TSIOAVRL.

STORAGE ACCOUNTING AREA (SAA)

The statement

```
  %INCLUDE DFHSAADS;
```

copies the based structure that defines the SAA and must be present in all programs requesting storage through use of the DFHSC TYPE=GETMAIN,CLASS=USER macro. This statement must precede the definition of user storage. The declaration for the SAA must be completed, and addressability must be established for the SAA using the statement

```
  SAACBAR=TCASCSA;
```

The following is an example of the coding required:

```
  %INCLUDE DFHSAADS;
     2 MSG CHAR(40);
     .
     .
     .
  DFHSC TYPE=GETMAIN,
        NUMBYTE=60,
        CLASS=USER
  SAACBAR=TCASCSA;
     .
     .
```

JOURNAL CONTROL AREA (JCA)

The statement

```
  %INCLUDE DFHJCADS;
```

copies the based structure that defines the CICS system section of the journal control area (JCA) and must be present in all programs requesting journal services. (See "Chapter 7.5. Journal Control (DFHJC Macro)" on page 305.)

A JCA is acquired dynamically by means of a DFHJC TYPE=GETJCA macro. Addressability to the JCA is provided automatically through the macro expansion, which loads the address of the area into JCABAR.

EXAMPLE OF CICS PL/I APPLICATION PROGRAM

The following example is a PL/I program written to run under CICS. The program asks a question of the terminal operator, receives a reply, acquires storage, and sends the operator's message back to the terminal. In effect, an echo test is performed. (The line numbers refer to the notes that follow the coding.)

```
01  PL1PROG: PROC OPTIONS(MAIN,
             REENTRANT);
02  %INCLUDE DFHCSADS;
03  %INCLUDE DFHTCADS;
04     2 SAVE_LENGTH BIN FIXED(15),
05     2 SAVE_MSG CHAR(36);
06  %INCLUDE (DFHTCTTE);
07  %INCLUDE (DFHTIOA);
08     2 TIOAMSG CHAR(36);
09  TIOAMSG='ENTER MSG TO BE ECHOED';
10  TIOATDL=26;
11  DFHTC TYPE=(WRITE,READ,WAIT)
12  TIOABAR=TCTTEDA;
13  SAVE_LENGTH=TIOATDL;
14  SAVE_MSG=TIOAMSG;
15  DFHSC TYPE=GETMAIN,
16        NUMBYTE=36,
17        CLASS=TERMINAL
18  TIOABAR=TCASCSA;
19  TCTTEDA=TIOABAR;
20  TIOAMSG=SAVE_MSG;
21  TIOATDL=SAVE_LENGTH;
22  DFHTC TYPE=WRITE
23  END;
```

Line	**Description**
01	Required for PL/I. REENTRANT option specified to meet requirement of CICS that code be quasi-reenterable.
02	Copies symbolic storage definitions for CSA and CSA optional features list and establishes addressability.
03	Copies symbolic storage definition for TCA and establishes addressability.
04-05	Defines the TWA and terminates the DECLARE statement. SAVE_MSG and SAVE_LENGTH are used to preserve the operator's reply
06	Copies symbolic storage definition for TCTTE and TCTTE and establishes addressability.
07	Copies symbolic storage definition for TIOA and establishes addressability.
08	Describes I/O area for terminal message and terminates the DECLARE statement.
09	Places message to be sent to operator in the TIOA.
10	Places the message length in the terminal data length field of the TIOA.
11	CICS macro that writes message to the terminal, waits for, and reads, the operator's reply.
12	Reestablishes addressability for the TIOA using address in TCTTE.
13-14	Saves the operator's message and its length in the TCA.
15-17	CICS macro requesting 36 bytes of terminal storage (terminal storage is chained to the TCT).

18	Establishes addressability for the new TIOA (address of the newly acquired storage is in TCASCSA).
19	Places address of new TIOA in TCT.
20-21	Moves message and length of message to output area (TIOA).
22	CICS macro that sends operator's message back to the terminal.
23	PL/I statement that marks the end of the procedure.

PART 3. FILES AND DATA BASES

CHAPTER 3.1. INTRODUCTION TO FILES AND DATA BASES

The other two chapters in this part describe the methods of handling records: directly by the file control macro; and indirectly by the DL/I interface.

FILE CONTROL MACRO

"Chapter 3.2. File Control (DFHFC Macro)" on page 51 describes how a CICS application program handles records by means of the file control program. Records are operated on by the file control macro (DFHFC), according to the various TYPE operands; for example, records can be retrieved by the DFHFC TYPE=GET macro.

The file control program can be used only with direct-access data sets. Sequential data sets are handled by the transient data program and the DFHTD macro, as described in "Chapter 5.6. Transient Data Control (DFHTD Macro)" on page 245.

An application program can also browse a data set by means of the file control macro. Browsing is defined as the retrieval of records in a direct-access data set, starting and ending at specified records, in ascending or descending sequence.

DL/I SERVICES

"Chapter 3.3. DL/I Services" on page 87 describes the macros and calls available to a CICS application program that enable that program to use a DL/I data base.

The method of invoking DL/I differs for the two operating systems used with CICS. For CICS/OS/VS, the DL/I interface is invoked by either a DL/I CALL statement or by a DFHFC macro. For CICS/DOS/VS, the DL/I interface is invoked only by a DL/I CALL statement.

DL/I is a general-purpose data base control system that executes in a virtual storage environment. When used online, it simplifies the task of creating and maintaining large data bases that are to be accessed by various application programs. For more information about DL/I, refer to the DL/I publications listed in the bibliography and to the appropriate CICS Facilities and Planning Guide.

CHAPTER 3.2. FILE CONTROL (DFHFC MACRO)

The CICS file control program processes fixed- or variable-length, blocked or unblocked, or undefined records of a data set that is stored in a direct-access storage device.

File control uses the standard access methods of the host operating system, namely:

- VSAM (Virtual Storage Access Method)
- DAM (Direct Access Method).

Application programs can access DAM data sets on a logical record level, deblocking services being provided by file control. Data sets on fixed block architecture (FBA) devices can be accessed by VSAM only.

Through the file control macro (DFHFC), an application program can perform file inquiry, that is, read a record from a data set; browse through records in the data set in sequence; update a record in a data set; or add a record to a data set. In the last case the application program must obtain sufficient main storage for the record by means of the DFHFC TYPE=GETAREA macro. The application program can also release the main storage that has been acquired.

For VSAM key sequenced (KSDS) or relative record (RRDS) data sets only, the DFHFC macro can be used to delete records, singly or in groups.

All buffers and work areas needed for data set operations are acquired by file control in accordance with the data set definitions supplied in the file control table (the FCT) by the system programmer. All data sets referred to in DFHFC macros must have been defined in the FCT. The application programmer should work with the system programmer in setting up these data set definitions. However, the application program need deal only with logical records; it is not directly involved with other characteristics of the data set.

For a VSAM data set, all data is read into or written from one of three areas in main storage:

- A file work area (FWA)
- A VSAM work area (VSWA)
- A file input/output area (FIOA).

In general, most data is read into or written from an FWA. There are two exceptions, a locate-mode read-only request, and when operating in ISAM compatibility mode. (ISAM compatibility mode is indicated by the system programmer specifying UNBLOCKED in the RECFORM operand of the FCT entry for the data set.)

For locate-mode read only, the address of the retrieved record, as it is positioned in the VSAM buffer, is made available to the application program in a VSWA (in field VSWAREA). The record (that must not be modified) remains in this buffer.

In ISAM compatibility mode, the retrieved record is moved to an FIOA for a read only request for an unblocked record. A symbolic storage definition must be provided for this area (for example, an assembler language DSECT) and addressability must be established to it. For CICS/OS/VS only, a 16-byte filler must be defined before the user data.

If an error occurs while a VSAM data set is being accessed, a DFHFC TYPE=RELEASE macro must be issued after the error has occurred, otherwise a permanent wait may result.

The user can determine which area (FWA, VSWA, or FIOA) is returned in response to a file request. Refer to "Chapter 2.2. Storage Definition - Assembler Language" on page 29, "Chapter 2.3. Storage Definition - COBOL" on page 35 or "Chapter 2.4. Storage Definition - PL/I" on page 41 (depending on the programming language being used) for details.

For a DAM data set, all data is read into or written from either of two areas in main storage:

- A file input/output area (FIOA)
- A file work area (FWA).

An FIOA is required to handle records that are unblocked, and that are read only.

An FWA is required to handle records that are blocked, that are to be added, or that are to be updated. In addition, an FWA is always used in a browse.

File control executes at the priority of the requesting program, under control of the task control area (the TCA) of the requesting program, saving and restoring registers from this TCA. The response to a request for file services can be checked as explained in "Test Response to a Request for File Services

(TYPE=CHECK)" on page 80. Control can be routed to any of various user written exception handling routines based on the outcome of the file operation.

Parameter values must be specified, when using the file control macro, either:

- By including the parameters in operands of the macro by which file services are requested, or
- By coding instructions that place the parameter values in fields of the TCA before issuing the macro.

The second of these approaches is provided to allow the application program to specify parameters that can only be determined during execution, for example, input messages from a terminal.

EXCLUSIVE CONTROL DEADLOCKS

CICS exclusive control serializes updates, additions, and deletions of individual records, maintaining such control until a sync point is taken or the transaction ends.

VSAM exclusive control, that operates in addition to CICS exclusive control, serializes based on a control interval that may contain more than one record. It distinguishes between shared use and exclusive use, and control lasts until the request has ended. The rules regarding exclusive control conflict, and the determination that a request has ended, vary according to the type of request and whether or not the data set is participating in VSAM Local Shared Resources (LSR).

While it is not necessary to understand the details of exclusive control conflicts in VSAM, the application programmer must follow certain rules in designing and programming, so as to reduce the probability of a deadlock in CICS or in VSAM. The rules are more stringent for data sets using LSR. However, LSR is widely used and provides many benefits. Even if the system programmer has not specified resource sharing for a data set (DFHFCT
| TYPE=DATASET,LSRPOOL=1), he may do so in the future. Therefore, it is strongly recommended that applications be written to execute correctly, without deadlock, in the LSR environment.

There are four distinct types of deadlock, and corresponding rules, when accessing VSAM data sets through DFHFC macros:

1. Two tasks are updating multiple resources. Each has CICS or VSAM exclusive control over one resource, but each needs the resource owned by the other in order to complete. This type of deadlock is more fully | explained in the appropriate <u>CICS Application Programmer's Reference Manual (Command Level)</u>.

   To prevent deadlock, all applications updating multiple resources should update them in the same order.

2. A task holds position for exclusive use over a control interval, and attempts to do another operation requiring exclusive use. If a task has done a GET UPDATE and attempts a GET UPDATE, DELETE, or PUT NEWREC; or if it has done a PUT for MASSINSERT and attempts a GET UPDATE, DELETE, or PUT NEWREC not for MASSINSERT, a deadlock may occur. Also, a second PUT for MASSINSERT with a nonascending key can cause a deadlock.

   To prevent deadlock, the programmer should release position before attempting another operation requiring exclusive use. Follow a GET UPDATE with a PUT UPDATE, PUT DELETE, or RELEASE. Follow a PUT for MASSINSERT by another PUT for MASSINSERT with an ascending key, or release position with a RELEASE.

3. A task holds position for exclusive use over a control interval, and attempts to do an operation requiring shared use, in the LSR environment. If a task has done a GET UPDATE or PUT for MASSINSERT, and attempts a browse (SETL) or non-update get (GET MOVE or GET LOCATE), a deadlock may occur.

   To prevent deadlock, the programmer should release position as described above before attempting any other operation on the data set, not just an operation requiring exclusive use.

4. A task holds position for shared use over a control interval, and attempts to do an operation requiring exclusive use in the LSR environment. If a task has done a SETL, RESETL, GETNEXT, GETPREV, or GET LOCATE, and attempts a GET UPDATE, DELETE, or PUT NEWREC, a deadlock may occur.

   To prevent deadlock, the programmer should release position by ending all browses on the data set with ESETL, or by ending a GET LOCATE with a RELEASE, before attempting an operation requiring exclusive use.

In summary, the application programmer will prevent exclusive control deadlocks by following these rules:

1. All applications should update multiple VSAM data sets in the same order.

2. Follow a GET UPDATE by a PUT UPDATE, PUT DELETE, or RELEASE before performing any other operation on the same data set.

3. Follow a PUT for MASSINSERT by another PUT for MASSINSERT with an ascending key, or release position with a RELEASE before performing any other operation on the same data set.

4. Before issuing a GET UPDATE, DELETE, or PUT NEWREC, end all browses of that data set with an ESETL.

5. Before issuing a GET UPDATE, DELETE, or PUT NEWREC, end a GET LOCATE to that data set with a RELEASE.

BROWSING

The application program can browse a data set. The file control macro is used to specify a starting point for the browse, request each succeeding, or preceding record, reset the starting point for the browse (if desired), and end the browse.

Browse operations are requested by the appropriate TYPE operands of the DFHFC macro; SETL, GETNEXT, GETPREV, RESETL, and ESETL. The capabilities associated with each are summarized below. Operands to request checking of a response can be specified with these macros as with other DFHFC macros (see "Test Response to a Request for File Services (TYPE=CHECK)" on page 80). Specific operands for each macro are discussed in detail at the end of the chapter.

When accessing a VSAM data set, the browse facility can be used to perform random skip-sequential processing in a forward direction only. The following steps are required:

1. Group several random requests into ascending key sequence.

2. Issue a DFHFC TYPE=SETL macro that finds the first required record. To achieve this, the record identification field pointed to by the RDIDADR operand should be initialized to the key of the required record.

3. Prior to each DFHFC TYPE=GETNEXT macro, place the key of the next required record into the record identification field.

This procedure allows quick direct access to a VSAM data set by reducing index search time. When the record having the highest key has been retrieved, an ESETL or RESETL should be issued to terminate or reset the operation.

A browse should always be terminated by an ESETL macro, but will also be terminated by the end of an LUW (that is, at a sync point), or by a normal or
| abnormal end of task.

ALTERNATE INDEXING

| Alternate indexing is a feature of VSAM,
| supported by CICS file control, that
| allows key-sequenced and entry-sequenced
| data sets to be accessed by one or more
| alternate paths. Each alternate index
| accesses the records in the base data
| set through a different, alternate, key
| within the record.

Also, a data set with an alternate index can have two or more records with the same alternate key. To retrieve the first record with the same key the DFHFC TYPE=GET macro with the DUPKEY operand is sufficient. However, to continue retrieving the remaining records with the same key, a browse operation must be initiated. The DUPKEY operand also must be specified on the appropriate macro. The records will be retrieved in the order in which they were added to the data set, the duplicate key condition being raised for each record except the last. When changing to the browse operation, the first record will be retrieved twice, once by the TYPE=GET and once by the browse.

Defining the alternate indexes as part
| of the upgrade set will eliminate the
possibility of one or more indexes becoming invalid whenever the data set
| is updated.

RECORD IDENTIFICATION FIELD

The record identification field is used by the application program to communicate to the file control program the identity, in the form of a key or address, of a specific record, or the starting point of a set of records, required in input/output operations. This field is identified by the RDIDADR operand of the DFHFC macro. The contents of this field should not be altered when doing a GET for UPDATE operation.

If multiple browse operations are performed concurrently by a single application program, a unique record identification field must exist for each operation. The application program must provide the storage area for the record identification field. Generally, this storage can be allocated within the transaction work area (TWA) of the TCA, or some area acquired dynamically by the application program. Because CICS application programs must be quasi-reenterable, it is not advisable to set up the record identification field within the application program.

For a VSAM data set, the record identification field contains either the key or the relative byte address of the desired record. If the generic key option is used, the first byte of the field must contain the length of the key, in binary, and the remainder of the field must contain the generic key.

A partial key may be used as a search argument in a browse operation referring to a VSAM data set.

For a DAM data set, the record identification field consists of three subfields that contain block reference information, a physical key (if keyed data sets are being used), and a deblocking argument (if blocked data sets are being used). These fields are as follows:

- **A block reference** for the data set is specified by the RELTYPE operand of the DFHFCT TYPE=DATASET system macro and may be one of the following:
  - Relative block (CICS/OS/VS only), 3-byte binary (RELTYPE=BLK)
  - Relative track and record, 2-byte TT, 1-byte R (RELTYPE=HEX)
  - Relative track and record (zoned decimal format), 6-byte TTTTTT, 2-byte RR (RELTYPE=DEC)
  - Actual address, 8-byte MBBCCHHR (RELTYPE omitted).
- **A physical key** is required only if the data set being accessed is written with recorded keys. This key must be the same length as specified in the BLKKEYL operand for the FCT entry that defines the data set. It must immediately follow the block reference.
- **A deblocking argument** is required only if the data set contains blocked records and specific logical records are to be retrieved from within a block. Not every record needs to be deblocked. If a deblocking argument is not specified, an entire block is read into an FIOA. The deblocking argument may be either a key or a relative record number, and is specified in the RETMETH operand of the DFHFC macro. If used, the deblocking argument must immediately follow the physical key (if present) or the block reference (if the physical key is not present).

  If the deblocking argument is a key, it must be the same length as specified in the KEYLEN operand of the FCT entry that defines the data set. The key used for deblocking need not be the same size as the physical record key (BLKKEYL).

Figure 11 on page 55 shows examples of record identification fields for a DAM data set.

DAM DATA SETS

Records in a nonkeyed DAM data set may be updated using either of two methods. One method is to issue a DFHFC TYPE=GET,TYPOPER=UPDATE to read the record, change the data in the FWA, and issue a DFHFC TYPE=PUT to update the record. This is the normal way that records are updated and should be used when portions of the record are to be changed and the contents of the record are unknown.

An alternative method may be used when the contents of the record to be updated are known, or when the entire record is to be changed, regardless of its contents. A DFHFC TYPE=GETAREA macro is used to acquire an FWA, the record is built in the FWA, and a DFHFC TYPE=PUT,TYPOPER=UPDATE is issued to write the data at the location specified in the record identification field, overwriting whatever was previously recorded at that location. Automatic logging must not be specified for files to be updated by this method.

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

1. When adding undefined or variable length records (keyed or nonkeyed), the application programmer must indicate the track on which each new record is to be added. If space is available on the track, the new 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 2-byte zoned decimal number in the seventh and eighth positions of the record identification field.

   In the CICS/DOS/VS system, an attempt to add a variable-length or undefined record is limited to the single track specified by the application programmer. If insufficient space is available on that track, a "no space available" error is returned, and the

```
| 0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 <——Bytes

| RELBLK# | N |          (CICS/OS/VS only)    Search by relative block;
                                              deblock by relative record

| RELBLK# |   KEY   |    (CICS/OS/VS only)    Search by relative block;
                                              deblock by key

| T  T  R |   PH-KEY   |   KEY   |            Search by relative track
                                              and record and key;
                                              deblock by key

| T  T  T  T  T  T  R  R |   PH-KEY   |   KEY   | Search by zoned
                                                  decimal relative
                                              track, record, and key.
                                              Deblock by key.

| T  T  R |   KEY   |                         Search by relative track
                                              and record; deblock by key

| M  B  B  C  C  H  H  R | N |                Search by address;
                                              deblock by relative record
```

Figure 11. Examples of Record Identification (DAM Data Set)

application programmer may then try to add the record on another track. Under these circumstances, the record is returned to the application program in an FWA, the address of which is at TCAFCAA. The programmer need only modify the track identification and issue another DFHFC TYPE=PUT,TYPOPER=NEWREC macro to add the record on another track.

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

2. The addition of keyed fixed-length records to DAM data sets requires that the data set first be formatted with dummy records or "slots" into which new records may be added. (The first byte of a dummy record is a key of X'FF's; in CICS/OS/VS, the first byte of data contains the record number.) A preformatted DAM data set cannot be added to by a COBOL batch program.

3. For nonkeyed, fixed-length records, the exact physical block reference must be given in the record identification field. The data in the new records is written in the exact location specified, overwriting the previous contents of that location.

4. For keyed, fixed-length record additions, only the track information is used as a starting location for the search of a dummy key and record. When a dummy key and record are found, the new key and record replace it. The location at which the new record is inserted is returned to the application program in the block reference subfield of the record identification field.

For example, suppose a user wishes to add a keyed, fixed-length record to a DAM data set. First, some algorithm determines that the search is to start at relative track 3. The record identification field of the new record might appear as follows:

```
0 3 0  ALPHA

T T R  KEY
```

When control is returned to the application program, the record identification field might reflect the fact that the record was added on relative track 4, record 6.

```
0 4 6  ALPHA

T T R  KEY
```

5. When adding records of undefined length, the length of the physical record must be placed in 2-byte binary format at TCAFCURL. When an undefined record is retrieved, the application program must determine its length.

6. When making additions to a DAM data set containing variable-length blocked or unblocked records, the application program must include a record descriptor field (RDF) that contains the length (LLbb) of the entire block to be written. Also, for each logical record within that block, an RDF must be included that contains the length of the logical record. Effectively, this allows the application to add a block containing multiple logical records, as shown in the following diagram:

```
|prefix| 96| 54|<---50--->| 24|<-- -
        |    |              |
     Block  Log rec        Log rec
      RDF   RDF            RDF
```

   If a single logical record only is to be added, the block RDF is still required, as shown in the following diagram:

```
|prefix|108|104|<--------100--------->|
        |    |
     Block  Log rec
      RDF   RDF
```

When updating records on a DAM data set, the following restriction applies:

If the file is blocked, and if two or more records are to be updated, a DFHFC TYPE=GET macro to retrieve a record must be followed by a DFHFC TYPE=PUT macro to write the updated record (or a DFHFC TYPE=RELEASE macro if the updated record is not required) before any further record in the same block is retrieved for update. Failure to do so will result either in one or more updates being lost or in a lockout.

DIRECT RETRIEVAL (TYPE=GET)

This macro is used for direct read-only (inquiry) or update (DFHFC TYPE=GET, TYPOPER=UPDATE) operations. The requested record is returned in:

- An FWA for update operations, read-only operations with blocked records, or for read-only operations with a blocked VSAM data set
- A VSWA for read-only operations in locate mode on the records of a VSAM data set.
- An FIOA for read-only operations with unblocked records from a VSAM data set in move mode or a DAM data set.

```
DFHFC TYPE=GET
      [,DATASET=symb-name]
      [,RDIDADR=symb-addr]
      [,TYPOPER=UPDATE]
      [,RETMETH={RELREC|KEY}]^1
      [,ARGTYP={KEY|RBA}]^2
      [,SRCHTYP=
         {FKEQ|FKGE|GKEQ|GKGE}]^2
      [,MODE={MOVE|LOCATE}]^2
      [,DUPKEY=symb-addr]^2
      [,NORESP=symb-addr]
      [,ERROR=symb-addr]
      [,DSIDER=symb-addr]
      [,NOTFND=symb-addr]
      [,INVREQ=symb-addr]
      [,IOERROR=symb-addr]
      [,NOTOPEN=symb-addr]
      [,ILLOGIC=symb-addr]^2

^1 DAM only
^2 VSAM only
```

Before this macro is used, instructions must be provided that define symbolically the required FWA, and/or VSWA, or FIOA, by:

1. Copying the appropriate storage definitions (DFHFWADS, and/or DFHVSWA, or DFHFIOA) provided by CICS

2. Providing storage definitions for the user's part of the FIOA, FWA, and/or the user's record in the VSAM buffer.

CICS performs the following services in response to a DFHFC TYPE=GET macro:

1. Acquires the appropriate main storage area (FWA, VSWA, or FIOA) required to read a record

2. Reads the requested record into that area

3. Makes the requested record available to the application program

The record required in an input/output operation is identified in a record identification field. The format of this field, as required for the various access methods, is described in "Record Identification Field" on page 53.

When a DAM data set is referenced, the record identification field should contain a block reference.

When a VSAM data set is referenced, the required record is accessed by either a relative byte address or a key. A search by key may be for a key equal to the search key or for one equal to or greater than the search key. A search may also be for a partial key (the first 2 bytes, or any number specified by the programmer), which may serve as a generic key. The generic or partial key search may, again, be either for an equal key or for an equal or greater key, but only the number of bytes specified will be compared.

In addition, CICS can acquire an FWA when the record is to be updated, or when records are blocked, depending on the operands included in the macro.

The length of the acquired FWA depends on whether or not the record is to be updated.

The FWA for a GET in move mode will also be large enough to contain a record of the maximum length defined in the FCT.

If the record is to be updated, the FWA acquired will be sufficient to contain a record of the maximum length specified by the system programmer in the FCT; otherwise, the FWA will be sufficient to contain the requested record.

When a record of a VSAM data set is retrieved in response to a read only request, move-mode or locate-mode processing can be specified. In move mode, the record is handled in the same way as a DAM record. In locate mode, the record is made available to the application program in the VSAM buffer. The application programmer must have copied the symbolic storage definition for the VSWA (DFHVSWA) and must also provide a symbolic storage definition for the record that is retrieved.

After requesting file services, the programmer must establish addressability for any required FIOA or FWA. The address of the area involved, provided by CICS at TCAFCAA, must be placed in FIOABAR or FWACBAR. In locate mode, the address of the VSWA is in TCAFCAA and must be placed in VSWABAR. The address of the area that holds the requested record is at VSWAREA within the VSWA and must be moved to the base locator that has been established for the symbolic storage definition of the area.

When retrieving variable length records from a VSAM data set in move mode, the file control program creates a length field and places it preceding the record in the FWA. The format of this length field is LLbb, where LL is a 2-byte binary length (including the 4 bytes for the length field itself) and bb is 2 bytes of binary zeros. In locate mode, the length is not included in the record itself but is placed at VSWALEN in the VSWA.

When a VSAM record is retrieved for update, VSAM maintains exclusive control of the control interval containing that record. A task should not attempt to retrieve (for update) a second record from the same control interval as a record it is already holding for update, otherwise a permanent wait will occur. The update should first be completed, by a DFHFC TYPE=PUT macro, or if it cannot be completed, terminated by a DFHFC TYPE=RELEASE macro.

A DFHFC TYPE=RELEASE macro frees an FIOA or FWA acquired in response to a request for file services, or a VSWA and VSAM string established for a VSAM read-only request using locate-mode I/O. Any of these areas that are not freed by the application program are freed by CICS at task termination.

DIRECT RETRIEVAL (READ-ONLY)

The following examples show how to retrieve a single record directly from a master data set, assuming blocked records.

ASM:

```
        COPY  DFHTCADS                  COPY TCA SYMBOLIC STRG DEFN
KEYF    DS    CL8                       RECORD IDENT FIELD IN TWA
FWACBAR EQU   7                         ASSIGN BASE REGISTER FOR FWA
        COPY  DFHFWADS                  SYMBOLICALLY DEFINE FWA
RECORD  DS    0CL350                    RECORD LAYOUT FOLLOWS CONTROL
        .                               FIELD AND HAS SAME BASE REGISTER
        .
        .
        MVC   KEYF,ACCTNO               MOVE RECORD IDENT TO KEY FIELD
READREC DFHFC TYPE=GET,                 GET RECORD FROM MASTER DATA SET      *
              DATASET=MASTERA,                                               *
              RDIDADR=KEYF
        L     FWACBAR,TCAFCAA           ESTABLISH ADDRESSABILITY FOR FWA
```

```
COBOL:

    02  FWACBAR PIC S9(8) COMP.
        .                         NOTE DEFINE BASE REGISTER FOR FWA.
        .
        .
01  DFHTCADS COPY DFHTCADS.       NOTE COPY SYMBOLIC STRG DEFN FOR TCA.
    02  KEYF PIC X(8).            NOTE DEFINE KEY FIELD IN TWA.
        .
        .
        .
01  DFHFWADS COPY DFHFWADS.       NOTE COPY SYMBOLIC STRG DEFN FOR FWA.
    02  RECORD PIC X(350).        NOTE DEFINE RECORD LAYOUT IN FWA.
        .
        .
        .
PROCEDURE DIVISION.
    MOVE CSACDTA TO TCACBAR.      NOTE ESTABLISH TCA ADDRESSABILITY.
        .
        .
        .
    MOVE ACCTNO TO KEYF.          NOTE MOVE RECORD IDENT TO KEY.
READREC.
        DFHFC TYPE=GET,           GET RECORD FROM MASTER DATA SET     *
              DATASET=MASTERA,                                        *
              RDIDADR=KEYF
    MOVE TCAFCAA TO FWACBAR.      NOTE ESTABLISH FWA ADDRESSABILITY.
```

```
PL/I:

%INCLUDE DFHTCADS;                /*COPY SYMBOLIC STRG DEFN FOR TCA*/
    02  KEYF CHAR(8);             /*DEFINE KEY FIELD IN TWA*/
%INCLUDE DFHFWADS;                /*COPY SYMBOLIC STRG DEFN FOR FWA*/
    02  RECORD CHAR(350);         /*DEFINE RECORD LAYOUT IN FWA*/
        .
        .
        .
KEYF=ACCTNO;                      /*ASSIGN RECORD IDENT TO KEY FIELD*/
READREC:
        DFHFC TYPE=GET,           GET RECORD FROM MASTER DATA SET     *
              DATASET=MASTERA,                                        *
              RDIDADR=KEYF
FWACBAR=TCAFCAA;                  /*ESTABLISH ADDRESSABILITY FOR FWA*/
```

DIRECT RETRIEVAL (VSAM LOCATE MODE)

The following examples show how to retrieve a single record directly from a VSAM data set using locate-mode I/O.

If the record is variable length, the LLbb field will not be part of the record. The length of the record can be found in VSWALEN in the VSWA.

```
ASM:

        COPY   DFHTCADS                COPY TCA SYMBOLIC STORAGE DEFN
KEYF    DS     CL8                     DEFINE KEY FIELD IN TWA
VSWABAR EQU    7                       ASSIGN BASE REGISTER FOR VSWA
RECBAR  EQU    8                       ASSIGN BASE REGISTER FOR RECORD
        COPY   DFHVSWA                 COPY VSWA SYMBOLIC DEFN
RECDS   DSECT                          DUMMY SECTION FOR RECORD
        USING  *,RECBAR                MAKE RECORD ADDRESSABLE
RECORD  DS     0CL350                  DEFINE RECORD LAYOUT
        .
        .
        .
        MVC    KEYF,ACCTNO             MOVE RECORD ID TO KEY FIELD
READREC DFHFC  TYPE=GET,               GET A RECORD FROM MASTER           *
               DATASET=MASTVSAM,           VSAM DATA SET USING            *
               RDIDADR=KEYF,               LOCATE MODE                    *
               MODE=LOCATE
        L      VSWABAR,TCAFCAA         ESTABLISH VSWA ADDRESSABILITY
        L      RECBAR,VSWAREA          ESTABLISH RECORD ADDRESSABILITY
        L      3,VSWALEN               LOAD RECORD LENGTH INTO WORK REG
```

COBOL:

```
    02  VSWABAR PIC S9(8) COMP.
                                    NOTE DEFINE BASE REGISTER FOR VSWA.
    02  RECBAR PIC S9(8) COMP.
        .                           NOTE DEFINE BASE REGISTER FOR RECORD.
        .
        .
01  DFHTCADS COPY DFHTCADS.         NOTE COPY SYMBOLIC STRG DEFN FOR TCA.
    02  KEYF PIC X(8).              NOTE DEFINE KEY FIELD IN TWA.
    02  RECLEN PIC S9(8) COMP.
        .                           NOTE DEFINE RECORD LENGTH WORK AREA.
        .
        .
01  DFHVSWA COPY DFHVSWA.           NOTE COPY SYMBOLIC STRG DEFN FOR VSWA.
01  RECDS SYNCHRONIZED.             NOTE DEFINE SYMBOLIC STRG DEFN FOR RECORD.
    02  RECORD PIC X(350).          NOTE DEFINE RECORD LAYOUT.
        .
        .
        .
PROCEDURE DIVISION.
    MOVE CSACDTA TO TCACBAR.        NOTE ESTABLISH TCA ADDRESSABILITY.
        .
        .
        .
    MOVE ACCTNO TO KEYF.            NOTE MOVE RECORD ID TO KEY FIELD.
READREC.
        DFHFC TYPE=GET,             GET A RECORD FROM MASTER           *
              DATASET=MASTVSAM,         VSAM DATA SET USING            *
              RDIDADR=KEYF,             LOCATE MODE                    *
              MODE=LOCATE
    MOVE TCAFCAA TO VSWABAR.        NOTE ESTABLISH VSWA ADDRESSABILITY.
    MOVE VSWAREA TO RECBAR.         NOTE ESTABLISH RECORD ADDRESSABILITY.
    MOVE VSWALEN TO RECLEN.         NOTE MOVE RECORD LENGTH TO WORK AREA.
```

PL/I:

```
%INCLUDE DFHTCADS;                  /*COPY SYMBOLIC STRG DEFN FOR TCA*/
    02  KEYF CHAR(8),               /*DEFINE KEY FIELD IN TWA*/
    02  RECLEN FIXED BINARY(31);    /*DEFINE RECORD LENGTH WORK AREA*/
%INCLUDE DFHVSWA;                   /*COPY SYMBOLIC STRG DEFN FOR VSWA*/
DCL 01 RECDS BASED (RECBAR),        /*DEFINE SYMB STRG DEFN FOR RECORD*/
    02  RECORD CHAR(350);           /*DEFINE RECORD LAYOUT*/
        .
        .
        .
KEYF=ACCTNO;                        /*MOVE RECORD ID TO KEY FIELD*/
READREC:
        DFHFC TYPE=GET,             GET A RECORD FROM MASTER           *
              DATASET=MASTVASM,         VSAM DATA SET USING            *
              RDIDADR=KEYF,             LOCATE MODE                    *
              MODE=LOCATE
VSWABAR=TCAFCAA;                    /*ESTAB ADDRESSABILITY FOR VSWA*/
RECBAR=VSWAREA;                     /*ESTAB ADDRESSABILITY FOR RECORD*/
RECLEN=VSWALEN;                     /*MOVE RECORD LENGTH TO WORK AREA*/
```

DIRECT RETRIEVAL (FOR UPDATE)

The following examples show how to retrieve a single record directly from a master data set for update.

```
ASM:

        COPY   DFHTCADS              COPY TCA SYMBOLIC STRG DEFN
KEYF    DS     CL8                   DEFINE KEY FIELD IN TWA
FWACBAR EQU    7                     ASSIGN BASE REGISTER FOR FWA
        COPY   DFHFWADS              SYMBOLICALLY DEFINE FWA
RECORD  DS     0CL350                RECORD LAYOUT FOLLOWS CONTROL
        .                            FIELD AND HAS SAME BASE REGISTER
        .
        .
        MVC    KEYF,ACCTNO           MOVE RECORD IDENT TO KEY FIELD
READREC DFHFC  TYPE=GET,             GET RECORD FROM MASTER DATA SET     *
               DATASET=MASTERA,      FOR UPDATE                          *
               RDIDADR=KEYF,                                             *
               TYPOPER=UPDATE
        L      FWACBAR,TCAFCAA       ESTABLISH ADDRESSABILITY FOR FWA
```

```
COBOL:

    02  FWACBAR PIC S9(8) COMP.
        .                            NOTE DEFINE BASE REGISTER FOR FWA.
        .
        .
01  DFHTCADS COPY DFHTCADS.          NOTE COPY SYMBOLIC STRG DEFN FOR TCA.
    02  KEYF PIC X(8).               NOTE DEFINE KEY FIELD IN TWA.
        .
        .
        .
01  DFHFWADS COPY DFHFWADS.          NOTE COPY SYMBOLIC STRG DEFN FOR FWA.
    02  RECORD PIC X(350).           NOTE DEFINE RECORD LAYOUT IN FWA.
        .
        .
        .
PROCEDURE DIVISION.
    MOVE CSACDTA TO TCACBAR.         NOTE ESTABLISH TCA ADDRESSABILITY.
        .
        .
        .
    MOVE ACCTNO TO KEYF.             NOTE MOVE RECORD IDENT TO KEY.
READREC.
        DFHFC TYPE=GET,              GET RECORD FROM MASTER DATA SET     *
              DATASET=MASTERA,                                           *
              RDIDADR=KEYF,                                              *
              TYPOPER=UPDATE
    MOVE TCAFCAA TO FWACBAR.         NOTE ESTABLISH FWA ADDRESSABILITY.
```

```
PL/I:

%INCLUDE DFHTCADS;                      /*COPY SYMBOLIC STRG DEFN FOR TCA*/
    02 KEYF CHAR(8);                    /*DEFINE KEY FIELD IN TWA*/
%INCLUDE DFHFWADS;                      /*COPY SYMBOLIC STRG DEFN FOR FWA*/
    02 RECORD CHAR(350);                /*DEFINE RECORD LAYOUT IN FWA*/
         .
         .
         .
KEYF=ACCTNO;                            /*ASSIGN RECORD IDENT TO KEY FIELD*/
READREC:
       DFHFC TYPE=GET,                  GET RECORD FROM MASTER DATA SET      *
             DATASET=MASTERA,                                                *
             RDIDADR=KEYF,                                                   *
             TYPOPER=UPDATE
FWACBAR=TCAFCAA;                        /*ESTABLISH ADDRESSABILITY FOR FWA*/
```

DIRECT ADDITION OR UPDATE (TYPE=PUT)

```
DFHFC TYPE=PUT
      [,RDIDADR=symb-addr]
      [,TYPOPER=
        {NEWREC|UPDATE|DELETE}]¹
      [,ARGTYP={KEY|RBA}]²
      [,NORESP=symb-addr]
      [,ERROR=symb-addr]
      [,DUPREC=symb-addr]
      [,INVREQ=symb-addr]
      [,IOERROR=symb-addr]
      [,NOSPACE=symb-addr]
      [,NOTOPEN=symb-addr]
      [,ILLOGIC=symb-addr]²

¹ DELETE can be used only with a
  VSAM KSDS or RRDS

² VSAM only (ARGTYP is only valid
  with TYPOPER=NEWREC)
```

This macro is used to:

- Add a new record to an existing data set
- Update an existing record that has been retrieved through the DFHFC TYPE=GET,TYPOPER=UPDATE macro
- Update an existing record in a nonkeyed DAM data set without first reading the record for update.

A DFHFC TYPE=PUT macro must never be issued without first issuing a DFHFC TYPE=GET,TYPOPER=UPDATE or DFHFC TYPE=GETAREA macro, because the results of such action are unpredictable.

When a VSAM key-sequenced or relative-record data set is being processed, a DFHFC TYPE=PUT,TYPOPER=DELETE macro can be used to delete a record previously retrieved by a DFHFC TYPE=GET,TYPOPER=UPDATE macro.

An FWA is used to contain the record to be written or updated. The first 16 bytes of the FWA form the CICS system section, which is followed by the record to be written to a data set.

CICS does the following in response to a DFHFC TYPE=PUT macro:

- Writes updated or new records in user defined data sets
- Acquires or locates the main storage and control blocks required to write the record
- Releases all data set storage associated with the request to write.

Before file services can be requested by means of the DFHFC TYPE=PUT macro, the application program must include instructions that do the following:

1. Symbolically define the FWA by (1) copying the appropriate system section storage definition (DFHFWADS), and (2) providing a storage definition for the user's section of the FWA.
2. Establish addressability for the new FWA by specifying a symbolic base address for the FWA.
3. Place the address of the FWA in TCAFCAA. For a request to add a new record, this address is returned to the application program by the preceding DFHFC TYPE=GETAREA request. For a request to update or delete a record, this address is made available to the application program in response to the preceding DFHFC TYPE=GET,TYPOPER=UPDATE request. It must have been stored

by the application program at that time, and should be moved to TCAFCAA immediately preceding the DFHFC TYPE=PUT request, with no intervening requests that could cause the contents of TCAFCAA to be altered.

If the records being written to a data set are undefined, the length of the record being written must be placed in TCAFCURL.

For records written to a variable length VSAM data set, the length of the record should be placed in an LLbb field in the beginning of the record. The field is 4 bytes long, the first 2 bytes containing the length in binary (including the 4 bytes for the length field) and the last 2 bytes set to binary zeros. This field is used by CICS to determine the length of the record and is not written to the data set.

VSAM does not allow an update operation on a control interval from which a record has already been retrieved for update. If a task attempts to perform an update operation on such a control interval before a previous record already held by the same task is updated by a DFHFC TYPE=PUT, or before the update is terminated by a DFHFC TYPE=RELEASE, the program will go into a permanent wait.

The programmer who is adding records to a DAM data set should also refer to "DAM Data Sets" earlier in the chapter.

The following examples show how to retrieve a record, update it, and return it to the data set.

```
ASM:

        COPY  DFHTCADS             COPY TCA SYMBOLIC STRG DEFN
KEYF    DS    CL8                  DEFINE KEY FIELD IN TWA
FWACBAR EQU   7                    ASSIGN BASE REGISTER FOR FWA
        COPY  DFHFWADS             SYMBOLICALLY DEFINE FWA
RECORD  DS    0CL350               RECORD LAYOUT FOLLOWS CONTROL
        .                          FIELD AND HAS SAME BASE REGISTER
        .
        .
READUPD DFHFC TYPE=GET,            READ RECORD FOR UPDATE             *
              DATASET=MASTERB,                                        *
              RDIDADR=KEYF,                                           *
              TYPOPER=UPDATE
        L     FWACBAR,TCAFCAA      ESTABLISH ADDRESSABILITY FOR FWA
        .
        .       (update record)
        .
        ST    FWACBAR,TCAFCAA      PLACE FWA ADDRESS IN TCA
WRITEUP DFHFC TYPE=PUT,            WRITE THE UPDATED RECORD           *
              RDIDADR=KEYF
```

COBOL:

```
    02  FWACBAR PIC S9(8) COMP.
        .                           NOTE DEFINE BASE REGISTER FOR FWA.
        .
        .
01  DFHTCADS COPY DFHTCADS.         NOTE COPY SYMBOLIC STRG DEFN FOR TCA.
    02  KEYF PIC X(8).              NOTE DEFINE KEY FIELD IN TWA.
        .
        .
        .
01  DFHFWADS COPY DFHFWADS.         NOTE COPY SYMBOLIC STRG DEFN FOR FWA.
    02  RECORD PIC X(350).          NOTE DEFINE RECORD LAYOUT IN FWA.
        .
        .
        .
PROCEDURE DIVISION.
    MOVE CSACDTA TO TCACBAR.        NOTE ESTABLISH TCA ADDRESSABILITY.
        .
        .
        .
READUPD.
       DFHFC TYPE=GET,              READ RECORD FOR UPDATE             *
             DATASET=MASTERB,                                          *
             RDIDADR=KEYF,                                             *
             TYPOPER=UPDATE
    MOVE TCAFCAA TO FWACBAR.        NOTE ESTABLISH FWA ADDRESSABILITY.
        .
        .  (update record)
        .
    MOVE FWACBAR TO TCAFCAA.        NOTE MOVE ADDRESS OF FWA TO TCA.
WRITEUP.
        DFHFC TYPE=PUT,             WRITE THE UPDATED RECORD           *
              RDIDADR=KEYF
```

PL/I:

```
%INCLUDE DFHTCADS;                  /*COPY SYMBOLIC STRG DEFN FOR TCA*/
    02 KEYF CHAR(8);                /*DEFINE KEY FIELD IN TWA*/
%INCLUDE DFHFWADS;                  /*COPY SYMBOLIC STRG DEFN FOR FWA*/
    02 RECORD CHAR(350);            /*DEFINE RECORD LAYOUT IN FWA*/
        .
        .
READUPD:
        DFHFC TYPE=GET,             READ RECORD FOR UPDATE             *
              DATASET=MASTERB,                                         *
              RDIDADR=KEYF,                                            *
              TYPOPER=UPDATE
FWACBAR=TCAFCAA;                    /*ESTABLISH ADDRESSABILITY FOR FWA*/
        .
        .  (update record)
        .
TCAFCAA=FWACBAR;                    /*PLACE ADDR OF WORK AREA IN TCA*/
WRITEUP:
        DFHFC TYPE=PUT,             WRITE THE UPDATED RECORD           *
              RDIDADR=KEYF
```

DIRECT DELETION, VSAM ONLY (TYPE=DELETE)

```
DFHFC TYPE=DELETE
      [,DATASET=symb-name]
      [,RDIDADR=symb-addr]
      [,ARGTYP=KEY]
      [,SRCHTYP=FKEQ|GKEQ}]
      [,NORESP=symb-addr]
      [,ERROR=symb-addr]
      [,DSIDER=symb-addr]
      [,NOTFND=symb-addr]
      [,INVREQ=symb-addr]
      [,IOERROR=symb-addr]
      [,NOTOPEN=symb-addr]
      [,ILLOGIC=symb-addr]
```

This macro is used to perform the following functions on KSDS and RRDS data sets only:

- Delete a single record.
- Delete a group of records that share the same partial key; that is where the first part of the keys is the same. This is called generic delete.

To delete a single record, the key must be placed in an area pointed to by the RDIDADR operand.

To delete a group of records with the same partial key, that is where the first part of the keys is the same, the partial key must be placed in an area pointed to by the RDIDADR operand. The binary length of the key must be placed in the first byte of the area pointed to by the RDIDADR operand. SRCHTYP=GKEQ must be specified.

Neither an FIOA nor an FWA is required for a delete operation.

Note that a DELETE operation is an update operation, and therefore the control interval concerned is held under exclusive control. Exclusive control is released either by successful completion of the DELETE operation, or failing this, by issuing a DFHFC TYPE=RELEASE macro.

OBTAIN A FILE WORK AREA (TYPE=GETAREA)

```
DFHFC TYPE=GETAREA
      [,DATASET=symb-name]
      [,INITIMG={value|YES}]
      [,ARGTYP={KEY|RBA}]²
      [,TYPOPER=MASSINSERT]¹
      [,NORESP=symb-addr]
      [,ERROR=symb-addr]
      [,DSIDER=symb-addr]
      [,INVREQ=symb-addr]
      [,NOTOPEN=symb-addr]

 ¹ VSAM only

 ² ARGTYP is only valid
   with TYPOPER=MASSINSERT
```

This macro is used to obtain an FWA. (A storage control DFHSC TYPE=GETMAIN request cannot be used for file operations.)

CICS performs the following services in response to a DFHFC TYPE=GETAREA macro:

1. Acquires main storage (an FWA) for the creation of a new record
2. Includes and initializes the FWA control fields (a 16-byte prefix to the FWA) required by file control.

If several new records whose keys are in ascending sequence are to be added to a VSAM data set, the TYPOPER=MASSINSERT operand should be used, in which case, the FWA is retained and made available to the application program after each DFHFC TYPE=PUT macro that adds a record to the data set.

A mass insert operation is terminated by a DFHFC TYPE=RELEASE macro. A lockout condition will occur if more than one transaction is simultaneously attempting to perform a mass insert 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.

In a DFHFC TYPE=GETAREA macro, the ARGTYP operand is only applicable when TYPOPER=MASSINSERT has been specified.

When the DFHFC TYPE=GETAREA macro is used, the application program must include instructions that do the following:

- Symbolically define the FWA by (1) copying the appropriate CICS system section storage definition (DFHFWADS), and (2) providing a storage definition for the user's section of the FWA.

- Establish addressability for the new FWA by specifying a symbolic base address for the FWA. (The address of the area involved, returned by CICS at TCAFCAA, must be placed in FWACBAR.)

The following examples show how to get an FWA, build a new record in the FWA, and write that record to a data set.

```
ASM:

         COPY    DFHTCADS            COPY TCA SYMBOLIC STRG DEFN
KEYF     DS    CL8                   DEFINE KEY FIELD IN TWA
FWACBAR  EQU   7                     ASSIGN BASE REGISTER FOR FWA
         COPY  DFHFWADS              SYMBOLICALLY DEFINE FWA
RECORD   DS    0CL350                RECORD LAYOUT FOLLOWS CONTROL
         .                           FIELD AND HAS SAME BASE REGISTER
         .
         .
NEWREC   DFHFC TYPE=GETAREA,         GET AN FWA TO CREATE A NEW        *
               DATASET=MASTERC       RECORD FOR A DATA SET
         L     FWACBAR,TCAFCAA       ESTABLISH ADDRESSABILITY FOR FWA
         .
         .     (build new record)
         .
         ST    FWACBAR,TCAFCAA       PLACE ADDR OF NEW RECORD IN TCA
WRITNEW  DFHFC TYPE=PUT,             WRITE THE NEW RECORD              *
               TYPOPER=NEWREC,                                         *
               RDIDADR=KEYF
```

```
COBOL:

   02 FWACBAR PIC S9(8) COMP.
      .                              NOTE DEFINE BASE REGISTER FOR FWA.
      .
      .
01 DFHTCADS COPY DFHTCADS.           NOTE COPY SYMBOLIC STRG DEFN FOR TCA.
   02 KEYF PIC X(8).                 NOTE DEFINE KEY FIELD IN TWA.
      .
      .
      .
01 DFHFWADS COPY DFHFWADS.           NOTE COPY SYMBOLIC STRG DEFN FOR FWA.
   02 RECORD PIC X(350).             NOTE DEFINE RECORD LAYOUT IN FWA.
      .
      .
      .
PROCEDURE DIVISION.
   MOVE CSACDTA TO TCACBAR.          NOTE ESTABLISH TCA ADDRESSABILITY.
      .
      .
      .
NEWREC.
       DFHFC TYPE=GETAREA,           OBTAIN A FWA TO CREATE A NEW        *
             DATASET=MASTERC         RECORD FOR A DATA SET
   MOVE TCAFCAA TO FWACBAR.          NOTE ESTABLISH FWA ADDRESSABILITY.
      .
      .   (build new record)
      .
   MOVE FWACBAR TO TCAFCAA.          NOTE ADDRESS OF NEW RECORD TO TCA.
WRITNEW.
        DFHFC TYPE=PUT,              WRITE THE NEW RECORD                *
              TYPOPER=NEWREC,                                            *
              RDIDADR=KEYF
```

```
PL/I:

%INCLUDE DFHTCADS;                /*COPY SYMBOLIC STRG DEFN FOR TCA*/
   02 KEYF CHAR(8);               /*DEFINE KEY FIELD IN TWA*/
%INCLUDE DFHFWADS;                /*COPY SYMBOLIC STRG DEFN FOR FWA*/
   02 RECORD CHAR(350);           /*DEFINE RECORD LAYOUT IN FWA*/
       .
       .
       .
NEWREC:
        DFHFC TYPE=GETAREA,        GET AN FWA TO CREATE A NEW           *
              DATASET=MASTERC      RECORD FOR A DATA SET
FWACBAR=TCAFCAA;                  /*ESTABLISH ADDRESSABILITY FOR FWA*/
       .
       .    (build new record)
       .
TCAFCAA=FWACBAR;                  /*PLACE ADDR OF NEW RECORD IN TCA*/
WRITNEW:
        DFHFC TYPE=PUT,            WRITE THE NEW RECORD                 *
              TYPOPER=NEWREC,                                           *
              RDIDADR=KEYF
```

RELEASE STORAGE/EXCLUSIVE CONTROL (TYPE=RELEASE)

The syntax of the DFHFC TYPE=RELEASE macro is as follows:

```
DFHFC TYPE=RELEASE
      [,NORESP=symb-addr]
      [,ERROR=symb-addr]
      [,INVREQ=symb-addr]
      [,IOERROR=symb-addr]
      [,ILLOGIC=symb-addr]¹

¹ VSAM only
```

If the storage area to be released contains a record that has been read for update (by means of a DFHFC TYPE=GET,TYPOPER=UPDATE macro), and the update is no longer required, this macro will release the record from exclusive control as well as free the storage areas associated with it.

Before the DFHFC TYPE=RELEASE macro is executed, the address of the FWA, FIOA, or VSWA to be released must be moved to TCAFCAA. Any associated areas are also released.

A mass insert operation on a VSAM data set (initiated by the TYPOPER=MASSINSERT operand, followed by DFHFC TYPE=PUT,TYPOPER=NEWREC macros) is terminated by a DFHFC TYPE=RELEASE macro.

A DFHFC TYPE=RELEASE macro should also be used to release the VSWA established by CICS in response to a read-only request for a VSAM data set record retrieved in locate mode. Failure to release the VSWA may cause significant performance degradation or task suspension if subsequent accesses are made to the file.

The DFHFC TYPE=RELEASE macro should not be specified if the DFHFC TYPE=PUT,TYPOPER=UPDATE macro is used to perform a successful write of an updated record back to a data set. CICS automatically releases all storage associated with the write operation. However, if an error condition occurs, preventing successful completion of the write, a DFHFC TYPE=RELEASE macro should be issued to release the storage.

DFHFC TYPE=RELEASE must be issued whenever a DUPREC, ILLOGIC, IOERROR, or NOTFND condition occurs, even if UPDATE is not specified in the GET.

For further details of these conditions, see "Operands of DFHFC Macro" on page 81.

CICS performs the following services in response to a DFHFC TYPE=RELEASE macro:

- Releases an FWA, FIOA, and/or VSWA
- Releases a VSAM string, if a VSWA is released
- Releases exclusive control of a record retrieved for update (if applicable).

Note, though, that for a file with auto-logging specified (by the system programmer), the resource remains under the task control enqueue until either a sync point is issued or end of task is reached.

There is a limit to the number of VSAM strings that may be in use at any one time, determined by the STRNO operand of the DFHFCT TYPE=DATASET system macro. If strings are not released when no longer required, tasks may have to wait unnecessarily owing to the strings all being in use.

Any FWAs, FIOAs, VSWAs, and VSAM strings acquired during execution of a task are automatically released at termination of the task, if not released earlier in response to a DFHFC TYPE=RELEASE macro.

The following examples show how to request the release of an FWA.

ASM:

```
FWACBAR EQU   7                  ASSIGN BASE REGISTER FOR FWA
        COPY  DFHFWADS           SYMBOLICALLY DEFINE FWA
RECORD  DS    0CL350             RECORD LAYOUT FOLLOWS CONTROL
        .                        FIELD AND HAS SAME BASE REGISTER
        .
        .
        ST    FWACBAR,TCAFCAA    ADDRESS OF FWA TO BE RELEASED
RLSEREC DFHFC TYPE=RELEASE       IN TCA AND ISSUE RELEASE REQUEST
```

```
COBOL:

   02 FWACBAR PIC S9(8) COMP.
      .                                 NOTE DEFINE BASE REGISTER FOR FWA.
      .
      .
  01 DFHFWADS COPY DFHFWADS.            NOTE COPY SYMBOLIC STRG DEFN FOR FWA.
   02 RECORD PIC X(350).                NOTE DEFINE RECORD LAYOUT IN FWA.
      .
      .
      .
PROCEDURE DIVISION.
   MOVE CSACDTA TO TCACBAR.             NOTE ESTABLISH TCA ADDRESSABILITY.
      .
      .
      .
   MOVE FWACBAR TO TCAFCAA.             NOTE ADDR OF FWA TO BE RELEASED.
RLSEREC.
        DFHFC TYPE=RELEASE              ISSUE RELEASE REQUEST
```

```
PL/I:

%INCLUDE DFHTCADS;                      /*COPY SYMBOLIC STRG DEFN FOR TCA*/
      .
      .
      .
%INCLUDE DFHFWADS;                      /*COPY SYMBOLIC STRG DEFN FOR FWA*/
   02 RECORD CHAR(350);                 /*DEFINE RECORD LAYOUT IN FWA*/
      .
      .
      .
TCAFCAA=FWACBAR;                        /*ADDRESS OF FWA TO BE RELEASED*/
RLSEREC:
        DFHFC TYPE=RELEASE              ISSUE RELEASE REQUEST
```

INITIATE BROWSE (TYPE=SETL)

```
DFHFC TYPE=SETL
      [,DATASET=symb-name]
|     [,RDIDADR=symb-addr]
      [,RETMETH={RELREC|KEY}]¹
      [,ARGTYP={KEY|RBA}]²
      [,SRCHTYP=
        {FKEQ|FKGE|GKEQ|GKGE}]²
      [,MODE={MOVE|LOCATE}]²
      [,NORESP=symb-addr]
|     [,ERROR=symb-addr]
      [,DSIDER=symb-addr]
      [,NOTFND=symb-addr]
      [,INVREQ=symb-addr]
      [,IOERROR=symb-addr]
      [,NOTOPEN=symb-addr]
      [,ILLOGIC=symb-addr]²

¹ DAM only
² VSAM only
```

This macro is used to establish the position within the data set where the browse operation is to begin. It must be issued before any DFHFC TYPE=GETNEXT macro; however, no data is available until a DFHFC TYPE=GETNEXT is used.

The starting point within a data set for a browse operation is identified by a record identification field established for the data set. See "Record Identification Field" on page 53.

| For a DAM data set, the record identification field must contain a block reference (for example, TTR or MBBCCHHR) that conforms to the addressing method defined for that data set. Processing begins with the specified block and continues with each subsequent block until the browse operation 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.

For a VSAM data set, the contents of the record identification field may be a key, a relative byte address, or a relative record number. If the field contains a relative byte address, the browse begins at the specified address. If the field contains a key, it may be either specific or generic. If the key is generic, the length of the partial key is specified in the first byte of the record identification field.

In either case, the application program can specify that the browse operation is to begin at the first record having a key that is:

- Equal to the key in the record identification field (for generic, compared on only the number of bytes specified), or
- Equal to or greater than the key in the record identification field (again, for generic, compared on only the bytes specified).

When the DFHFC TYPE=SETL macro is used, the application programmer must provide instructions that do the following:

- Symbolically define the FWA by (1) copying the appropriate CICS system section storage definition (DFHFWADS), and (2) providing his own storage definition for the user's section of the FWA.
- Establish addressability for the FWA by specifying a symbolic base address for the FWA, typically following the DFHFC macro. (The address of the FWA, provided by CICS at TCAFCAA, must be placed at FWACBAR upon normal return from execution of the SETL macro.)

In most cases, records retrieved during a browse operation are returned to the application program in a FWA. However, in locate mode the addresses of the record are passed in the VSWA. The FWA allocated by CICS following a SETL request is unique for the duration of that particular browse operation. If the application program issues another SETL request, for the same or another data set, a different FWA is created by CICS. Thus it is possible for a single application program to concurrently browse the same data set at several different locations.

CICS performs the following services in response to a DFHFC TYPE=SETL macro:

1. Acquires the main storage I/O areas and work areas to be associated with this browse operation
2. Returns the address of the allocated FWA in TCAFCAA for other than locate-mode VSAM data set processing; returns the address of the allocated VSWA that will contain the VSAM buffer-area address of each retrieved record for locate-mode VSAM data set processing.

The information supplied by the user in the record identification field is preserved by CICS for use when GETNEXT requests are issued. Since CICS places into this field the identification of each record retrieved in response to a subsequent GETNEXT request, the field should not be released by the application program.

The information placed into the record identification field by CICS is always in a form that completely identifies the record. For example, assume a browse operation is to start with the first record of a blocked, keyed DAM data set. Before issuing the DFHFC TYPE=SETL macro, the application programmer should place the TTR (assuming that is the addressing method) of the first block into the record identification field. After executing each DFHFC TYPE=GETNEXT macro, CICS places the complete record identification into the record identification field. After the first GETNEXT, 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, if the application program is browsing a blocked, nonkeyed DAM data set and the second record from the second physical block on the third relative track is read in response to a GETNEXT request, the record identification field contains

```
X'00020201'
```

upon return to the application program, where 0002 represents the track, 02 represents the block, and 01 represents the record within the block.

The following examples show how to initiate a browse operation.

```
ASM:

        COPY   DFHTCADS             COPY TCA SYMBOLIC STORAGE DEFN
KEYF    DS     CL8
FWACBAR EQU    7                    ASSIGN BASE REGISTER FOR FWA
        COPY   DFHFWADS             DEFINE SYSTEM SECTION OF FWA
RECORD  DS     0CL350               RECORD LAYOUT
        .
        .
        CSECT
        .
        .
        MVC    KEYF(5),=C'JONES'
        XC     KEYF+5(3),KEYF+5     INITIALIZE KEY FIELD
START   DFHFC  TYPE=SETL,           INITIATE BROWSE                   *
               DATASET=MASTER,                                        *
               RDIDADR=KEYF,                                          *
               NOTOPEN=ERROR        GO TO ERROR LABEL IF ERROR
        L      FWACBAR,TCAFCAA      ESTABLISH ADDRESSABILITY FOR FWA
        .
        .
ERROR   DS     0H                   ENTRY TO ERROR ROUTINE
        .
        .
```

```
COBOL:

   02 FWACBAR PIC S9(8) COMP.
      .                              NOTE DEFINE BASE REGISTER FOR FWA.
      .
01 DFHTCADS COPY DFHTCADS.           NOTE COPY SYMBOLIC STRG DEFN FOR TCA.
   02 KEYF PIC X(8).                 NOTE DEFINE KEY FIELD IN TWA.
01 DFHFWADS COPY DFHFWADS.           NOTE COPY SYMBOLIC STRG DEFN FOR FWA.
   02 RECORD PIC X(350).             NOTE DEFINE RECORD LAYOUT IN FWA.
      .
      .
PROCEDURE DIVISION.
   MOVE CSACDTA TO TCACBAR.          NOTE ESTABLISH TCA ADDRESSABILITY.
      .
      .
   MOVE 'JONES' TO KEYF.
START.
        DFHFC TYPE=SETL,             INITIATE BROWSE                  *
              DATASET=MASTER,                                         *
              RDIDADR=KEYF,                                           *
              NOTOPEN=ERROR          GO TO ERROR LABEL IF ERROR
   MOVE TCAFCAA TO FWACBAR.
      .
      .
ERROR.
      .
      .
```

```
PL/I:

%INCLUDE DFHTCADS;                  /*COPY SYMBOLIC STRG DEFN FOR TCA*/
   02 KEYF CHAR(8);
      .
      .
%INCLUDE DFHFWADS;                  /*COPY SYMBOLIC STRG DEFN FOR FWA*/
   02 RECORD CHAR(350);             /*DEFINE RECORD LAYOUT IN FWA*/
      .
      .
KEYF='JONES';
START:
         DFHFC TYPE=SETL,           INITIATE BROWSE                   *
               DATASET=MASTER,                                        *
               RDIDADR=KEYF,                                          *
               NOTOPEN=ERROR        GO TO ERROR LABEL IF ERROR
FWACBAR=TCAFCAA;
      .
      .
ERROR:
      .
      .
```

FORWARD BROWSE (TYPE=GETNEXT)

```
DFHFC TYPE=GETNEXT
      [,DUPKEY=symb-addr][1]
      [,NORESP=symb-addr]
      [,ERROR=symb-addr]
      [,NOTFND=symb-addr]
      [,INVREQ=symb-addr]
      [,IOERROR=symb-addr]
      [,NOTOPEN=symb-addr]
      [,ENDFILE=symb-addr]
      [,ILLOGIC=symb-addr]

[1] VSAM only
```

This macro can also be used to perform skip-sequential processing upon a VSAM data set. After a DFHFC TYPE=SETL macro has been issued to initiate a browse, the next (or first) record in ascending sequence can be obtained by issuing the DFHFC TYPE=GETNEXT macro. For a DAM data set, CICS acquires the first record specified by the user. Each subsequent GETNEXT request causes CICS to acquire the next record in ascending sequence.

When VSAM is used, a browse operation can be specified to begin at a particular relative byte location or with a record identified by a key. In the former case, the first GETNEXT request retrieves that record. Each succeeding GETNEXT retrieves the next record in ascending sequence.

If a key is specified for a VSAM data set, it may be either specific or generic, and the application programmer can specify that the search begin (1) at a record having a key equal to the specific or generic key, or (2) at a record having a key equal to or greater than the specific or generic key. The effects of GETNEXT macros are as described below.

Before issuing the DFHFC TYPE=GETNEXT macro, the application programmer must place the address of the FWA associated with the particular operation in TCAFCAA. If the application program has initiated multiple browse operations, it must keep track of the FWA associated with each operation and refer to a specific FWA when requiring services related to that browse. Similar requirements apply to the address of a specific VSWA in locate-mode processing of VSAM records.

CICS performs the following services in response to a DFHFC TYPE=GETNEXT macro referring to a VSAM or DAM data set:

1. Retrieves the next sequential record and places it in the FWA specified by the user at TCAFCAA
2. Places the identification (key, block identification, or the like) of the record just retrieved into the record identification field specified in the DFHFC TYPE=SETL request initiating the browse.

If the user issues a DFHFC TYPE=GET,TYPOPER=UPDATE request on the record returned in response to a DFHFC TYPE=GETNEXT request, the address of the record identification field can be specified in the DFHFC TYPE=GET request. The update operation cannot be processed immediately following the GETNEXT macro. The browse must first be completed.

The first DFHFC TYPE=GETNEXT macro referring to a VSAM data set retrieves the record located in response to the DFHFC TYPE=SETL macro initiating the browse. On a subsequent GETNEXT, CICS checks the contents of the record identification field set aside for records of the data set. If this field contains the identifier of the record previously received, CICS retrieves the next logical record in sequence and places the identifier of that record in the record identification field. Sequential retrieval such as described above for DAM data sets then occurs.

It is possible, however, when using VSAM data sets, for the application programmer to utilize skip-sequential processing. All that is needed is to place the identification of the next record desired into the record identification field before issuing a GETNEXT macro. If, upon checking this field, CICS determines that its contents have been changed by the application program, CICS accesses the record having the identification currently stored in the record identification field. This record need not be the next sequential record in the data set. Skip-sequential processing is available only for VSAM data sets.

When VSAM skip-sequential processing is used, the record identification placed in the record identification field before issuing the GETNEXT request must be of the same form as that specified in the SETL or last RESETL request for this browse operation. That is, if the SETL or last RESETL specified a generic key, then the new record identification must be a generic key. It need not be the same length as that specified in the SETL or last RESETL. If the SETL or last RESETL specified an RBA, the new identification must be an RBA. Note that if the SETL or last RESETL specified an equal search (FKEQ or GKEQ), a GETNEXT request using skip-sequential processing may result in a NOTFND (record not found) condition.

If the NOTFND condition occurs during a browse operation, the application program must issue either a RESETL macro to reset the browse or an ESETL macro to terminate the browse. Both these macros are discussed later in the chapter.

The following examples show how to initiate a browse operation and retrieve successive records from the data set.

```
ASM:

        COPY   DFHTCADS            COPY TCA SYMBOLIC STRG DEFN
KEYF    DS     8X                  DEFINE KEY FIELD IN TWA
FWACBAR EQU    7                   ASSIGN FWA BASE REGISTER
        COPY   DFHFWADS            COPY CICS CONTROL SECTION OF FWA
RECORDA DS     0CL350              DEFINE RECORD LAYOUT IN FWA.
        .
        .
        .
        CSECT
        MVC    KEYF(8),=8X'00'     START AT BEGINNING OF DATA SET
INITIAL DFHFC  TYPE=SETL,          INITIATE BROWSE                      *
               DATASET=MASTER,                                          *
               RDIDADR=KEYF
        L      FWACBAR,TCAFCAA     ESTABLISH FWA BASE REGISTER
        .
        .
        .
        ST     FWACBAR,TCAFCAA     RESTORE FWA ADDRESS
        DFHFC  TYPE=GETNEXT        GET NEXT SEQUENTIAL RECORD
        L      FWACBAR,TCAFCAA     ASSURE ADDRESSABILITY
        .
        .
        .
```

COBOL:

```
   02 FWACBAR PIC S9(8) COMP.
         .                          NOTE DEFINE BASE REGISTER FOR FWA.
         .
01 DFHTCADS COPY DFHTCADS.          NOTE COPY SYMBOLIC STRG DEFN FOR TCA.
   02 KEYF PIC S9(18) COMP.
                                    NOTE DEFINE KEY FIELD IN TWA.
         .
         .
         .
01 DFHFWADS COPY DFHFWADS.          NOTE COPY SYMBOLIC STRG DEFN FOR FWA.
   02 RECORD PIC X(350).            NOTE DEFINE RECORD LAYOUT IN FWA.
         .
         .
         .
PROCEDURE DIVISION.
   MOVE CSACDTA TO TCACBAR.         NOTE ESTABLISH TCA ADDRESSABILITY.
         .
         .
         .
   MOVE 0 TO KEYF.                  NOTE START AT BEGINNING OF DATA SET.
         .
         .
INITIAL.
         DFHFC TYPE=SETL,           INITIATE BROWSE                    *
               DATASET=MASTER,                                         *
               RDIDADR=KEYF
   MOVE TCAFCAA TO FWACBAR.         NOTE ESTABLISH FWA ADDRESSABILITY.
         .
         .
         .
   MOVE FWACBAR TO TCAFCAA.
         DFHFC TYPE=GETNEXT         GET NEXT SEQUENTIAL RECORD.
   MOVE TCAFCAA TO FWACBAR.
         .
         .
```

```
PL/I:

%INCLUDE DFHTCADS;                /*COPY SYMBOLIC STRG DEFN FOR TCA*/
   02 KEYF CHAR(8);               /*DEFINE KEY FIELD IN TWA*/
        .
        .
        .
%INCLUDE DFHFWADS;                /*COPY SYMBOLIC STRG DEFN FOR FWA*/
   02 RECORD CHAR(350);           /*DEFINE RECORD LAYOUT IN FWA*/
        .
        .
        .
KEYF=LOW(8);                      /*START AT BEGINNING OF DATA SET*/
        .
        .
        .
INITIAL:
        DFHFC TYPE=SETL,          INITIATE BROWSE                    *
              DATASET=MASTER,                                        *
              RDIDADR=KEYF
FWACBAR=TCAFCAA;                  /*ESTABLISH FWA ADDRESSABILITY*/
        .
        .
        .
TCAFCAA=FWACBAR;
        DFHFC TYPE=GETNEXT        GET NEXT SEQUENTIAL RECORD
FWACBAR=TCAFCAA;
        .
        .
```

BACKWARD BROWSE, VSAM AND ASSEMBLER LANGUAGE ONLY (TYPE=GETPREV)

```
DFHFC TYPE=GETPREV
      [,DUPKEY=symb-addr]
      [,NORESP=symb-addr]
      [,ERROR=symb-addr]
      [,NOTFND=symb-addr]
      [,INVREQ=symb-addr]
      [,IOERROR=symb-addr]
      [,NOTOPEN=symb-addr]
      [,ENDFILE=symb-addr]
      [,ILLOGIC=symb-addr]
```

After a DFHFC TYPE=SETL macro has been issued to initiate a browse operation, the next (or first) record in descending sequence can be obtained by issuing the DFHFC TYPE=GETPREV macro.

A browse operation can be specified to begin at a particular relative byte location or with a record identified by a key. In the former case, the first GETPREV request retrieves that record. Each succeeding GETPREV retrieves the next record in descending sequence.

If a key is specified for a VSAM data set, it must be specific, and the application programmer can specify that the search begin at a record having a key equal to the specified key. The effects of GETPREV macros are as described below.

Before issuing the DFHFC TYPE=GETPREV macro, the application programmer must place the address of the FWA associated with the particular operation in TCAFCAA. If the application program has initiated multiple browse operations, it must keep track of the FWA associated with each operation and refer to a specific FWA when requiring services related to that browse. Similar requirements apply to the address of a specific VSWA in locate-mode browsing of VSAM records.

CICS performs the following services in response to a DFHFC TYPE=GETPREV macro referring to a VSAM data set:

1. Retrieves the next record in descending sequence and places it in the FWA specified by the user at TCAFCAA
2. Places the identification (key, or relative byte address) of the record just retrieved into the record identification field specified in the DFHFC TYPE=SETL request initiating the browse.

If the user issues a DFHFC TYPE=GET,TYPOPER=UPDATE request on the record returned in response to a DFHFC TYPE=GETPREV request, the address of the record identification field can be specified in the DFHFC TYPE=GET request.

The update operation cannot be processed immediately following the GETPREV macro. The browse must first be completed.

The first DFHFC TYPE=GETPREV macro retrieves the record located in response to the DFHFC TYPE=SETL macro initiating the browse. On a subsequent GETPREV, CICS checks the contents of the record identification field set aside for records of the data set. If this field contains the identifier of the record previously received, CICS retrieves the next logical record in sequence and places the identifier of that record in the record identification field.

If the DFHFC TYPE=GETPREV macro is issued following a DFHFC TYPE=SETL macro using a generic key, an invalid request will be indicated.

TERMINATE BROWSE (TYPE=ESETL)

```
DFHFC TYPE=ESETL
      [,NORESP=symb-addr]
      [,ERROR=symb-addr]
      [,INVREQ=symb-addr]
      [,ILLOGIC=symb-addr][1]

[1] VSAM only
```

Before this macro is issued, the programmer must ensure that TCAFCAA contains the address of the FWA associated with the browse operation he wishes to terminate.

When locate-mode processing of VSAM records is used, TCAFCAA must contain the address of the VSWA associated with the browse operation being terminated. In response to an ESETL request, CICS releases all I/O and work areas associated with the browse operation.

The following examples show how to end two concurrent browse operations.

```
ASM:

        COPY  DFHTCADS            COPY TCA SYMBOLIC STRG DEFN
FWACELL1 DS   A                   CONTAINS ADDR OF FWA USED
*                                 FOR FIRST BROWSE OPERATION
FWACELL2 DS   A                   CONTAINS ADDR OF FWA USED
*                                 FOR SECOND BROWSE OPERATION
FWACBAR EQU   7                   ASSIGN FWA BASE REGISTER
        COPY  DFHFWADS            DEFINE FWA SYMBOLIC STORAGE DEFN
RECORD  DS    0CL350              DEFINE RECORD
        .
        .
        CSECT
        .
        .
        MVC   TCAFCAA,FWACELL1    MOVE BROWSE 1 FWA ADDR TO TCA
        DFHFC TYPE=ESETL          ISSUE ESETL MACRO INSTRUCTION
        MVC   TCAFCAA,FCACELL2    MOVE BROWSE 2 FWA ADDR TO TCA
        DFHFC TYPE=ESETL          ISSUE ESETL MACRO INSTRUCTION
```

```
COBOL:

   02 FWACBAR PIC S9(8) COMP.
         .                          NOTE DEFINE BASE REGISTER FOR FWA.
         .
01 DFHTCADS COPY DFHTCADS.          NOTE COPY SYMBOLIC STRG DEFN FOR TCA.
   02 FWACELL1 PIC S9(8) COMP.
   02 FWACELL2 PIC S9(8) COMP.
01 DFHFWADS COPY DFHFWADS.          NOTE COPY SYMBOLIC STRG DEFN FOR FWA.
   02 RECORD PIC X(350).            NOTE DEFINE RECORD LAYOUT IN FWA.
         .
         .
   MOVE FWACELL1 TO TCAFCAA.        NOTE PREPARE TO END FIRST BROWSE.
       DFHFC TYPE=ESETL             TERMINATE FIRST BROWSE.
         .
         .
   MOVE FWACELL2 TO TCAFCAA.        NOTE PREPARE TO END 2ND BROWSE.
       DFHFC TYPE=ESETL             TERMINATE SECOND BROWSE.
```

```
PL/I:

%INCLUDE DFHTCADS;                  /*COPY SYMBOLIC STRG DEFN FOR TCA*/
   02 FWACELL1 POINTER;
   02 FWACELL2 POINTER;
         .
         .
%INCLUDE DFHFWADS;                  /*COPY SYMBOLIC STRG DEFN FOR FWA*/
   02 RECORD CHAR(350);             /*DEFINE RECORD LAYOUT IN FWA*/
         .
         .
         .
TCAFCAA=FWACELL1;                   /*MOVE BROWSE1 FWA ADDR TO TCA*/
        DFHFC TYPE=ESETL
TCAFCAA=FWACELL2;                   /*MOVE BROWSE2 FWA ADDR TO TCA*/
        DFHFC TYPE=ESETL
```

RESET BROWSE (TYPE=RESETL)

```
  DFHFC TYPE=RESETL
|       [,ARGTYP={KEY|RBA}]¹
        [,SRCHTYP=
          {FKEQ|FKGE|GKEQ|GKGE}]¹
|       [,NORESP=symb-addr]
        [,ERROR=symb-addr]
        [,NOTFND=symb-addr]
        [,INVREQ=symb-addr]
        [,IOERROR=symb-addr]
        [,NOTOPEN=symb-addr]
        [,ILLOGIC=symb-addr]¹

  ¹ VSAM only
```

Once a browse operation has been initiated, the application programmer may, at any time before issuing an ESETL macro for the browse, reset the search argument to some record other than the next sequential record in the data set.

| For a VSAM data set, the type of search argument used in retrieving records can also be reset by issuing the DFHFC TYPE=RESETL macro. Prior to issuing the request, the application programmer should place the address of the appropriate FWA into TCAFCAA and the new record identification in the record identification field specified in the original SETL request.

The use of the RESETL macro allows the application programmer to avoid issuing an ESETL request followed by another SETL request, and causes CICS to use the same I/O and work area. Upon return from the RESETL request, TCAFCAA contains the address of a new FWA that the user can use for the browse operation.

The RESETL request allows the user to "skip" through his data set in a browse operation with ease. A similar capability is available to VSAM users through the GETNEXT macro.

A browse operation should be ended by issuing a TYPE=ESETL macro, but a normal or abnormal end of task will also end a browse.

If browsing in backward mode (GETPREV) and a request is made to reposition the browse (RESETL), VSAM requires that a specific key is used. If the key supplied does not exist, NOTFND will be returned.

The following examples show how to reset the search argument for a browse operation.

```
ASM:

        COPY   DFHTCADS                   COPY TCA SYMBOLIC STRG DEFN
KEYF    DS     D                          DEFINE KEY FIELD IN TWA
FWACBAR EQU    7                          ASSIGN FWA BASE REGISTER
        COPY   DFHFWADS                   COPY FWA DSECT
RECORD1 DS     0CL350                     DEFINE RECORD
        .
        .
        ORG    RECORD1
RECORD2 DS     0CL250                     DEFINE RECORD
        .
        .
        CSECT
        MVC    KEYF(8),=8X'00'            INITIALIZE KEY FIELD
        DFHFC TYPE=SETL,                  ISSUE INITIAL SETL MACRO        *
               DATASET=MASTER,            FOR DATA SET "MASTER"           *
               RDIDADR=KEYF               INIT SEARCH ARG=0
        L      FWACBAR,TCAFCAA            ESTABLISH ADDRESSABILITY TO FWA
        .
        .
        ST     FWACBAR,TCAFCAA            STORE FWA ADDR IN TCA
        MVC    KEYF(8),=CL8'SMITH'        ESTABLISH NEW SEARCH ARGUMENT
        DFHFC TYPE=RESETL                 ISSUE RESETL MACRO
        L      FWACBAR,TCAFCAA            ESTABLISH ADDRESSABILITY TO FWA
```

```
COBOL:

   02 FWACBAR PIC S9(8) COMP.
         .                             NOTE DEFINE BASE REGISTER FOR FWA.
         .
01 DFHTCADS COPY DFHTCADS.             NOTE COPY SYMBOLIC STRG DEFN FOR TCA.
   02 KEYF PIC S9(18) COMP.            NOTE DEFINE KEY FIELD IN TWA.
   02 FILLER REDEFINES KEYF.
      03 KEYC PIC X(8).
         .
         .
01 DFHFWADS COPY DFHFWADS.             NOTE COPY SYMBOLIC STRG DEFN FOR FWA.
   02 RECORD1 PIC X(350).              NOTE DEFINE RECORD.
         .
         .
01 DFHFWA REDEFINES DFHFWADS.          NOTE CREATE STRG DEFN FOR FWA.
   02 FILLER PIC X(16).                NOTE LENGTH OF FWA.
   02 RECORD2 PIC X(250).              NOTE DEFINE RECORD.
         .
         .
   MOVE 0 TO KEYF.
   DFHFC TYPE=SETL,                    ISSUE INITIAL SETL MACRO INSTR      *
         DATASET=MASTER,               FOR DATA SET "MASTER"               *
         RDIDADR=KEYF                  INITIAL SEARCH ARG=0
   MOVE TCAFCAA TO FWACBAR.            NOTE ESTABLISH ADDRESSABILITY TO FWA.
         .
         .
   MOVE FWACBAR TO TCAFCAA.            NOTE STORE FWA ADDRESS IN TCA.
   MOVE 'SMITH' TO KEYC.               NOTE ESTABLISH NEW SEARCH ARGUMENT.
   DFHFC TYPE=RESETL                   ISSUE RESETL MACRO
   MOVE TCAFCAA TO FWACBAR.            NOTE ESTABLISH ADDRESSABILITY TO FWA.
```

```
PL/I:

%INCLUDE DFHTCADS;                     /*COPY SYMBOLIC STRG DEFN FOR TCA*/
   02 KEYF CHAR(8);                    /*DEFINE KEY*/
         .
         .
         .
%INCLUDE DFHFWADS;                     /*COPY SYMBOLIC STRG DEFN FOR FWA*/
   02 RECORD1 CHAR(350);               /*DEFINE RECORD*/
DECLARE 01 DFHXFWA BASED(FWACBAR),
   02 FILL CHAR(16),                   /*LENGTH OF FWA*/
   02 RECORD2 CHAR(250);               /*DEFINE RECORD*/
         .
         .
         .
KEYF=LOW(8);                           /*SET KEY VALUE TO ZERO*/
        DFHFC TYPE=SETL,               ISSUE INITIAL SETL MACRO INSTR      *
              DATASET=MASTER,          FOR DATA SET "MASTER"               *
              RDIDADR=KEYF             INITIAL SEARCH ARG EQUALS ZERO      *
FWACBAR=TCAFCAA;                       /*ESTABLISH ADDRESSABILITY FOR FWA*/
         .
         .
         .
TCAFCAA=FWACBAR;                       /*STORE FWA ADDR IN TCA*/
KEYF='SMITH';                          /*ESTABLISH NEW SEARCH ARGUMENT*/
      DFHFC TYPE=RESETL                ISSUE RESETL
FWACBAR=TCAFCAA;                       /*ESTABLISH ADDRESSABILITY TO FWA*/
```

TEST RESPONSE TO A REQUEST FOR FILE SERVICES (TYPE=CHECK)

```
DFHFC TYPE=CHECK
      [,NORESP=symb-addr]
      [,ERROR=symb-addr]
      [,DSIDER=symb-addr]
      [,NOTFND=symb-addr]
      [,DUPKEY=symb-addr]¹
      [,DUPREC=symb-addr]
      [,INVREQ=symb-addr]
      [,IOERROR=symb-addr]
      [,NOSPACE=symb-addr]
      [,NOTOPEN=symb-addr]
      [,ENDFILE=symb-addr]
      [,ILLOGIC=symb-addr]¹

¹ VSAM only
```

FILE CONTROL RESPONSE CODES

To test a response code the application programmer must know the CICS response codes and their meanings, and the symbolic labels by which he can refer to the response codes. These are shown below.

In an assembler language or PL/I program the response code is in TCAFCTR; in a COBOL program it is in TCAFCRC.

Condition	Response Code		
Name	ASM	COBOL	PL/I
NORESP	X'00'	LOW-VALUES	00000000
ERROR	¬X'00'	¬LOW-VALUES	¬00000000
DSIDER	X'01'	12-1-9	00000001
ILLOGIC	X'02'	12-2-9	00000010
INVREQ	X'08'	12-8-9	00001000
NOTOPEN	X'0C'	12-4-8-9	00001100
ENDFILE	X'0F'	12-7-8-9	00001111
IOERROR	X'80'	12-0-1-8	10000000
NOTFND[1]	X'81'	12-0-1	10000001
DUPREC	X'82'	12-0-2	10000010
NOSPACE[2]	X'83'	12-0-3	10000011
DUPKEY[3]	X'84'	N/A	N/A

[1] NOTFND for GETNEXT, GETPREV, RESETL, or SETL can occur only for VSAM data sets.

[2] NOSPACE can occur only when TYPOPER=NEWREC or TYPOPER=UPDATE is specified.

[3] VSAM and assembler language only.

The multipunch codes in COBOL commonly correspond to unprintable characters. In these cases, the response code can be evaluated by referring to the condition names generated by CICS (for example, FCNORESP), as shown in the examples at the end of this discussion.

The operands that can be used to request tests of the response to a request for file services (that is, a DFHFC macro) are identified in the discussions of the macro formats. The condition expressed by each keyword is explained in detail and should be referred to by the application programmer when using any of the checking methods described above.

When certain conditions (for example, NOTFND, IOERROR, or DUPREC) occur, the FIOA, FWA, or VSWA that has been acquired for the file control request, is retained. Its address is available to the application program. Before other file control requests are issued, the storage occupied by the FIOA or VSWA should be freed by a DFHFC TYPE=RELEASE macro. When the conditions DSIDER, INVREQ, or NOTOPEN occur no storage areas are acquired for the associated file control request.

The following examples show how to examine the response code provided by CICS at TCAFCTR (for assembler language or PL/I) or TCAFCRC (for COBOL) and transfer control to an appropriate user written error handling routine. The alternative approach available to COBOL programmers is also shown.

ASM

```
       DFHFC TYPE=GET,DATASET=MASTER,
             RDIDADR=KEYF
       CLI   TCAFCTR,X'00'
       BE    GOOD
       CLI   TCAFCTR,X'80'
       BE    ERROR
       CLI   TCAFCTR,X'08'
       BE    ERROR
       .
       .
GOOD   DS    0H
       .
       .
ERROR  DS    0H
       DFHPC TYPE=ABEND
```

COBOL

```
       DFHFC TYPE=GET,DATASET=MASTER,
             RDIDADR=KEYF
       IF TCAFCRC=LOW-VALUES GO TO GOOD
       IF TCAFCRC=' ' GO TO ERROR.
       IF TCAFCRCY=' ' GO TO ERROR.
       .
       .
GOOD.
       .
       .
ERROR.
       DFHPC TYPE=ABEND
```

where the value specified within single quotation marks is an unprintable multipunch code for the required hexadecimal value. For example, X'80' has a multipunch code of 12-0-1-8.

The alternative approach to response code checking, that is available to COBOL programmers as described earlier, is generally a coding convenience and provides concise code documentation. When this approach is used, the IF statements above are replaced by statements of the form shown below, using the CICS generated condition names:

```
        IF FCNORESP THEN GO TO GOOD.
        IF FCIOERROR THEN GO TO ERROR.
        IF FCINVREQ THEN GO TO ERROR.
        .
        .
        .
```

PL/I

```
        DFHFC TYPE=GET,DATASET=MASTER,
              RDIDADR=KEYF
        IF TCAFCTR='00000000'B
        THEN GO TO GOOD;
        IF TCAFCTR='10000000'B
        THEN GO TO ERROR;
        IF TCAFCTR='00001000'B
        THEN GO TO ERROR;
        .
        .
GOOD:
        .
        .
ERROR:
        DFHPC TYPE=ABEND
```

OPERANDS OF DFHFC MACRO

ARGTYP= **(VSAM only)**
describes the contents of the record identification field.

KEY
indicates that the record identification field contains a search key or a relative record number.

RBA
indicates that the record identification field contains a relative byte address.

In a DFHFC TYPE=GETAREA macro, this operand can only be used when TYPOPER=MASSINSERT is also specified. It describes the record identification fields of the records to be mass inserted by DFHFC TYPE=PUT macros. When used in a mass insert operation, this operand cannot be overridden.

DATASET=symb-name
specifies the name of the file to be accessed. The name must appear in the file control table (FCT).

If this operand is omitted, the name is assumed to be in TCAFCDI.

This name corresponds to the file name in the DLBL job control statement, which identifies the data set.

DSIDER=symb-addr
specifies the entry label of the user-written routine to which control is to be passed if the file name specified by the DATASET operand (or at TCAFCDI) cannot be located in the FCT. The contents of TCAFCAA are not meaningful.

DUPKEY=symb-addr **(VSAM only)**
specifies the entry label of the user-written routine to which control is to be passed if the duplicate key condition is raised. This condition indicates that a record has been retrieved but that there are other records in the data set that have the same key. These records can be retrieved by a browse.

This condition can occur when a record is retrieved via an alternate index with the NONUNIQEKEY attribute, and another alternate index record with the same key follows. It does not occur as a result of a TYPE=GETNEXT macro that reads the last of the records having the nonunique key.

DUPREC=symb-addr
specifies the entry label of the user-written routine to which control is to be passed if an attempt is made to add a record either to a data set, or to an alternate index with the UNIQUEKEY attribute, in which the same key already exists.

TCAFCAA contains the address of a VSWA if the PUT is for a VSAM data set.

The FWA will be released by the file control program. After interrogation of the FIOA or VSWA returned is complete, the program should issue a DFHFC TYPE=RELEASE macro.

ENDFILE=symb-addr
specifies the entry label of the user-written routine to which control is to be passed if an end-of-file condition is detected during the sequential retrieval (browse) of records in a data set. This condition can occur in response to a GETNEXT, GETPREV, or CHECK request. TCAFCAA contains the address of the FWA for the browse operation if move mode is specified or implied in the SETL request. TCAFCAA contains the

address of the VSWA that represents the browse if locate mode is specified.

ERROR=symb-addr
specifies the entry label of the user-written routine to which control is to be passed if any error occurs on a file operation. The CICS response code should be further interrogated in this user-written routine.

ILLOGIC=symb-addr (VSAM only)
specifies the entry label of the user-written routine to which control is to be transferred if an error that does not fall within one of the other CICS response categories occurs. TCAFCAA contains the address of a VSWA. The user's routine may check the logical error codes in the RPL that is at VSWARPL. The error code is at VSWAERRC, and the return code is at VSWARTNC.

After interrogation of the area returned, the program should issue a DFHFC TYPE=RELEASE macro.

INITIMG=
specifies a 1-byte (2-digit) hexadecimal initialization value for the FWA.

value
is a 2-digit hexadecimal numeral to be used as the initialization value.

YES
indicates that the hexadecimal initialization value has been placed in TCASCIB.

If this operand is omitted, the FWA is initialized to EBCDIC blanks (X'40').

INVREQ=symb-addr
specifies the entry label of the user-written routine to which control is to be passed if the attempted file operation is not provided for or allowed according to the file entry specification in the FCT, or if the FWA is corrupted.

TCAFCAA contains X'08' in byte 0, and the following in bytes 1-3:

- A nonzero value if the request is not allowed according to the FCT entry for the file.
- Zero if the request is invalid or if the code to support the request has not been generated into the FCT.

IOERROR=symb-addr
specifies the entry label of the user-written routine to which control is to be passed if an input/output error occurs during a file operation. When an I/O event error code is not covered by one of the CICS error classes (for example, by NOSPACE or NOTFND), it is considered to be an I/O error.

TCAFCAA contains:

- The address of an FIOA if the request is against a DAM data set
- The address of a VSWA if the request is against a VSAM data set.

The application programmer should be aware of the following considerations:

- For a DAM data set, a user routine may check for error codes in the FIOA: in field FCFIOBEX for CICS/OS/VS or in field FCIOERR for CICS/DOS/VS. Because of access method and operating system dependencies, checks for these codes may have a limiting effect on the usability of an application program in varying environments, particularly if migration from CICS/DOS/VS to CICS/OS/VS becomes desirable.
- For a VSAM data set, the error codes may be checked in the request parameter list (RPL) located at VSWARPL. The error code is at VSWAERRC, and the return code is at VSWARTNC. Because of access method and operating system dependencies, checks for these codes may have a limiting effect on the usability of an application program in varying environments, particularly if migration from CICS/DOS/VS to CICS/OS/VS becomes desirable.
- For RESETL or GETNEXT the browse operation is still active, but the position in the data set may have been lost. A RESETL using the record identification for the next record required should be issued to reestablish the position in the data set. If move mode is specified or implied in the initiating SETL request, the FWA representing the browse operation must be used for the RESETL; if locate mode is specified in the SETL request, the VSWA must be used.

- For PUT, the FWA will have been released.

  Except for RESETL or GETNEXT, after any interrogation of the area returned is complete, the program should issue a DFHFC TYPE=RELEASE macro.

MODE= (VSAM only)
is used to specify the processing mode for a read-only or browse request.

MOVE
specifies move-mode processing. Upon return to the application program, TCAFCAA contains the address of the FWA acquired for the read-only or browse operation. If the data set referred to contains variable-length records, the LLbb length field is included as part of the record.

LOCATE
specifies locate-mode processing. Upon return to the application program, TCAFCAA contains the address of a VSWA. The address of the retrieved record is at VSWAREA. If the data set referred to contains variable-length records, the LLbb length field is not retrieved as part of the record; instead, the length of the record is placed in VSWALEN. This parameter cannot be specified if TYPOPER=UPDATE is specified.

NORESP=symb-addr
specifies the entry label of the user-written routine to which control is to be passed if no error occurs on a file operation, that is, a normal response.

The field TCAFCAA in the TCA of the task contains:

- The address of an FIOA after a read only GET against an unblocked non-VSAM data set or a blocked DAM data set if deblocking is not requested
- The address of an FWA after a GET against a blocked data set, a GET for update, a GETAREA, SETL, GETNEXT, or RESETL. An FWA is always acquired for VSAM move-mode operations, regardless of blocking
- The address of a VSWA after a locate-mode GET or SETL against a VSAM data set and after a GETNEXT or RESETL for a browse operation initiated by a locate-mode SETL
- Meaningless information after a PUT, DELETE, RELEASE, or ESETL.

NOSPACE=symb-addr
specifies the entry label of the user-written routine to which control is to be transferred if no direct access space is available for adding records to a data set. (This error condition is not applicable when adding records to a fixed length DAM data set that does not contain keys.) TCAFCAA contains the address of an FWA containing the record to be added. This FWA may be at a different storage location from the FWA passed with the PUT request.

NOTFND=symb-addr
specifies the entry label of the user-written routine to which control is to be passed if an attempt to retrieve or delete a record based on the search argument provided is unsuccessful.

TCAFCAA contains:

- The address of an FIOA if the request was a GET against a DAM data set
- The address of a VSWA if the request was a GET, DELETE, SETL, RESETL, or GETNEXT request using skip-sequential against a VSAM data set.

The application programmer should be aware of the following considerations:

- Except for RESETL or GETNEXT, the program should issue a DFHFC TYPE=RELEASE macro when any interrogation of the area is complete.
- For SETL, the browse operation was not initiated.
- For RESETL or GETNEXT, the browse operation is still active, but the position in the data set has been lost. A RESETL should be issued to reestablish the position in the data set. If move mode is specified or implied in the SETL request initiating the browse operation, the FWA representing the browse must be used for the RESETL; if locate mode is specified in the SETL request, the VSWA must be used.

NOTOPEN=symb-addr
specifies the entry label of the user-written routine to which control is to be transferred if the

requested file is closed and unenabled. This state is reached after an open enabled file has been closed.

This condition does not occur if the request is made to a closed enabled file; instead the file is opened as part of executing the request. A request made to a closed disabled file causes the application to terminate abnormally.

This condition can occur in response to a GET,PUT NEWREC that is not part of a MASSINSERT,DELETE or the first of a GENERIC DELETE sequence, GETAREA or SETL request. When the condition is returned, the contents of TCAFCAA are not meaningful.

RDIDADR=
specifies the address of the record identification field for a record, or the relative record number of a record. The contents of this field should not be altered when a GET or UPDATE macro is issued.

symb-addr
is the address of the record identification field that contains the block reference (for DAM), or the key, relative byte address, or the relative record number (for VSAM) of the record to be processed. If this operand is omitted, the address is assumed to be in TCAFCRI. This field is used when adding a new record or when updating an existing record in a nonkeyed DAM data set without previously reading it for update.

Notes:

1. This operand must not refer to a field in the FWA, because the FWA might be freed before the write occurs.

2. The DFHFC TYPE=PUT,TYPOPER=NEWREC macros for a VSAM mass-insert operation may specify the same record identification field or different record identification fields.

3. When adding a record to a VSAM ESDS, there is no need to supply an RBA. However, a field must be provided to receive the RBA after the record has been added; the address of the field must be supplied either in TCAFCRI or by using the RDIDADR operand.

relative record number
is the number of the required record in a VSAM RRDS. The format of this field must be fullword binary. If this operand is omitted, the address of the field that contains the record number is assumed to be in TCAFCRI.

Notes:

1. The SRCHTYP operand is assumed to be FKEQ on all DFHFC macros except SETL and RSETL when only FKEQ and FKGE will be accepted.

2. In skip sequential operation, if the relative record number refers to a nonexistent or deleted record, the NOTFND condition will be raised, even if SETL or RESETL included SRCHTYP=FKGE.

RETMETH= (DAM only)
applies only to blocked data sets and is used to specify the argument type (retrieval method) for deblocking the data sets. It is also used to specify the format of the information placed in the record identification field each time a record is retrieved in a browse operation.

RELREC
specifies that retrieval is to occur by relative record, with the first record in a block considered to be record zero. It also specifies that a one-byte binary relative record number is provided in a browse operation.

KEY
specifies that retrieval is to occur by key or that in a browse operation, a key is to be provided.

If TYPOPER=UPDATE is specified, the RETMETH operand is required. If RETMETH is omitted and a request to read a blocked DAM data set is issued, the entire physical record (block) is returned in the FIOA to the application program. The block reference field, required by DAM, contains the criteria for deblocking the data set. If a retrieved record is "undefined," the application program must determine the length of the record.

SRCHTYP= (VSAM only)
specifies how the search key in the record identification field is to

be used. This operand is meaningful only when ARGTYP=KEY is specified or implied by default.

FKEQ
indicates that the search key is a full key and that only a record with an equal key satisfies the search.

FKGE
indicates that the search key is a full key and that the first record with a key equal to or greater than the search key satisfies the search.

GKEQ
indicates that the search key is a generic (partial) key, the binary length of which is specified in the first byte of the record identification field. A record whose key is equal to the search key (compared on only the number of bytes specified in the first byte of the record identification field) satisfies the search. When used with TYPE=DELETE, all records whose keys begin with the search key are to be deleted. A count of the number of records deleted is returned in TCAFCNRD.

GKGE
indicates that the search key is a generic key and that the first record with a key equal to or greater than the search key (compared on only the number of bytes specified in the first byte of the record identification field) satisfies the search.

TYPOPER=
describes the file operation to be performed.

NEWREC
indicates that a new record is to be added to an existing data set.

UPDATE
when used with a TYPE=PUT macro, it indicates that a record retrieved previously by a DFHFC TYPE=GET,TYPOPER= UPDATE macro is to be updated (in effect, rewritten to the data set).

When used with a TYPE=GET macro, it indicates that a record is to be obtained for updating, or, if a VSAM KSDS is referred to, for either updating or deletion. If the record is from a blocked DAM data set, the RETMETH operand must be specified. If TYPOPER=UPDATE is omitted, a read-only operation is assumed.

The UPDATE parameter can also be used with TYPE=PUT to write a record to a DAM data set after building the record in an area obtained by a DFHFC TYPE=GETAREA macro. This technique is described in detail in "DAM Data Sets" on page 54.

DELETE (VSAM only)
is valid when a KSDS data set is being accessed and indicates that a record previously retrieved for update by a DFHFC TYPE=GET, TYPOPER=UPDATE request is to be deleted from the data set.

MASSINSERT (VSAM only)
specifies that the acquired FWA is to be used for a mass-insert operation. This ensures that the same FWA is used for subsequent DFHFC TYPE=PUT macros adding new logical records with keys or relative byte addresses in ascending sequence to the data set. The FWA is made available to the application program after each DFHFC TYPE=PUT macro. The FWA is reinitialized, before each return to the application program, to the value specified in the INITIMG operand (if specified) or otherwise to EBCDIC blanks (X'40'). A mass-insert operation is terminated by a DFHFC TYPE=RELEASE macro.

CHAPTER 3.3. DL/I SERVICES

The CICS application programmer can request DL/I services in two ways:

1. By issuing a DL/I CALL statement, written according to DL/I specifications. This method is available to both CICS/OS/VS and CICS/DOS/VS users. This DL/I CALL is mandatory if the user wishes to access remote DL/I data bases using ISC (intersystem communication).
2. By issuing a DFHFC macro. This method is available to CICS/OS/VS users only.

In both cases, control is passed to a routine that acts as an interface between the CICS application program and the DL/I request handler. This routine performs validity checks on the CALL list, prepares DL/I to handle the request, and passes control and the CALL list to DL/I. After DL/I has handled the request, it returns control to the calling program unless a DL/I pseudoabend has occurred, in which case the CICS task is abnormally terminated.

Under CICS, two or more transactions (tasks) may require the same application program at any given time during system operation. Because CICS application programs must be quasi-reenterable (see "Quasi-Reenterability," in "Chapter 1.1. Macro-Level Application Programming" on page 3), DL/I areas that may be modified under CICS/OS/VS (such as PCB pointers, I/O work areas, and segment search arguments) should not be placed in static storage. They should also not be placed in working storage (unless the application program contains one or more command level statements, in which case working storage is dynamic). Storage for such areas must be obtained from CICS dynamic storage by each transaction using the program.

Four steps must be performed by an application program requesting DL/I data base services. These steps are listed below and explained in the sections that follow.

1. Obtain addresses of program communication blocks (PCBs) to be used by the application program.
2. Build segment search arguments (SSAs) if they are to be used in the CALL.
3. Acquire I/O work areas for DL/I segments processed by the program.
4. Issue the DL/I CALL.

OBTAINING ADDRESSES OF PROGRAM COMMUNICATION BLOCKS

An application program that uses the CICS-DL/I interface accesses data bases by means of program communication blocks (PCBs). One PCB for each data base is included in the program specification block (PSB) for the program. To process DL/I CALLs within a CICS transaction, the PSB for the transaction must be scheduled and the PCB addresses obtained before any DL/I CALLs are made. Scheduling involves, for example, that all the required DL/I control blocks exist and are in main storage, and that the processing options associated with this PSB permit it to be scheduled concurrently with those PSBs already scheduled. If they are not obtained, an INVREQ (invalid request) indicator is returned in response to any DL/I CALL within the program.

A transaction may schedule only one PSB at a time. An attempt to schedule a second PSB while one is still scheduled causes the INVREQ condition to be returned.

A sync point request (see "Chapter 7.6. Recovery/Restart (Sync Point) Control (DFHSP Macro)" on page 319) by a task that is scheduled to use DL/I resources implies the release of those resources. This means that if, after issuing a DFHSP TYPE=USER macro, access to a DL/I data base is required, the desired PSB must be rescheduled. The previous position of that data base has been lost.

I/O PCBs (a type of control block used by IMS/VS) are not passed to CICS programs, even though they may be included in a PSB for a transaction.

DFHFC MACRO (CICS/OS/VS ONLY)

To schedule the desired PSB and obtain PCB addresses, the CICS/OS/VS application programmer may use a special form of the DFHFC macro, as follows:

```
DFHFC TYPE=(DL/I,PCB)
      [,PSB=
        {'psbname'|symb-addr|YES}]
      [,NORESP=symb-addr]
      [,DLINA=symb-addr]
      [,PSBSCH=symb-addr]
      [,PSBNF=symb-addr]
      [,PSBFAIL=symb-addr]
      [,INVREQ=symb-addr]
```

where:

TYPE=(DL/I,PCB)
indicates that PCB addresses are to be acquired. (DL/I may also be coded as DLI or DL1.)

If the PSB has been located, TCADLPCB contains the address of a list of PCB addresses in the sequence in which the PCB addresses were specified during the PSBGEN of this PSB. If the PSB cannot be found, TCADLPCB contains zero. If the PSB pool or DMB pool is too small to hold the requested blocks even when no other PSBs or DMBs are in their pools, the transaction is terminated with the ADLA abend code.

DL/I CALL STATEMENT (CICS/DOS/VS OR CICS/OS/VS)

Upon receiving control from CICS, a CICS application program must issue a DL/I call to perform scheduling before attempting to access DL/I data bases. If the scheduling call is successful, the address of the PCB list is returned in the field TCADLPCB and TCAFCTR is set to zeros. If the call is unsuccessful, TCAFCTR contains a one-byte return code. Before continuing with subsequent DL/I calls, it is the application programmer's responsibility to test these indicators to determine whether scheduling is successful.

The format of the CALL statement to request scheduling is as follows:

ASM:

```
CALLDLI {ASMTDLI|CBLTDLI},([parmcount,]
         function,[psb])
```

COBOL:

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

PL/I:

```
CALL PLITDLI ([parmcount,]
     function,[psb]);
```

where:

parmcount
is the name of a binary fullword containing the parameter count (value of one or two). This parameter is optional.

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

psb
is the name of the eight-byte field containing the PSB generation name which the application program accesses. (This name is one to eight bytes long under CICS/OS/VS, or one to seven bytes long under CICS/DOS/VS, and is left justified and padded on the right with blanks as appropriate.) This parameter is optional. Under CICS/DOS/VS if it is omitted, the PSB name is assumed to be the first PSB name associated with the application program name in the DL/I application control table generation. Under CICS/OS/VS if it is omitted, the PSB name is assumed to be the name of the application program associated with the task in the PCT.

Note: With PL/I, the preprocessor and assembler steps cause a single CALL PLITDLI statement to be expanded into a series of PL/I statements. If a single CALL PLITDLI statement appears in a THEN, ELSE, or WHEN clause, it should be coded within

```
  DO
   .
   .
  END
```

statements.

If a single CALL PLITDLI statement is to be coded in an ON-unit or is to be the scope of a conditional prefix, the CALL PLITDLI should be coded within

```
  BEGIN
   .
   .
  END
```

statements.

BUILDING SEGMENT SEARCH ARGUMENTS

Both CICS/OS/VS and CICS/DOS/VS application programmers can use segment search arguments (SSAs) in a DL/I CALL to identify a specific segment, or, if qualified, to identify the range of values within which a segment exists. In addition, the CICS/OS/VS programmer can specify SSAs in a DFHFC TYPE=DL/I macro. If used, SSAs must be built by the application programmer before a DL/I CALL is issued. (For information on how to build an SSA, CICS/OS/VS application programmers should refer to the IMS/VS

Application Programming for CICS/VS Users; CICS/DOS/VS users should refer to the DL/I DOS/VS Application Programming Reference Manual. Note that for CICS/OS/VS users all IMS/VS DB command codes are supported, including the "Q" code (although corresponding dequeueing must be performed by a CICS sync point, or by a DL/I TERM call, since the DEQ call is not supported).)

In a DL/I application program, SSAs are built in fixed storage within the program. In a CICS application program, SSAs must be built in dynamic storage to maintain the quasi-reenterability of the program.

The storage acquired to build the SSAs is addressed as follows:

- For assembler language programs, the address should be placed in the register that establishes addressability for the SSA dynamic storage.
- For COBOL programs, the address is moved to the BLL pointer for this storage. The BLL pointer is defined under the COPY DFHBLLDS statement in the linkage section and must be in the same relative position in the BLL list as the 01 statement for the SSA dynamic storage is among the 01 statements in the linkage section.
- For PL/I, the address is stored in the variable upon which the SSA dynamic storage is based.

After the storage has been acquired and the SSAs built, DL/I CALLs in which the SSAs are used can be issued by the application program. The names of the SSAs to be used, if any, are specified in the parameter list of the CALL. Under CICS/OS/VS, a DFHFC TYPE=DL/I macro can also be used. In a DFHFC TYPE=DL/I macro, the application programmer can specify the number and names of the SSAs in different ways:

1. SSAS=NO indicates that there are no SSAs in this CALL.
2. SSAS=(ssacount,ssa1,ssa2,...), where ssacount is optional, represents either the fixed-point number of SSAs in the CALL or the symbolic address of the fullword that contains the number of SSAs. Specifying a field to contain the number of SSAs provides the application programmer with flexibility in writing one DFHFC statement to be used in many different CALLs. ssa1, ssa2,..., are the symbolic names of the SSAs.
3. SSALIST=YES indicates that the application programmer has built a list of fullwords, optionally containing the number of SSAs (which may be zero) in the first word, and the addresses of the SSAs in the following words, and that he has stored the address of this list at TCADLSSA.
4. SSALIST=symbolic address indicates that the address of an SSA list built by the application programmer as indicated in item 3 is at the location specified.

In assembler language programs, ssacount,ssa1,ssa2,..., can be contained in registers if the specifications are enclosed in parentheses.

ACQUIRING AN I/O WORK AREA

When issuing a request for DL/I services, the address of a work area, either that in which a current segment is contained or that in which DL/I is to place the segment in a retrieval CALL, is required. This area must be specified by the CICS/OS/VS or CICS/DOS/VS programmer who issues a DL/I CALL. It may be provided by the interface, if the programmer desires, for a retrieval-type DFHFC macro.

If the CICS/OS/VS application programmer knows the address of the work area to be used in the DFHFC macro, including the case for which storage is acquired for a retrieval-type (Gxxx) request, he specifies the name of the pointer to that storage in the WRKAREA=name operand, or he places the address of the storage in TCADLIO before issuing the request and specifies WRKAREA=YES.

If the application programmer wishes to allow the interface to obtain the work area for a retrieval-type request, he does not include the WRKAREA operand in the DFHFC macro request. If the request was serviced successfully, the address of an acquired I/O work area is in TCADLIO. The address at TCADLIO is the address of the storage accounting area (SAA) preceding the retrieved data. The area becomes the responsibility of the programmer and is not freed until he frees it or until the transaction terminates. If the application programmer elects to free the work area, he must use a DFHSC TYPE=FREEMAIN macro.

Note: The address of the I/O area is specified as the address of the storage accounting area preceding the data when a DFHFC macro is used; the address of the first byte of the data area is required when a DL/I CALL is used.

REQUESTING DL/I SERVICES

The application program request for DL/I services may be either a CICS DFHFC macro (CICS/OS/VS) or a DL/I call (CICS/OS/VS or CICS/DOS/VS).

DFHFC MACRO (CICS/OS/VS)

```
DFHFC TYPE=(DL/I [,function])
      [,PCB={symb-addr|(register)}]
      [,WRKAREA={symb-addr|
        YES|(register)}]
      [,SSAS={NO|([ssacount][,ssa1]
      [,ssa2,...])|([(register1)]
        [,(register2),...])}]
      [,SSALIST={YES|NO
      |symb-addr|(register)}]
      [,NORESP=symb-addr]
      [,PSBFAIL=symb-addr]
      [,DLINA=symb-addr]
      [,FUNCNS=symb-addr]
      [,INVREQ=symb-addr]
```

where:

TYPE=(DL/I [,function])
: specifies the two- to four-byte name of the function to be performed. If the function is not specified, it is assumed to be in TCADLFUN. (DL/I may also be coded as DLI or DL1.)

DL/I CALL STATEMENT (CICS/OS/VS OR CICS/DOS/VS)

DL/I data base services are available to CICS application programs through CALL statements. The CALL statement formats for COBOL and PL/I are similar. For assembler language application programs, a CALLDLI macro is used. The formats of the DL/I calls are as follows:

ASM:

```
CALLDLI {ASMTDLI|CBLTDLI}
          [,([parmcount,]function,pcb
           ,workarea[,ssa,...])]
```

COBOL:

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

PL/I:

```
CALL PLITDLI (parmcount,function
     ,pcb,workarea[,ssa,...]);
```

where:

parmcount
: is the name of a binary fullword containing the parameter count or argument count of the arguments which follow; this is optional for assembler language and COBOL.

function
: is the name of the field containing the four-character DL/I input/output CALL function desired.

pcb
: is the program communication block (PCB) name (or DSECT name if assembler).

workarea
: is the name of the I/O work area.

ssa1 to ssan
: are the names of the segment search arguments (SSAs); these are optional.

Notes:

1. If no parameters are specified in an assembler language CALLDLI macro, R1 is assumed to contain the address of a parameter list.

2. In assembler language, an alternative format may be used:

   ```
   CALLDLI {ASMTDLI|CBLTDLI}
   ,MF=(E,(register) or address)
   ```

   where:

   address
   : is the address of the parameter list, or register that contains the address of the parameter list.

RELEASING A PSB IN THE CICS APPLICATION PROGRAM

To reduce pool and intent contention, the CICS/OS/VS application program can release the PSB after a DL/I service has been requested.

It is recommended that conversational programs release the PSB before writing to a terminal so that other transactions can use the PSB while the conversational program is waiting for an operator response.

To ensure the integrity of the data base, a request to release a PSB implies the end of a logical unit of work for the entire task. This means that a DFHSP TYPE=USER macro is issued on behalf of a task that is releasing a PSB.

DFHFC MACRO (CICS/OS/VS ONLY)

To release a PSB for use by other transactions, the CICS/OS/VS application programmer may issue a macro of the following format:

```
DFHFC TYPE=(DL/I,{TERM|T})
      [,DLINA=symb-addr]
      [,TERMNS=symb-addr]
      [,INVREQ=symb-addr]
```

where:

TYPE=(DL/I,TERM)
specifies that the PSB is to be released for use by other transactions, or, if not required, its pool space and associated DMB pool space may be released for other purposes. (DL/I may also be coded as DLI or DL1.)

Before issuing any other DL/I CALLs or DFHFC macros requesting DL/I access to a data base, the application programmer must again issue a schedule request. All positioning in data bases referred to by the transaction is lost when the PSB is released. If the program is processing a hierarchy through GNxx requests before releasing the PSB, it is necessary to explicitly reposition the data bases after issuing another schedule request, to continue the GNxx requests.

DL/I CALL STATEMENT (CICS/DOS/VS OR CICS/OS/VS)

If the CICS application program desires to relinquish control of the PSB, it must issue a terminal call to DL/I. The format of the CALL statement to request termination is as follows:

ASM:

```
CALLDLI {ASMTDLI|CBLTDLI}
,([parmcount,]function)
```

COBOL:

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

PL/I:

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

where:

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

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

When a termination call is issued for a previously scheduled PSB, the resources acquired for the task are released, and tasks awaiting the resources are given an opportunity to be scheduled.

DL/I SERVICES RESPONSE CODES

To test a response code, the application programmer must know the CICS response codes and their meanings. If the assembler language or PL/I programmer uses this approach, he can access the response codes for NORESP, INVREQ, and NOTOPEN at TCAFCTR; the response codes for all other conditions can be accessed at TCADLTR. The COBOL programmer can access the response codes for NORESP, INVREQ, and NOTOPEN at TCAFCRC; the response codes for all other conditions can be accessed at TCADLTR. Response codes and their associated conditions are shown in Figure 12 on page 92.

TEST RESPONSE TO A DL/I REQUEST (TYPE=CHECK)

```
DFHFC TYPE=CHECK
      [,NORESP=symb-addr]
      [,DLINA=symb-addr]
      [,PSBSCH=symb-addr]
      [,PSBNF=symb-addr]
      [,PSBFAIL=symb-addr]
      [,FUNCNS=symb-addr]
      [,TERMNS=symb-addr]
      [,LANGCON=symb-addr][1]
      [,TASKNA=symb-addr][1]
      [,PSBNA=symb-addr][1]
      [,INVREQ=symb-addr]

[1] CICS/DOS/VS only
```

where:

TYPE=CHECK
indicates that the CICS-DL/I interface response is to be checked.

The application programmer may use the DFHFC TYPE=CHECK macro following a DL/I CALL statement or a DFHFC TYPE=(DL/I[,function]) macro. This macro does not check the DL/I return codes in the PCB. If DL/I issues a pseudoabend during processing of the request, control is not returned to the application program. The transaction is terminated with CICS abend code ADLA. For CICS/DOS/VS, if DL/I issues a pseudoabend during a call, the transaction is terminated with a Dnnn abend code where nnn is the DL/I pseudoabend code.

If the application programmer does not provide for the checking of a particular response, and if the exception condition corresponding to that response occurs, program flow proceeds to the instruction following the DL/I request in the application program.

DL/I Interface Request	Condition	Response Code		
		ASM	COBOL	PL/I
(DL/I,PCB),(DL/I [,function]),CHECK	NORESP (Normal Response)	X'00'	LOW-VALUES (FCNORESP)	00000000
All	INVREQ (Invalid Request)	X'08'	12-8-9 (FCINVREQ)	00001000
(DL/I[function]), CHECK	NOTOPEN (Not Open)	X'0C'	12-4-8-9 (FCNOTOPEN)	00001100
Codes returned in TCADLTR after NOTOPEN condition				
(DL/I[function]), CHECK	Data base not open; request issued in VSE system	X'01'	12-1-9	00000001
	Intent scheduling conflict	X'02'	12-2-9	00000010
Codes returned in TCADLTR after INVREQ condition				
ALL	Data base not in FCT, or not open according to FCT, or invalid argument passed to DL/I	X'00'	LOW-VALUES	00000000
(DL/I,PCB),CHECK	PSBNF (PSB Not Found)	X'01'	12-1-9 (DLPSBNF)	00000001
CHECK	TASKNA (Task Not Authorized)	X'02'	12-2-9 (DLTASKNA)	00000010
(DL/I,PCB),CHECK	PSBSCH (PSB Already Scheduled)	X'03'	12-3-9 (DLPSBSCH)	00000011
CHECK	LANGCON (Language Conflict)	X'04'	12-4-9 (DLLANGCON)	00000100
(DL/I,PCB),CHECK	PSBFAIL (PSB Initialization Failed)	X'05'	12-5-9 (DLPSBFAIL)	00000101
CHECK	PSBNA (PSB Not Authorized)	X'06'	12-6-9 (DLPSBNA)	00000110
(DL/I,T),CHECK	TERMNS (Termination Not Scheduled)	X'07'	12-7-9 (DLTERMNS)	00000111
(DL/I[,function]), CHECK	FUNCNS (Function Not Scheduled)	X'08'	12-8-9 (DLFUNCNS)	00001000
All	DLINA (DL/I Not Active)	X'FF'	HIGH-VALUES (DLINA)	11111111

Notes:

1. The TASKNA and LANGCON conditions apply only to CICS/DOS/VS.
2. PSBNA occurs only when the data base is on a VSE system.
3. The names enclosed in parentheses in the COBOL column indicate the names generated by CICS. These names may be used in testing for the respective conditions in a COBOL program.
4. For CICS/OS/VS only, NOTOPEN will never be returned to the application. If a schedule request is made against a closed data base, PSBFAIL will be returned. A data base cannot be closed until all activity against it has been quiesced. While this is happening, no further scheduling is allowed.

Figure 12. CICS-DL/I Interface Response Codes

DL/I REQUESTS IN AN ASSEMBLER LANGUAGE PROGRAM (CICS/OS/VS)

The application programmer must first get the addresses of the PCB. When CICS/OS/VS returns from servicing the PCB request, if the programmer loads R1 from TCADLPCB, his program is in the same state as after an ENTRY DLITCBL statement has been executed in an IMS/VS DL/I application program.

The examples that follow show the options available to the application programmer in a few of the acceptable combinations. The application program must be quasi-reenterable. If a DFHFC macro is issued, the PCB and WRKAREA operands are used to specify the addresses of pointers to fields rather than the addresses of fields desired.

```
        COPY DFHTCADS                         COPY TCA DEFINITION - INCLUDES
*                                             DL/I FIELDS
PSBNAME DC    CL8'PSBNAME'                    NAME OF PSB TO BE USED
PCBFUN  DC    CL4'PCBb'                       PCB FUNCTION
PCBPTRS DSECT                                 PCB POINTERS RETURNED BY
*                                             INTERFACE
PCB1PTR DS    F                               STORAGE FOR PCB POINTERS
PCB2PTR DS    F
        .
        .
WORKAPTR DS   F                               STORAGE FOR POINTER IN I/O WORK
*                                             AREA
PCB1    DSECT                                 PCB DSECT
        .
        .
        .
PCB2    DSECT                                 PCB DSECT
        .
        .
        .
WRKAREA DSECT                                 DL/I WORK AREA DSECT
        DS    2F                              STORAGE PREFIX
WORKA1  DS    CL40                            WORK AREA
SSAREA  DSECT                                 SSA DSECT
        DS    2F                              STORAGE PREFIX
SSA1    DS    CL40                            SSA1 LAYOUT
SSA2    DS    CL20                            SSA2 LAYOUT
        .
        .
        .
       DFHFC TYPE=(DL/I,PCB)                  USE PSB FOR THIS PROGRAM
       DFHFC TYPE=(DL/I,PCB),                 GET PCB'S IN 'PSB14'           *
             PSB='PSB14'
       DFHFC TYPE=(DL/I,PCB),                 GET PCB'S IN SPECIFIED PSB     *
             PSB=PSBNAME
       MVC TCADLPSB,=CL8'PSBA'                PUT PSB NAME IN TCA
       DFHFC TYPE=(DL/I,PCB),                 GET PCB'S OF PSB NAMED IN TCA  *
             PSB=YES
       L     R1,TCADLPCB                      GET ADDRESS OF PCB ADDR LIST
       USING PCBPTRS,R1                       REG 1 IS BASE OF PCB POINTERS --
*                                             USER MUST PROVIDE ADDRESSABILITY
*                                             TO PCB'S WHEN USING THEM
* ISSUE A PCB REQUEST VIA CALLDLI
       CALLDLI CBLTDLI,(PCBFUN)               USE PSB FOR THIS PROGRAM
       CALLDLI CBLTDLI,(PCBFUN,PSBNAME)GET PCB'S IN SPECIFIED PSB
       L     R1,TCADLPCB                      GET ADDRESS OF PCB ADDRESS LIST
* ACQUIRE STORAGE FOR WORK AREA
       DFHSC TYPE=GETMAIN,...                 GET STORAGE FOR WORK AREA
       L     R2,TCASCSA                       REG 2 IS BASE FOR WORK AREA
       USING WRKAREA,R2                       TELL ASSEMBLER
* ACQUIRE STORAGE FOR SSA'S
       DFHSC TYPE=GETMAIN,...                 GET STORAGE FOR SSA'S
       L     R3,TCASCSA                       REG 3 IS BASE FOR SSA'S
       USING SSAREA,R3                        INDICATE TO ASSEMBLER
*
       CALLDLI CBLTDLI,(function,PCB1,WORKA1,SSA1,SSA2)
*
```

```
* CALL DL/I VIA DFHFC MACRO -- VARIOUS EXAMPLES
*
* EXAMPLE 1
*
       DFHFC TYPE=(DL/I,function),                                     *
             PCB=PCB1PTR,              PCB IS POINTED TO               *
             WRKAREA=WORKAPTR,         WORK AREA IS POINTED TO         *
             SSAS=(2,SSA1,SSA2),       SSA COUNT AND SSAS SPECIFIED    *
             NORESP=GOOD1              NORMAL RESPONSE BRANCH
*
* EXAMPLE 2
*
       MVC   TCADLPCB,PCB1PTR          PRELOAD PCB POINTER
       LA    R0,WRKAREA                PICK UP WORK AREA ADDRESS
       ST    R0,TCADLIO                STORE IN TCA
       DFHFC TYPE=(DL/I,DLET),         FUNCTION SPECIFIED              *
             WRKAREA=YES,              WORK AREA ADDRESS PRELOADED     *
             SSAS=NO                   NO SSAS
*
* EXAMPLE 3
*
       MVC   TCADLFUN,=CL4'GU'         PRELOAD FUNCTION
       DFHSC TYPE=GETMAIN,...          GET STORAGE FOR SSA LIST
       L     R4,TCASCSA                PICK UP STORAGE ADDRESS
       LA    R4,8(R4)                  BYPASS PREFIX
       LA    R0,1                      GET COUNT OF SSA'S
       ST    R0,0(R4)                  STORE IN SSA LIST
       LA    R0,SSA1                   GET ADDRESS OF 'SSA1'
       ST    R0,4(R4)                  STORE IN SSA LIST
       ST    R4,TCADLSSA               STORE LIST ADDRESS IN TCA
       OI    4(R4),X'80'               SET ON THE END-OF-LIST BIT
       DFHFC TYPE=DL/I,                DL/I CALL, FUNCTION PRELOADED   *
             PCB=PCB1PTR,              POINTER TO PCB TO BE USED       *
                                       INTERFACE WILL PROVIDE WORK AREA*
             SSALIST=YES               PROBLEM PGM PROVIDES SSA LIST
       L     R3,TCADLIO                PICK UP ADDRESS OF SUPPLIED
*                                      WORK AREA
```

DL/I REQUESTS IN A COBOL PROGRAM (CICS/OS/VS)

Upon program entry, the COBOL programmer should obtain PCB addresses by issuing a DFHFC TYPE=(DL/I,PCB) request or a DL/I 'PCB' call. After CICS/OS/VS returns control, the programmer moves the contents of TCADLPCB to the BLL pointer which is the base for the layout of the PCB pointers in the linkage section. He then moves the addresses of the PCBs to their BLL pointers to provide the base addresses for the PCBs. When this is done, the program is in the same state as after an

```
  ENTRY 'DLITCBL' USING PCB1,PCB2
```

statement has been executed in an IMS/VS DL/I application program.

For an explanation of how BLL pointers to 01 statements in the linkage section are defined, see the discussion of COBOL application programming in "Chapter 2.3. Storage Definition - COBOL" on page 35.

Examples showing how to write DL/I requests are given below. Only some combinations of operands are shown, but other combinations are acceptable. Note that, in a DFHFC request, BLL pointers to the PCB and work area are used rather than actual field names. This is the only way the addresses can be passed to DL/I.

```
WORKING-STORAGE SECTION.
77   PSBNAME PIC X(8) VALUE 'PSBNAME'.
77   PCB-FUNCTION PIC X(4) VALUE 'PCB/'.
77   FUNCTION-1 PIC X(4) VALUE 'DLET'.
77   SSA-COUNT PIC S9(8) COMP VALUE 2.
LINKAGE SECTION.
01   DFHBLLDS COPY DFHBLLDS
     02  ...                              NOTE POINTERS TO OTHER CICS
*                                         AREAS NEEDED
     02  B-PCB-PTRS PIC S9(8) COMP.
     02  B-PCB1 PIC S9(8) COMP.
     02  B-PCB2 PIC S9(8) COMP.
     02  B-WORKAREA PIC S9(8) COMP.
     02  B-SSAS PIC S9(8) COMP.
01   DFHCSADS COPY DFHCSADS.
01   DFHTCADS COPY DFHTCADS.
     .                                    NOTE TWO DEFINITIONS.
     .                                    NOTE OTHER AREA DEFINITIONS.
     .
01   PCB-PTRS.
     02  PCB1-PTR PIC S9(8) COMP.
     02  PCB2-PTR PIC S9(8) COMP.
01   PCB1.
     .
     .
     .
01   PCB2.
     .
     .
     .
01   WORKAREA.
     02  FILLER PIC X(8).                 NOTE STORAGE PREFIX.
     02  WORKA1 PIC X(40).
     .
     .
     .
01   SSAREA.
     02  FILLER PIC X(8).
     02  SSA1 PIC X(40).
     02  SSA2 PIC X(60).
     .
```

```
    .
    .
PROCEDURE DIVISION.
* GET PCB ADDRESSES
    DFHFC TYPE=(DL/I,PCB)            GET PSB FOR THIS PROGRAM
* GET PCB ADDRESSES VIA CALL
    CALL 'CBLTDLI' USING PCB-FUNCTION,PSBNAME.
                                     NOTE GET PCB'S FOR SPECIFIED PSB.
* SAVE PCB ADDRESSES IN BLL TABLE SO PCB'S CAN BE ADDRESSED
    MOVE TCADLPCB TO B-PCB-PTRS.
    MOVE PCB1-PTR TO B-PCB1.
    MOVE PCB2-PTR TO B-PCB2.
* OPTIONALLY, ACQUIRE STORAGE FOR WORK AREA
    DFHSC TYPE=GETMAIN,...
    MOVE TCASCSA TO B-WORKAREA.
* OPTIONALLY, ACQUIRE STORAGE FOR SEGMENT SEARCH ARGUMENTS
    DFHSC TYPE=GETMAIN,...
    MOVE TCASCSA TO B-SSAS.
* CALL DL/I VIA CALL
    CALL 'CBLTDLI' USING FUNCTION-1,PCB1,WORKA1,SSA1,SSA2.
*
* EXAMPLE 1 OF DFHFC MACRO INSTRUCTION
    DFHFC TYPE=(DL/I,GHU),           FUNCTION                          *
          PCB=B-PCB1,                PCB POINTER                       *
          WRKAREA=B-WORKAREA,        WORK AREA POINTER                 *
          SSAS=(SSA-COUNT,SSA1,SSA2) SSA COUNT AND NAMES
*
* EXAMPLE 2 OF DFHFC MACRO INSTRUCTION
    MOVE 'GNP ' TO TCADLFUN.         NOTE PRELOAD FUNCTION.
    MOVE B-PCB1 TO TCADLPCB.         NOTE PRELOAD PCB ADDRESS.
    DFHFC TYPE=DL/I,                 FUNCTION PRELOADED                *
          SSAS=NO                    PCB ADDRESS PRELOADED             *
                                     WORK AREA TO BE ACQUIRED          *
                                     NO SSA'S
    MOVE TCADLIO to B-WORKAREA.      NOTE SAVE ACQUIRED WORK AREA ADDR.
```

DL/I REQUESTS IN A PL/I PROGRAM (CICS/OS/VS)

Upon entry to his program, the PL/I application programmer should get PCB addresses through a DFHFC TYPE=(DL/I,PCB) macro or a DL/I 'PCB' call. When CICS returns, the base address of a structure of PCB pointers is in TCADLPCB. The PL/I programmer must move the value from TCADLPCB to the based variable for his declared structure of PCB pointers. He then loads the pointers to all PCBs from this structure. When this has been done, the program is in the same state as an IMS/VS DL/I application program in which the

```
  DLITPLI: PROCEDURE (pcbnamel,...)
                OPTIONS(REENTRANT,MAIN);
```

statement has been executed.

The PL/I programmer may then make DL/I requests, either through CALLs or DFHFC macros. Note that the PCB and WRKAREA operands in a DFHFC request specify the addresses of pointers to fields rather than of the fields desired.

```
%INCLUDE DFHCSADS;                            /* CSA DEFINITION */
%INCLUDE DFHTCADS;                            /* TCA DEFINITION - INCLUDES */
                                              /* DL/I FIELDS */
             1 PCB_POINTERS BASED (B_PCB_PTRS),
               2 PCB1_PTR POINTER,
               2 PCB2_PTR POINTER;
                 .
                 .
    DECLARE 1 PCB1 BASED (BPCB1),             /* PCB DEFINITIONS */
               2 ...
               2 ... ;
    DECLARE 1 PCB2 BASED (BPCB2),
               2 ...
               2 ... ;
    DECLARE 1 DLI_IOAREA BASED (BDLIIO), /* DL/I I/O AREA */
               2 STORAGE_PREFIX CHAR(8),
               2 IOKEY CHAR(6),
               2 ... ;
    DECLARE 1 DLI_SSADS BASED (BSSADS),  /* DL/I SSA LIST */
               2 STORAGE_PREFIX CHAR(8),
               2 SSA1,
                 3 SSA1KEY CHAR(6),
                 3 ...
               2 SSA2,
                 3 ...
                 3 ...;
    DECLARE PSBNAME CHAR(8) INIT ('PSBNAME');
    DECLARE PCB_FUNCTION CHAR(8) INIT ('PCB ');
/*  OBTAIN PCB POINTERS  */
       DFHFC TYPE=(DL/I,PCB)             GET PSB FOR THIS PROGRAM
/*  OBTAIN PCB POINTERS VIA CALL */
       CALL PLITDLI (PARM_CT,PCB_FUNCTION,PSBNAME): /* GET SPECIFIED PSB */
/*  SAVE POINTERS IN PCB BASES  */
    B_PCB_PTRS=TCADLPCB;
    BPCB1=PCB1_PTR;
    BPCB2=PCB2_PTR;
/*  ACQUIRE STORAGE FOR DL/I I/O AREA  */
       DFHSC TYPE=GETMAIN,CLASS=USER,...
       BDLIIO=TCASCSA;
/*  OPTIONALLY, ACQUIRE STORAGE IN WHICH TO BUILD SSA'S */
       DFHSC TYPE=GETMAIN,CLASS=USER,...
       BSSADS=TCASCSA;
/*  OPTIONALLY, BUILD SEGMENT SEARCH ARGUMENTS */
       SSA1KEY=TERMKEY;
```

```
/* CALL DL/I  */
     CALL PLITDLI(PARM_CT,DLI_FUNCTION,PCB1,IOKEY,SSA1,
     SSA2);
/* EXAMPLE 1 OF DFHFC MACRO INSTRUCTION */
   DFHFC TYPE=(DL/I,ISRT),                                   *
         PCB=BPCB1,            PCB POINTER                   *
         WRKAREA=BDLIIO,       WORK AREA POINTER             *
         SSAS=(2,SSA1,SSA2)    SSA COUNT AND NAMES
/* EXAMPLE 2 OF DFHFC MACRO INSTRUCTION */
   TCADLPCB=BPCB1;
   DFHFC TYPE=(DL/I,GU),             PCB PRELOADED           *
         SSAS=(SSA_COUNT,SSA1,SSA2)  WORK AREA TO BE ACQUIRED *
                                     SSA COUNT AND NAMES
   BDLIIO=TCADLIO;                   /* SAVE WORK AREA ADDRESS */
/* EXAMPLE 3 OF DFHFC MACRO INSTRUCTION */
   TCADLFUN='GN';                    /* PRELOAD FUNCTION */
   TCADLIO=BDLIIO;                   /* PRELOAD WORK AREA ADDRESS */
   DFHFC TYPE=DL/I,                  FUNCTION PRELOADED      *
         PCB=BPCB1,                  PCB POINTER             *
         WRKAREA=YES,                WORK AREA ADDRESS PRELOADED *
         SSAS=NO                     NO SSA'S
```

When using the PL/I Optimizing Compiler, all SSAs used in DFHFC macros and all parameters used in CALLs must be defined as elementary items. This can be done by defining structures based on the same pointers as the structures containing the nonelementary definitions, as follows:

```
  DCL 1 DLI_CALL_SSADS BASED (BSSADS),
        2 STORAGE_PREFIX CHAR(8),
        2 CALL_SSA1 CHAR(...),
        2 CALL_SSA2 CHAR(...);
  /* SET UP SSA1 AND USE IN CALL */
  SSA1KEY=SEARCH_KEY;
  DFHFC TYPE=DL/I,
        SSAS=(SSA_COUNT,CALL_SSA1)
  CALL PLITDLI (PARM_CT,FUNCTION,PCB1,
        IOKEY,CALL_SSA1);
```

OPERANDS OF DFHFC MACRO (DL/I)

DLINA=symb-addr
specifies the entry label of the user-written routine to which control is passed if the CICS-DL/I interface is inactive.

FUNCNS=symb-addr
specifies the entry label of the user-written routine to which control is passed if a DL/I function request (a request other than PCB or TERM) is made and the task has no PSB scheduled.

INVREQ=symb-addr
specifies the entry label of the user-written routine to which control is passed if:

1. One of the conditions DLINA, FUNCNS, LANGCON, PSBFAIL, PSBNA, PSBNF, PSBSCH, TASKNA, or TERMNS occurs and the associated operand is omitted
2. An error condition is detected, as follows:
   a. The required data base is not in the FCT
   b. The required data base is not open according to the FCT
   c. An invalid argument has been passed to DL/I.

If an INVREQ condition occurs and the INVREQ and an associated expansion operand(s) are both omitted, processing continues with the next sequential instruction in the application program.

LANGCON=symb-addr (CICS/DOS/VS only)
specifies the entry label of the user-written routine to which control is passed if the calling program is in a different source language than the called PSB.

NORESP=symb-addr
specifies the entry label of the user-written routine to which control is passed upon normal execution of the request, that is, if the PSB is located and the PCB addresses are returned, or when the application program regains control. The CICS-DL/I interface must have been able to pass control to DL/I and a DL/I pseudoabend of the transaction cannot have occurred. The return code in the PCB must be checked to determine whether DL/I was able to service the request. NORESP signifies "normal response." If this operand is omitted, but a described condition applies, processing

continues with the next sequential instruction in the application program.

PCB=
specifies the field that contains the address of the PCB.

symb-addr
is the symbolic address of the field containing the address of the PCB.

(register)
is valid only when assembler language is used. It is the number of a register that contains the address of the PCB.

PSB=
specifies the name of the PSB to be scheduled for the transaction.

'psbname'
is the name of the PSB to be used.

symb-addr
is the symbolic address of an eight-byte field containing the name of the PSB, padded to the right with blanks.

YES
indicates that the name of the PSB has been placed in TCADLPSB by the application program.

If this operand is omitted, the name of the program associated with the transaction in the CICS program control table (PCT) is used as the PSB name.

PSBFAIL=symb-addr
specifies the entry label of the user-written routine to which control is passed if the PSB fails to initialize, also specifies the entry label of the user-written routine to which control is passed if the data base specified in the PCB used in this request is logically (not necessarily physically) closed. The PCB does not contain a DL/I AI status code.

PSBNA=symb-addr (CICS/DOS/VS only)
specifies the entry label of the user-written routine to which control is passed if the task is not authorized to access this PSB.

PSBNF=symb-addr
specifies the entry label of the user-written routine to which control is passed if the PSB cannot be found in the PSB directory.

PSBSCH=symb-addr
specifies the entry label of the user-written routine to which control is passed if a PSB is already scheduled for this task.

SSALIST=
specifies whether or not segment search arguments are used in this request and if so, identifies the list containing these arguments.

YES
indicates that a list of segment search arguments is used and that the address of the list has been placed in TCADLSSA by the application program.

NO
indicates that no SSA list is used in this request.

symb-addr
is the symbolic address of a field that contains the address of the SSA list.

(register)
is valid only when assembler language is used. It is the number of a register that contains the address of the SSA list.

If this operand is specified, SSAS cannot be specified.

SSAS=
specifies whether or not segment search arguments are used in this request and, if so, identifies them.

NO
indicates that no SSAs are used in this request.

([ssacount][,ssa1][,ssa2,...])
specifies the names of segment search arguments used in this request (thereby creating an SSA list). The ssacount parameter specifies the number of SSAs to be used; it is the address of a fullword containing the count, or, in the case of assembler language, may be expressed as a numeric value. Each ssa specification represents an element of the SSA list. The first element of an SSA list, or it may point to a fullword containing this count; the remaining elements represent addresses of SSAs. If the first element of an SSA list is not a count, all elements of the SSA list are assumed to be addresses of SSAs; the high-order bit of the last element of the list must be

set on to indicate the end of the list.

([(register1)][,(register2),...])
is interpreted as described above; that is, R1 contains a count of the SSAs in the list or is the first list entry, R2 is the first or second list entry (depending on whether a count has been specified), and so on.

If this operand is specified, SSALIST cannot be specified.

TASKNA=symb-addr (CICS/DOS/VS only)
specifies the entry label of the user-written routine to which control is passed if the calling task is not authorized to access DL/I data bases.

TERMNS=symb-addr
specifies the entry label of the user-written routine to which control is passed if a termination request is made and the task has no PSB scheduled.

WRKAREA=
specifies the address of the work area to be used.

symb-addr
is the symbolic address of a field that contains a pointer to the work area.

YES
indicates that the address of the work area to be used has been placed in TCADLIO by the application program.

(register)
is valid only when assembler language is used. It is the number of a register that contains the address of the work aea.

If this operand is omitted and a Gxxx function is to be performed, the CICS-DL/I interface acquires storage for the work area and returns the address of the work area at TCADLIO. The application program must save this address upon return. If any other type of function is requested, the application program must provide the work area. A work area whose address is specified in a DFHFC macro or placed at TCADLIO prior to execution of the DFHFC macro includes the CICS storage accounting area prefix. A work area specified in a CALLDLI or CALL statement does not.

PART 4. DATA COMMUNICATION OPERATIONS

CHAPTER 4.1. INTRODUCTION TO DATA COMMUNICATION OPERATIONS

This part describes the data communication operations Terminal Control, Basic Mapping Support, and Batch Data Interchange.

The essential differences between these data communication facilities is that terminal control is the basic method of communicating with devices, whereas both basic mapping support (BMS) and batch data interchange (BDI) extend the facilities of terminal control to simplify further the manipulation of data streams. In fact, both BMS and BDI use terminal control facilities.

Terminal Control (Chapter 4.2) provides specific macros and options for particular devices so that the application programmer can tailor his input and output requests according to the requirements of the devices. However, application programs written in this way are dependent on data formatting requirements of devices and therefore the application programmer must have detailed knowledge of the devices.

Basic Mapping Support (Chapter 4.3) provides macros and options that the application programmer can use to format data in a standard manner. BMS performs the conversion of data streams provided by the application program to conform to the requirements of particular devices. Conversely, data received from a device is converted by BMS to a standard form. However, not all devices supported by CICS can be used with BMS and therefore 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 appropriate _CICS Facilities and Planning Guide_.

Batch Data Interchange (Chapter 4.4) is a set of macros that may be used either instead of terminal control macros, or in conjunction with BMS macros to communicate with the batch logical units of the 3790 and 3770 subsystems.

CHAPTER 4.2. TERMINAL CONTROL (DFHTC MACRO)

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

Terminal control uses the standard access methods available with the host operating system. The Basic Telecommunications Access Method (BTAM) is used by CICS 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 initiation, synchronization of I/O 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 terminal control program monitors which task is associated with which terminal. The task to be initiated is determined as described in "Terminal-Oriented Task Identification" on page 115. Terminal control detects and logs errors, and also, where appropriate, inserts a default.

Terminal control is used for communication with terminals. In SNA systems, however, it is used to control communication with logical units. 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 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 any 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.

Facilities that apply specifically to logical units are described in "Facilities for Logical Units" on page 110.

Terminal control macro instructions are provided to request the following services that are applicable to most, or all, of the types of terminal supported by CICS:

- Read data from a terminal.
- Write data to a terminal.
- Synchronize terminal I/O for a transaction.
- Converse with a terminal.
- Read or write records to a card reader, 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. This facility allows transactions to be tested when normal communications terminals are not available.

For more information about the last of these services, see "Chapter 7.2. Sequential Terminal Support" on page 293.

Other services available in response to terminal control macros apply to specific types of terminal. Because many types of terminal are supported by CICS, many special services are provided. For a list of terminals supported by CICS, see the appropriate
| CICS Facilities and Planning Guide. The following list is representative of the terminal-oriented input/output services available:

- Read the entire contents of a buffer (3270 Information Display System).
- Read a message containing both uppercase and lowercase data (3270 Information Display System).
- Print out the contents of an information display buffer on a printer (3270 Information Display System).
- Transmit a message to a common buffer (2980 General Banking System).
- Read or write data in transparent mode, that is, without translation (System/7, System/370, System/3, 2770 Data Communication System, 2780 Data Transmission Terminal, 3600 Finance Communication System (BTAM),

3740 Data Entry System, and the 3780 Data Communications Terminal).

- Use the attention key to interrupt a write operation or signal a read attention request (for example, on the 2741 Communication Terminal).

The general form of the terminal control macro (DFHTC) resembles that of other CICS macros. Keyword operands are separated by commas. Although most CICS macros use only one entry following the keyword TYPE, the DFHTC macro can contain several, for example

```
DFHTC TYPE=(WRITE,READ)
```

causes a write to the terminal, a wait for that write to be completed (an implied wait), and a read from the terminal to which data has just been written.

Another example is the macro

```
DFHTC TYPE=(ERASE,WRITE,READ,WAIT)
```

which causes an erase and then a write to a terminal, followed by an implied wait, followed by a read and a requested wait. The latter wait ensures that the read is complete before control is returned to the application program.

Two separate DFHTC macros must be used when two options that would be incompatible for the same macro are needed. Examples of incompatible options are:

```
DFHTC TYPE=(WRITE,PRINT)

DFHTC TYPE=(WRITE,READB)

DFHTC TYPE=(PRINT,READ)
```

In such cases, the first macro should include the WAIT option; for example:

```
DFHTC TYPE=(WRITE,WAIT)

DFHTC TYPE=READB
```

As in other CICS macro operands, if only one entry is given in the TYPE operand, no parentheses are necessary.

The application programmer must determine the combination of keywords that follow TYPE=, depending on the terminal (and sometimes, access method) used and the operations required. Additional operands may be required or desired, again depending upon the terminal and access method used. Some common input/output requests are discussed later in the chapter.

Before using the DFHTC macro to request terminal services, the application program must include instructions that:

1. Symbolically define the TCTTE and TIOA by copying the appropriate storage definitions (DFHTCTTE and DFHTIOA) provided by CICS. (It is assumed that the storage definitions for the CSA and TCA have already been copied, as described in Part 2.)

2. Establish addressability for the TCTTE by specifying a symbolic base address. If using assembler language or COBOL, the application programmer must obtain the base address of the TCTTE from TCAFCAAA and place it in TCTTEAR; with PL/I, addressability for the TCTTE is established automatically. Any field in the TCTTE can then be accessed by field name. Addressability for the TIOA must be established each time a DFHTC TYPE=READ or TYPE=WRITE macro is issued. The ways of doing this are described in the following section.

FACILITIES FOR ALL TERMINALS AND LOGICAL UNITS

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

READ DATA FROM A TERMINAL OR LU

Data can be read from a terminal or logical unit by issuing the

```
DFHTC TYPE=READ
```

macro. The incoming data is placed in a TIOA acquired by terminal control, which also places the address of the TIOA in TCTTEDA. The operation will be complete when another terminal control TYPE=WAIT has been issued. On completion of the read operation, the application program must copy the address from TCTTEDA to the TIOA base address register (TIOABAR): any field in the TIOA can then be accessed by field name.

The length of the data read into the TIOA is stored in TIOATDL.

Terminal control attempts to reuse TIOAs that have been used in previous operations. For this purpose, it maintains a chain of TIOAs whose addresses are anchored in the TCTTE. If no TIOA is attached to the chain, or if the existing TIOAs are too short or are otherwise unsuitable, terminal control acquires a new TIOA. The current TIOA as addressed by TCTTEDA may be freed by terminal control unless the SAVE operand is specified.

Note that when using BTAM, data is read into a line input/output area (an LIOA). BTAM determines which terminal has been read, changes the LIOA into a TIOA and places the address of that TIOA in TCTTEDA after freeing the area previously pointed to by TCTTEDA.

A new TIOA is also acquired by terminal control for the read when the

```
  DFHTC TYPE=(READ,SAVE)
```

macro is issued. All TIOAs currently chained off the TCTTE are retained and may subsequently be reused; a new TIOA is dynamically acquired for this read and is added to the chain.

A write, followed by a read operation, can be specified in a single request. See "Write Data and Read Reply", below.

When a TIOA which was previously obtained as an LIOA by terminal control is passed to a user task, the contents of the data part cannot be guaranteed beyond the data length supplied in TIOATDL. Therefore users should not attempt to interrogate the contents of a TIOA beyond this supplied length.

When the contents of a 3270 buffer are read (by using DFHTC TYPE=READB), the programmer should be aware that the attention identifier (AID) byte and the cursor address are made available at TCTTEAID and TCTTECAD respectively. A set of standard symbolic names for testing the 3270 AID is provided in a copy book called DFHAID. For further details refer to "Standard Attention Identifier List (DFHAID)" on page 184.

WRITE DATA TO A TERMINAL OR LU

Data is written to a terminal or logical unit using the

```
  DFHTC TYPE=WRITE
```

macro. (For a transaction that has been started by automatic transaction initiation (ATI), a DFHTC TYPE=WRITE macro should always precede the first DFHTC TYPE=READ macro in a transaction.)

Before using this macro, the application program must acquire a TIOA in which to build the data to be transmitted, and must place the address of the TIOA in TCTTEDA and the length of the data to be written in TIOATDL. The maximum data length is 32,767 bytes which includes the length of the function management header (FMH) when writing to a logical unit.

The required TIOA is acquired by a DFHSC TYPE=GETMAIN,CLASS=TERMINAL macro. CICS places the address of the TIOA in TCASCSA, from where it must be copied into the TIOA base address register (TIOABAR).

The application program must not change the contents of TCTTEDA until after the I/O operation has completed. The operation will only be complete when another terminal control request has been issued (that is, TYPE=WAIT or TYPE=READ).

If WAIT is not specified on a terminal control TYPE=WRITE operation, the operation may be deferred until the next terminal control request. When the next terminal control request is issued, the SNA flows are optimized before the actual I/O is issued. For example, a terminal control write followed by a terminal control read could cause two flows to be sent, whereas only one flow is sent if it can be determined that the next operation is a read request.

When writing data to a 3600 (nonpipeline) or 3790 inquiry logical unit, the application program must not put data into the first three bytes of the TIOA, unless it is building its own FMH. See "Function Management Header (FMH)" on page 112. The FMH is built either by CICS or by the application program.

When the write operation is completed by terminal control, the TIOA is released to a dynamic storage pool (unless SAVE is specified). Subsequent reference to this TIOA by the application program will produce unpredictable results.

However, a TIOA can be reused by the application program after a write if the request to write data to a terminal uses the

```
  DFHTC TYPE=(WRITE,SAVE,WAIT)
```

macro. In this case, the TIOA is not released by terminal control. The WAIT parameter ensures that the write of the TIOA is complete before the area is reused.

If a dump of the TIOA is required following a terminal control write, the SAVE and WAIT operands should be included with the DFHTC TYPE=WRITE macro that precedes the DFHDC macro.

WRITE DATA AND READ REPLY

As stated earlier, a write followed by a read operation can be specified in a single request by issuing the

```
  DFHTC TYPE=(WRITE,READ)
```

macro. A typical use for this macro occurs in a conversational environment in which the application program writes a question to the terminal, waits for a reply, and subsequently reads the reply.

Because the SAVE parameter is not specified, terminal control can reuse the TIOA (from which data is written) as a TIOA for the input data. Under certain conditions, however, a new TIOA is obtained for the read operation, for example:

- Local 3270 terminals.
- PSEUDOBIN specified with READ,WRITE.
- The TIOA length for the WRITE instruction less than that specified by the system programmer in the DFHTCT TYPE=TERMINAL,TIOAL=length specification (binary synchronous terminals) or in the DFHTCT TYPE=LINE,INAREAL=length specification (all other terminals).
- Certain error conditions.

The user should always reload TIOABAR from TCTTEDA following the (WRITE,READ) macro.

For a terminal connected to the 7770 Audio Response Unit, a read request that does not include the WRITE parameter causes the "ready" message (defined in the terminal control table by the system programmer) to be written to the terminal before the read operation occurs.

If both a write and a read operation are specified in a single request by issuing

```
  DFHTC TYPE=(WRITE,READ,SAVE)
```

the TIOA used for writing is saved; a new TIOA is then acquired by terminal control for the read. The size of the TIOA is determined by the system programmer when specifying the TCTTE for the terminal (rather than by the size of the TIOA used for the write). If the saved TIOA is reused later for either writing or reading, the application program must place the address of the TIOA into TCTTEDA prior to issuing the request to use the area.

The manner in which the address of a TIOA is "remembered" is the application programmer's responsibility .

Upon completion of a (WRITE,READ,SAVE), place the value at TCTTEDA into TIOABAR to establish addressability for the newly-acquired TIOA.

SYNCHRONIZE TERMINAL I/O (WAIT)

In a task under which more than one terminal or logical unit operation is performed, the application programmer must ensure that a current terminal operation is complete before another begins. Furthermore, for all (WRITE,READ) and (WRITE,READ,SAVE) requests, control is returned to the application program after the write operation is executed, but the read operation has not necessarily been completed at this time. Therefore, in order that the data resulting from the READ operation can be processed, the application programmer must ensure that the operations are completed. To do this the

```
  DFHTC TYPE=WAIT
```

macro is issued, where the WAIT parameter is coded separately, as shown, or in combination with READ or WRITE. A PUT can be coded in place of a (WRITE,WAIT); a GET can be coded in place of a (READ,WAIT). To ensure that the data has been transferred to the TIOA, a wait must be issued for each read request.

A wait may cause execution of a task to be suspended. If suspension is necessary, control is returned to CICS. Execution of the task is resumed when the write or read is posted complete.

A wait need not be coded for a write if the write is the last terminal operation of the transaction. The TIOA is retained until the data is written, even if the transaction and its associated storage are deleted from the system before the write occurs.

CONVERSE WITH A TERMINAL OR LU

A conversational mode of communication with a terminal or logical unit is requested by the

```
  DFHTC TYPE=CONVERSE
```

macro, where CONVERSE (or CONV) is the same as (WRITE,READ,WAIT). This instruction is always executed in the sequence: WRITE, implied wait, READ, WAIT.

It is possible, for most devices, to use this macro rather than TYPE=READ, but it must not be used for the 3600 or 3650 pipeline logical units. However, its use is recommended for all other logical units.

DISCONNECT A SWITCHED LINE

To break a line connection between a terminal or logical unit and a host processor, the

```
  DFHTC TYPE=DISCONNECT
```

macro is used. This applies only to devices operating on switched lines or to logical units.

| When used with a VTAM terminal,
| DISCONNECT, which does not become
| effective until the task completes,

signs off the terminal, frees the COMMAREA, clears the next TRANID, stops any BMS paging, and, if autoinstall is in effect, deletes the terminal definition.

When used with logical units, DISCONNECT, which does not become effective until the task has been terminated, terminates the session, without causing a physical disconnection.

Note: CICS/OS/VS implements DISCONNECT for World Trade Teletype Terminals by writing a message to the terminal indicating that the terminal operator should manually disconnect.

EXAMPLES

The following examples, in assembler language (ASM), COBOL, and PL/I, show the use of a terminal control macro (DFHTC) that erases the screen, returns the cursor to the upper left corner of the screen, writes to the terminal, and reads from the terminal. The lines of code in the examples are keyed to the notes that follow.

ASM

```
1   L      TCTTEAR,TCAFCAAA
2   DFHSC     TYPE=GETMAIN,
              NUMBYTE=80,
              CLASS=TERMINAL
3   L         TIOABAR,TCASCSA
4   ST        TIOABAR,TCTTEDA
5   MVC       TIOADBA(80),DATA
6   MVC       TIOATDL,=H'80'
               .
               .
7   DFHTC     TYPE=(WRITE,ERASE,
              READ,WAIT)
8   L         TIOABAR,TCTTEDA
```

COBOL

```
1   MOVE TCAFCAAA TO TCTTEAR.
2   DFHSC     TYPE=GETMAIN,
              NUMBYTE=80,
              CLASS=TERMINAL
3   MOVE      TCASCSA TO TIOABAR.
4   MOVE      TIOABAR TO TCTTEDA.
5   MOVE      DATA TO TIOADATA.
6   MOVE      80 TO TIOATDL.
               .
               .
7   DFHTC     TYPE=(WRITE,ERASE,
              READ,WAIT)
8   MOVE      TCTTEDA TO TIOABAR.
```

PL/I

```
1   TCTEAR=TCAFCAAA;
2   DFHSC     TYPE=GETMAIN,
              NUMBYTE=80,
              CLASS=TERMINAL
3   TIOABAR=TCASCSA;
4   TCTTEDA=TIOABAR;
5   TIODATA=DATA;
6   TIOATDL=80;
              .
              .
7   DFHTC     TYPE=(WRITE,ERASE,
              READ,WAIT)
8   TIOBAR=TCTTEDA;
```

The statements in the above examples:

1. Establish addressability for the TCTTE.

2. Acquire storage for use as a terminal input/output area by use of the DFHSC macro.

3,4 Place the address of the acquired area into TCTTEDA.

5. Place data in the TIOA.

6. Place the length of the data to be written into TIOATDL.

7. Issue a terminal control macro to a 3270 terminal, thus erasing the screen, returning the cursor to the upper left corner of the screen, writing to the terminal, and reading from the terminal (allowing terminal control to manage storage for the TIOA).

8. Establish addressability to the TIOA into which the data has been read.

FACILITIES FOR LOGICAL UNITS

A CICS application program communicates with a TCAM or VTAM logical unit in much the same way that it does with BTAM or TCAM terminals (that is, by using the various forms of the DFHTC macro 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 facilities provided by CICS to enable the application programmer to comply with these protocols.

The types of logical units and the related protocols for each of the SNA subsystems supported by CICS are described in the IBM 3270 Data Stream Device Guide, and in the CICS subsystem guides for the IBM 4700/3600/3630, IBM 3650/3680, IBM 3767/3770/6670 and the IBM 3790/3730/8100. See "Bibliography" on page 351.

SEND/RECEIVE MODE

For SNA logical units, a transaction conversing with such a logical unit must conform to the send/receive protocols of SNA, unless the read-ahead queueing feature has been specified.

However, a transaction is normally in send mode and can issue any terminal control request. For displays (for example, the 3270), the send/receive mode is transparent to the application program, but for logical units that perform chaining, or make use of the full SNA protocols (for example, the 3767), the send/receive mode should be taken into account.

If the application program is in receive mode, flag TCTEURCV in field TCTERCVI is set on, and the application program must continue to issue terminal control READ requests.

For compatibility, the read-ahead queueing feature (RAQ=YES specified in the DFHSG PROGRAM=TCP system macro) is provided so that the application program is independent of the send/receive mode. However, it is recommended that application programs be changed to use SNA send/receive protocols and that, wherever possible, they specify RAQ=NO.

OVERLAPPING LOGICAL UNIT OUTPUT

Write operations are not initiated until a subsequent terminal control operation to the logical unit is issued, a syncpoint is taken, or the task terminates.

If a terminal control write operation is awaiting completion, a terminal control wait should be issued unless the next operation is a read request, in which case a terminal control read can be issued directly.

A terminal control write and wait request causes the operation to be initiated immediately. If only a terminal control write is issued, the operation is delayed until the next operation so that SNA flow handling can be improved.

The point at which a wait is satisfied depends upon whether task protection, message integrity, or DEFRESP=YES is requested for the task. Task protection and message integrity are specified, by the system programmer, in the DFHPCT macro; if data is sent with task protection or message integrity, a wait is completed when a logical unit responds to the write request; otherwise, the wait is completed after VTAM has accepted the output request.

If a task is operating under task protection or message integrity and an exception response is returned for an output request, the output message is still available in the TIOA. The node error program (NEP) can therefore request that the operation be retried as many times as specified by the installation.

CHAINING OF INPUT DATA

For transmission purposes, data handled by a logical unit is divided into request/response units (RUs). The data may be transmitted as one or more RUs, called a chain, depending on the length of the data, and on the maximum size of the RU defined for the logical unit or that has been defined for the terminal network in general.

Each RU contains a set of indicators that specify whether the RU is the first, middle, or end, of the chain (FOC, MOC, or EOC, respectively). If the chain consists of only one RU, this RU contains both the FOC and the EOC indicators.

Data is transmitted as a chain of one or more RUs from a logical unit to the application program. If the chain contains more than one RU, further read requests are required, one for each RU, unless chain assembly has been specified. (Chain assembly is described later in the chapter.) The length of each RU must be less than or equal to the maximum RU size.

The EOC operand of the DFHTC TYPE=(READ,WAIT) macro is used to test for the presence of the EOC indicator. If it is present, that is, the complete chain has been received, control is passed to a user-written routine that provides additional processing.

For some logical units, the data transmitted may contain a function management header (FMH), in which case, inbound-FMH processing will take precedence over EOC processing. (Inbound FMH is described later in the chapter.)

Further, if the FMH indicates the end of the data set, control will be passed to the EODS routine instead of to the INBFMH or EOC routines. The DFHTC TYPE=WAIT macro with the EOC operand specifies that control is to be passed to an EOC routine from within either the inbound FMH or the EODS routine.

An FMH may also occur in the first RU of a chain that contains more than one RU. In this case, control is passed to the INBFMH routine when a DFHTC TYPE=READ is satisfied by that RU.

The application program must read all the data from the logical unit, that is, it should not terminate (except abnormally) before EOC has been received. Application programs should also ensure that the complete data stream has been received from the logical unit; this will be ensured if the application program is not in receive mode when it terminates.

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 supplied in the TIOA exceeds the RU size, CICS automatically breaks up the data into RUs and transmits these RUs as a chain. During transmission from CICS to the logical unit, RUs are marked FOC, MOC, or EOC 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 can inhibit the end-of-chain marker on the last (or only) RU resulting from the write request by including the CCOMPL=NO operand (specifying that the chain is not yet complete). The data supplied in the TIOA for the next write request is treated as a continuation of the chain.

CHAIN ASSEMBLY

Chain assembly, which is specified by the system programmer in the TCTTE, is the process of assembling RUs together to form a chain which is transmitted as an entity to the application program in a single TIOA in response to a single read request. This ensures the integrity of the whole chain before presentation to the application program. If the EOC operand is specified in the read request, the EOC routine receives control for every read request (except when an FMH is received and the appropriate EODS or INBFMH routine is specified, as described earlier in the chapter under "Chaining of Input Data").

The length of the TIOA required to hold a chain is unknown because a chain can consist of any number of RUs. To allow for this, two TIOA lengths can be specified in the TCTTE by the system programmer. The first length specifies a TIOA that will normally be provided. The second specifies a larger TIOA for use when the normal TIOA is not large enough. If the larger TIOA cannot hold the complete chain, the node abnormal condition program (DFHZNAC) is invoked and the task is terminated abnormally. Additional processing of the chain can, however, be initiated by the node error program (DFHZNEP) when a further read request will be needed to cause transmission of the rest of the chain. The use of two TIOA sizes minimizes storage requirements.

Chain assembly is recommended for most interactive applications, because the input data is usually made up of a chain of more than one RU. In many cases the application program logic is simplified by use of this option.

LOGICAL RECORD PRESENTATION

Normally a chain contains the data to be processed and this chain is presented to the application program in a TIOA as specified in the TCTTE.

In some cases, however, the chain contains many logical entities for processing. These may be each RU itself, or the RUs may be further subdivided into logical records delimited by interrecord separator control characters, or new line characters.

The entire RU will be presented to the application program if chain assembly is not specified in the TCTTE. However, if the data stream is delimited by separators into logical records, the system programmer can specify in the PCT that logical records will be presented to the application program instead of RUs or chains, so overriding on an application basis the TCTTE options for the logical unit.

If the RU contains more than one logical record, the records will be separated by NL (new line), IRS (interrecord separator), or TRN (transparent) characters. Except in the case of LUTYPE4, one 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 unit may contain logical records that span RUs, in which case chain assembly should be specified.

Because a card reader inserts an IRS character after the last nonblank character on the card, the user may receive card images that are less than 80 characters in length. Conversely, a series of full cards will begin at 81-character intervals.

For those application programs for which this option is specified, each read request results in one logical record being presented to the application program in a TIOA, 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 TIOA. Therefore, a blank card will appear as a TIOA with 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 TIOA. All the previously described communication features are still in operation. That is, notification of end-of-chain, 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 (either NL, IRS, or TRN), or the end of the assembled chain. The end of chain notification is given with the last logical record of the chain.

DEFINITE RESPONSE

The type of response requested by CICS for outbound data is generally determined by the system programmer when generating the PCT. The system programmer can specify that all outbound data for an application program will require a definite response, or allow the exception-response protocol to be used, which means that a response will be made only if an error situation occurs.

If exception response protocol is used, a negative response may be received and handled on a subsequent command.

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 operand is provided for use on the DFHTC TYPE=WRITE macro. One example of the use of this operand is to request a definite response for every tenth write request, exception response being the general rule.

Because a response cannot be received until the whole chain has been sent, the DEFRESP operand and the CCOMPL=NO operand are mutually exclusive. The DEFRESP operand and the ERASE operand are also 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. For input, the FMH can be passed to the application program or it can be suppressed by CICS.

The rules governing the use of FMHs for each type of logical unit, and the formats of the FMHs, are given in the CICS subsystem guides (for example, the _CICS IBM 3790/3730/8100 Guide_), which are listed in the Bibliography.

Inbound FMH

The CICS application program can request notification when an FMH is included in the data received during a read from a logical unit; when present, the FMH is at the start of the TIOA.

Whether or not inbound FMHs will be passed to the application program is specified by the system programmer in the PCT. It can be specified that no inbound FMHs will be passed, or that only the FMH indicating end of data set (EODS) will be passed, or that all inbound FMHs will be passed, or that the data interchange program (DFHDIP) will process the FMH.

The INBFMH operand of the DFHTC TYPE=READ or WAIT macro specifies that control is to be passed to a user-written routine whenever an inbound FMH is received. Use of the INBFMH operand implies that the WAIT option of the TYPE operand is in effect.

The user-written routine can examine the FMH and take some action depending on, for example, from which device the data has come. The routine then scans the TIOA for input data, starting after the FMH. If the data is initial data from a logical unit, the transaction identification will start after the FMH.

When input data is received as a chain of RUs, only the first (or only) RU of the chain contains an FMH.

Outbound FMH

Some logical units require or allow control information to be specified by means of an FMH. For 3600 (nonpipeline) and 3790 inquiry logical units, CICS will build the FMH, but the application program must reserve space in the TIOA for it. CICS will not build on FMH for any other type of logical unit.

If the FMH is to be built by the application program, the write request must specify FMH=YES. The FMH must start at the beginning of the TIOA.

END OF DATA SET (EODS)

The DFHTC TYPE=EODS macro specifies that an FMH containing an EODS indicator is sent to a 3650 interpreter logical unit. This FMH delimits the output. The end of the input is detected similarly by the EODS operand of a DFHTC TYPE=READ macro.

LOGICAL DEVICE CODE (LDC)

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).

An LDC is a CICS-supported and installation-defined logical device code. 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 (the LDC itself whose code value can range from 0 to 255). A device type and a logical page size are also associated with each LDC. "LDC" or "LDC value" is used in this publication to refer to the code specified by the user. "LDC mnemonic" refers to the two-character symbol that represents the LDC numeric value.

Within the 3601 subsystem, a user-written application program provides the function of the logical unit. For batch and batch data interchange logical units the functions of the logical unit are built in and in general cannot be modified further by the user. The following paragraphs discuss some of the functions that may be provided in a user-written application program.

When a CICS application program issues a write request with the LDC operand specified, 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; the interpretation of that numeric value is the responsibility of the subsystem application program.

As a minimum, the installation can choose a different LDC to correspond to each device attached to the logical unit. The values (codes) chosen for the LDC can correspond exactly to the logical device address (LDA) for each device. The subsystem application program can then take the CICS output data and write it directly to the indicated LDA.

LDCs can be used to provide support for multiple-form printers. When used for these printers, each LDC within a specified range corresponds to a particular type of form. Whenever the subsystem application program receives data with an LDC that indicates a particular printer and a particular form, the application program can check the device to determine whether the correct form is currently on the printer. If the correct form is on the printer, the application program proceeds with the output operation. If the correct form is not on the printer, the application program can request the operator to load the appropriate form and to signal when the load is completed.

Some LDCs can be used to indicate certain standard actions to be undertaken by the application program. Using the LDC in this way can reduce the overhead of writing messages to the subsystem application program. An example of this use of LDCs is an instruction to the application program to turn on specific indicator lights on a device. A range of LDCs can be specified for each device, each LDC within this range corresponding to a specific light. Upon receipt of such an LDC, the application program determines the appropriate device and indicator and issues the commands necessary to turn on the light. Other standard actions that can be invoked by LDCs are dumping operator totals, checking diskettes for transaction backlogs, or indicating a change in operational mode.

The LDC operand of the DFHTC TYPE=WRITE macro is only for use with 3600 (3601) nonpipeline logical units and provides a symbolic way of conveying to CICS the type of FMH it is to build on behalf of the application program. Alternatively, the application program may build its own FMH (which may be greater than three bytes) and indicate this by means of the FMH operand.

Component or destination selection for batch and batch data interchange logical units is accomplished by means of an FMH, the length of which depends on the type of logical unit. The application program must build its own FMH, or use the LDC operands of the basic mapping support (BMS) macros DFHMSD or DFHBMS TYPE=OUT or TYPE=STORE to instruct BMS to build the correct FMH. If the FMH is to be built by the application program the DFHTC CTYPE=LOCATE, LDC=YES macro may be used to symbolically obtain the component selection value to be inserted in the appropriate FMH field. Refer to the IBM 3770 and IBM 3790 guides for a further discussion of component selection.

UNSOLICITED INPUT

If a task is in progress and unexpected data (that is, data from a terminal for which a read request has not been issued) arrives from a start-stop or BSC terminal, CICS ignores the data and it is lost.

If, however, unexpected data arrives from a 3600, 3650, 3767 or 3770 interactive (contention only), or 3790 inquiry logical unit, it is queued and is used to satisfy any future input requests for that logical unit. For the 3270 logical unit (but not for the 3270 LUTYPE2 logical unit , data is queued only if PUNSOL=NO is specified in the DFHSG PROGRAM=TCP macro; otherwise it is lost. Unsolicited input does not occur for the other logical units.

SIGNAL COMMANDS FROM LOGICAL UNITS

Signal data-flow-control commands from the logical unit must be handled by the application program. The DFHTC TYPE=SIGNAL macro allows an address to be specified to which control will pass when a signal command is received. The associated signal code will be stored in the four-byte field TCTESIDI in the terminal control table terminal entry (TCTTE).

If a hard request-change-direction (RCD) signal is received from an LUTYPE4 unit (signal code = X'00010000'), the transaction should either end or read data from the logical unit. Any attempt to write to the unit immediately following a hard RCD would be an error, indicated by the flag TCTERCD in the TCTTE. If a further attempt to write to the logical unit is made, CICS will abnormally terminate the transaction with an abend code of ATCL.

Most logical units that can send a signal command with a code of X'00010000' do so when an attention key is pressed.

BRACKET PROTOCOL

The use of bracket protocol is a means of preventing interruption of the exchange of data between CICS and a logical unit. CICS or the logical unit may send begin-bracket, but only CICS may send the end-bracket. Brackets can delimit a conversation between CICS and the logical unit or merely the transmission of a series of data chains in one direction.

Bracket protocol is used when CICS communicates with a logical unit. The use of brackets is usually transparent to the CICS application program.

Only on the last write operation of a task to a logical unit does the bracket protocol become apparent to the CICS application program. On the last output request to a logical unit, the CICS application program may specify LAST in the DFHTC TYPE=WRITE macro. The last output request is defined as either the last DFHTC TYPE=WRITE macro specified for a task without chain control; or as the write operation that transmits the FOC or OC marker of the last chain of a transaction with chain control.

The LAST specification causes CICS 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 operand is not specified, CICS 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 operand is ignored for a transaction with chain control unless FOC or OC is also specified. Refer to the publication VTAM Concepts and Planning for more details on bracket protocol.

TERMINAL-ORIENTED TASK IDENTIFICATION

When CICS 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 checks these specifications are as follows (see also Figure 13 on page 116).

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 the basic mapping support terminal paging facility?

b) Is the input a paging command? (The terminal operator can enter paging commands defined by the system programmer in the DFHSIT macro. See the appropriate CICS Resource Definition manual.)

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 macro)?

If yes, the specified transaction is initiated.

Test 5:

Is a transaction specified by the TRANSID operand of a DFHPC TYPE=RETURN macro (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 (LPA), or magnetic stripe card reader (OPID)?

c) Is this input (PA, PF, LPA, or OPID) specified by the TASKREQ operand of a DFHPCT TYPE=ENTRY macro? (See the appropriate CICS Resource Definition manual.)

If yes to (a), (b), and (c), the program specified by the PROGRAM operand of 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. Message DFH2001 is sent to the terminal.

Note: 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.

SYNTAX OF THE DFHTC MACRO

This section shows the syntax of the DFHTC macro available for use with each type of device or logical unit, arranged in numerical order.

The syntax displays for each device and for the 3270 logical unit are followed by information specific to that device

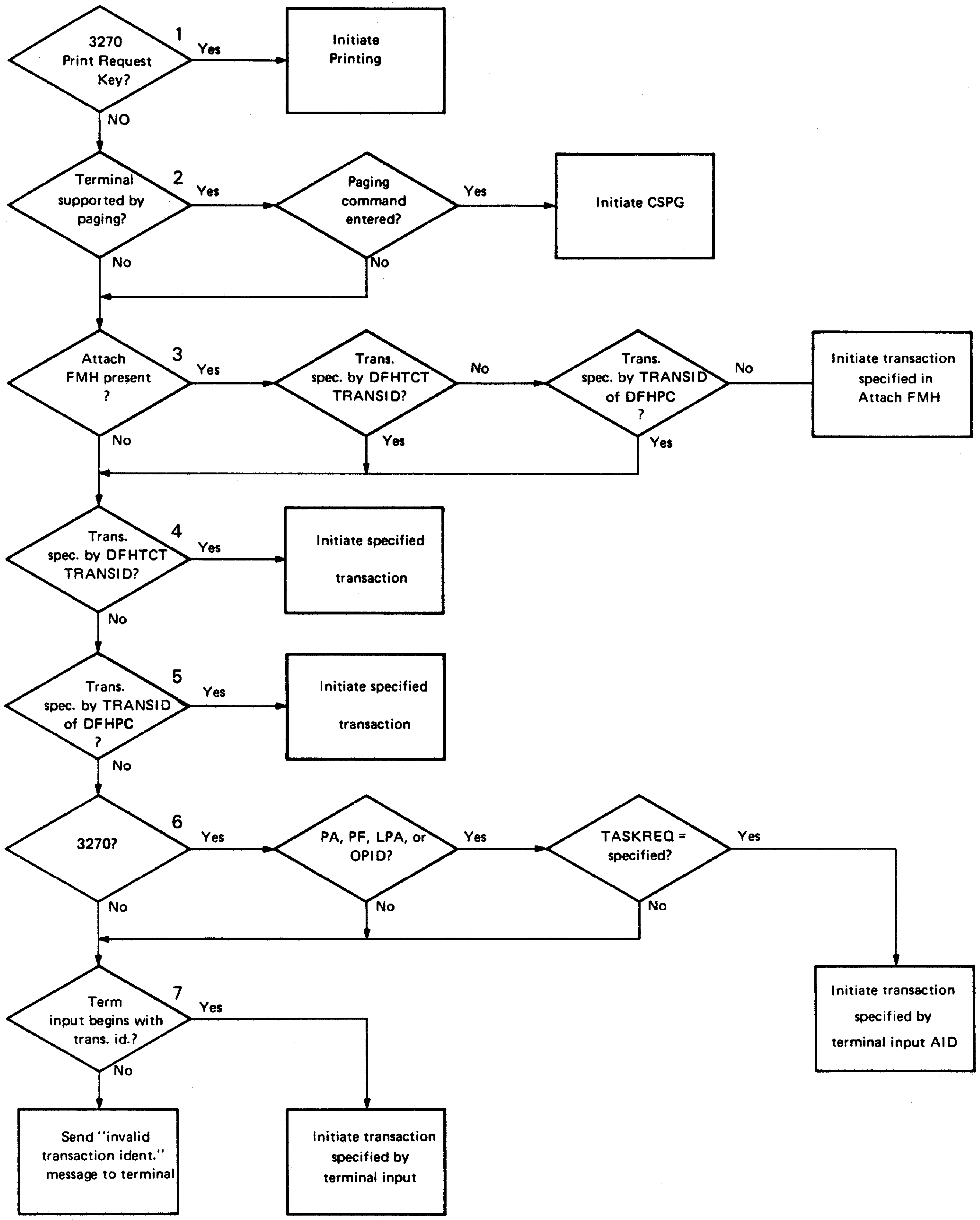

3270 Print Request Key?
1
Yes
Initiate Printing
NO
Terminal supported by paging?
2
Yes
Paging command entered?
Yes
Initiate CSPG
No
No
Attach FMH present ?
3
Yes
Trans. spec. by DFHTCT TRANSID?
No
Trans. spec. by TRANSID of DFHPC ?
No
Initiate transaction specified in Attach FMH
No
Yes
Yes
Trans. spec. by DFHTCT TRANSID?
4
Yes
Initiate specified transaction
No
Trans. spec. by TRANSID of DFHPC ?
5
Yes
Initiate specified transaction
No
3270?
6
Yes
PA, PF, LPA, or OPID?
Yes
TASKREQ = specified?
Yes
No
No
No
Initiate transaction specified by terminal input AID
Term input begins with trans. id.?
7
Yes
No
Send "invalid transaction ident." message to terminal
Initiate transaction specified by terminal input

Figure 13. Terminal-Oriented Task Identification

or logical unit. However, information about 3600, 3650, 3767, 3770, and 3790 logical units is given in the CICS subsystem guides.

TCAM SUPPORTED TERMINALS AND LOGICAL UNITS (CICS/OS/VS ONLY)

Under CICS/OS/VS only, because TCAM permits many applications to share a single network, the CICS-TCAM interface supports data streams rather than specific terminals or logical units.

Operations for terminals and logical units connected through TCAM use the same operands as the terminals and logical units connected through the other access methods used with CICS.

For input, TCAM supports only the READ and READL operations. For output, TCAM supports only the WRITE operation with the optional use of ERASE. The DEST operand can be specified for all TCAM output operations. (The syntax of the DFHTC macro for TCAM operations is given later in the chapter.)

3650 logical units cannot be connected through TCAM.

BTAM PROGRAMMABLE DEVICES

When BTAM is used by CICS for programmable BSC (binary synchronous communication) line management, CICS initializes the communication line with a BTAM "read initial" (TI); the terminal response must be a "write initial" (TI) or the equivalent. If a user-written application program then issues a read, CICS issues a "read continue" (TT) to that line; if the application program issues a write, CICS issues a "read interrupt" (RVI) to that line. If "end of transmission" (EOT) is not received on the RVI, CICS 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.

The programmable terminal response to an RVI must be 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 TCT. CICS issues a TT 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 issues a TI or the equivalent (depending on the type of line). The programmable terminal response must be a read initial (TI) or the equivalent.

If another write is issued by the application program, CICS issues a write continue (TT) to that line. If the application program issues a read after it has issued a write, CICS turns the line around with a "write reset" (TR). (CICS does not recognize a read interrupt.)

When CICS initiates a transaction using automatic transaction initiation, it first of all issues a write initial (TI) or the equivalent. The terminal must respond with a read initial (TI) or the equivalent. Reading from or writing to the terminal can then continue as if the write initial had been caused by a write instruction in the application program.

ATI transactions attached to the device will cause message DFH2503 to be sent to that device. The device must be prepared to action it.

To ensure that binary synchronous terminals (for example, System/370, 1130, 2780) remain coordinated, CICS 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 macros and CICS actions, are summarized in Figure 14 on page 118.

Input data is deblocked to ETX, ETB, RS, and US characters. These characters are moved with the data to the TIOA but are not included in the data length (TIOATDL). The CICS application programmer should be aware that characters such as NL, CR, LF, and EM are passed in the TIOA as data.

TELETYPEWRITER PROGRAMMING

The teletypewriter (World Trade only) uses two different control characters for print formatting:

<	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 application programmer should always use < first; that is <≡ or <≡≡≡, but never ≡<, otherwise following characters (data) may be printed while the typebar is moving to the left.

Application Program	CICS (note 1)	Programmable Terminal Program
	Read initial (TI)	Write initial (TI)
DFHTC TYPE=READ	Read continue (TT)	Write continue (TT)
DFHTC TYPE=WRITE (note 2) (note 3)	Read interrupt (RVI) Read continue (TT)	Write reset (TR), or Write continue Write reset
	Write initial (TI)	Read initial (TI)
DFHTC TYPE=WRITE	Write continue (TT)	Read continue (TT)
DFHTC TYPE=READ (note 4)	Write reset (TR) Read initial (TI)	Read continue (TT) Write initial (TI)

Notes:

1. CICS 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 extent 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 read interrupt.
4. Write reset is issued only for point-to-point terminals.

Figure 14. Summary of Programmable Terminal Actions

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 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 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 next to last 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 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. However, the data stream is not translated to a 3767 data stream but remains as that for a TWX WTTY.

SYSTEM/3

```
DFHTC TYPE=(READ[,SAVE])
```

```
DFHTC TYPE=(WRITE[,WAIT][,SAVE]
      [,TRANSPARENT])
      [,DEST={symb-addr|YES}]
      [,ENDMSG=NO]

DEST is TCAM only
```

```
DFHTC TYPE={DISCONNECT|RESET}
```

TYPE=DISCONNECT applies to switched line System/3s only.

SYSTEM/370

Support and macro syntax as for System/3.

SYSTEM/7

```
DFHTC TYPE=(READ[,WAIT][,SAVE]
      [,{TRANSPARENT|PSEUDOBIN}])
```

```
DFHTC TYPE=(WRITE[,WAIT][,SAVE]
      [,{TRANSPARENT|PSEUDOBIN}])
      [,DEST={symb-addr|YES}]

DEST is TCAM only
```

CICS treats the System/7 as any other programmable terminal. Transactions are normally initiated from the System/7 by issuing a four-character transaction code which is sent in BCD mode. Pseudobinary mode can be used only while communicating with an active CICS 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 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. Therefore, CICS application programs for the multipoint System/7 should be designed with the shortest possible execution time.

It is an MSP/7 standard that the first word (two characters) of every message received by the System/7 be an identification word. However, all identification words beginning with "@" (X'20') are reserved by CICS for future use.

When the PSEUDOBIN parameter is specified as part of an input request (for example, DFHTC TYPE=(READ,PSEUDOBIN)), the length of the TIOA provided by the application program must be at least twice that of the data to be read. If for example, twenty System/7 words (40 bytes) are to be read, the data area of the TIOA must be at least 80 bytes in length.

When the PSEUDOBIN parameter is specified as part of an output request, terminal control always obtains a new TIOA and frees the old TIOA unless SAVE is specified. Therefore, on a DFHTC TYPE=(WRITE,READ,PSEUDOBIN) request, the application program must reload the TIOA address (from TCTTEDA) to access the input data from the System/7.

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 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 initiates the connection to a dial-up System/7, CICS 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 (host) system console. This is normal if the task initiated by CICS is to IPL the System/7. Although the data check message is printed, CICS 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 (that is, by an automatically initiated transaction).

2260 DISPLAY STATION

```
DFHTC TYPE=({READ|READL}[,WAIT]
          [,SAVE])
```

```
DFHTC TYPE=(WRITE[,WAIT]
          [,SAVE][,ERASE])
          [,DEST={symb-addr|YES}]

DEST is TCAM only
```

2265 DISPLAY STATION

Support and macro syntax as for 2260 Display Station except that the hexadecimal equivalent of a line number can be in the range 1 through 15 (F0 through FE).

2740 COMMUNICATION TERMINAL

```
DFHTC TYPE=(READ[,WAIT])
```

```
DFHTC TYPE=(WRITE[,WAIT][,SAVE])
          [,DEST=symb-addr|YES}]

DEST is TCAM only
```

2741 COMMUNICATION TERMINAL

```
DFHTC TYPE=(READ[,WAIT])
          ,RDATT=symb-addr
```

```
DFHTC TYPE=(WRITE[,WAIT][,SAVE])
          ,WRBRK=symb-addr
          [,DEST=symb-addr|YES}]

DEST is TCAM only
```

If 2741 read attention support is included by the system programmer at system generation, a 2741 terminal operator can signal Read Attention by pressing the ATTN key after typing a message. To provide for this, the application programmer must issue a

```
  DFHTC TYPE=READ,RDATT=symb-addr
```

macro, where symb-addr is the label of a routine to which control is passed if the terminal operator terminates the input by pressing the ATTN key. (See "Read Attention" below.)

If 2741 write break support is included by the system programmer at system generation, a 2741 terminal operator can terminate the receipt of a message by pressing the ATTN key. To provide for this, the application programmer must issue a

```
  DFHTC TYPE=WRITE,WRBRK=symb-addr
```

macro, where symb-addr is the label of a routine to which control is passed if the terminal operator presses the ATTN key while a message is being received. (Write Break support, described below, is not available under CICS/DOS/VS.)

Read Attention support may be generated in any CICS/OS/VS or CICS/DOS/VS system to permit a response to the terminal operator pressing the ATTN key (rather than the return key) after typing a message, or without typing a message if no data is to be entered. Write Break support may be generated in any CICS/OS/VS system to permit a response to the terminal operator pressing the ATTN key while receiving a message. The following features must be installed on the 2741:

- For Read Attention: Transmit Interrupt (7900).
- For Write Break: Receive Interrupt (4708).

READ ATTENTION

If the terminal operator presses the attention key 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 DFHTC TYPE=READ,RDATT=symb-addr macro (which has an implied WAIT).

When this occurs, control is transferred to a CICS 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 some 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 DFHTC TYPE=READ macro. The return is made whenever one of the following occurs:

- The exit routine issues no more requests for input.
- The exit routine issues a DFHTC TYPE=READ macro 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 READ.)

If the terminal operator presses the attention key during a read, it is recognized as a read attention only if read attention support is generated and if the RDATT operand is included in the DFHTC macro requesting the input. If either or both of these conditions do not exist, the "attention" is treated as a normal read completion, that is, as if the return key had been pressed.

WRITE BREAK (CICS/OS/VS ONLY)

If the terminal operator presses the attention key 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.
- The write was initiated by a DFHTC TYPE=WRITE,WRBRK=symb-addr macro (which has an implied WAIT).

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 in the DFHTC TYPE=WRITE macro.

If the attention key is pressed and the Write Break feature is generated in CICS/OS/VS, but the DFHTC TYPE=WRITE macro does not have the WRBRK=symb-addr operand, the write break is treated as an I/O error. The same is true if the attention key is pressed, but the Write Break feature is not generated in CICS/OS/VS. A write can be interrupted only if both conditions identified above are satisfied.

Note: TYPE=WAIT and/or SAVE can be coded with READ and/or WRITE, but only RDATT or WRBRK (not both) can be specified in one DFHTC macro.

2770 DATA COMMUNICATION SYSTEM

Support and macro syntax as for System/3. The 2770 Data Communication System 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 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 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 macros. Therefore, the user should provide the appropriate screen control characters in the TIOA.

For 2770 input, data is deblocked to ETX, ETB, RS, and US characters. These characters are moved with the data to the TIOA but are not included in the data length (TIOATDL). The application programmer should be aware that such characters as NL, CR, and LF are passed in the TIOA as data.

2780 DATA TRANSMISSION TERMINAL

Support and macro syntax as for System/3. The 2780 Data Transmission Terminal 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 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 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 programmer insert the 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 TIOA but are not included in the data length (TIOATDL). The application programmer should be aware that such characters as NL, CR, and LF are passed in the TIOA as data.

2980 GENERAL BANKING TERMINAL

```
DFHTC TYPE=(READ[,WAIT][,SAVE])
```

```
DFHTC TYPE={CBUFF|PASSBK}
     [DEST={symb-addr|YES}]

DEST is TCAM only
```

PASSBOOK CONTROL

Two one-byte fields of the terminal control table terminal entry (TCTTE) may be interrogated by an application program servicing passbook requests from the 2980. These fields are:

- TCTTETAB, which contains the binary representation of the number of tabs necessary to position the print element to the correct passbook area.
- TCTTEPCF, which contains the indicators (flags) necessary for passbook control operations. The indicators TCTTEPCR and TCTTEPCW indicate whether or not the passbook is present on a read or a write operation, respectively. The same indicators are used to show the presence of the Auditor key on the 2980 Model 2.

By testing indicators TCTTEPCR and TCTTEPCW, the application program can maintain positive control with regard to the absence or presence of a passbook during an update operation. Care must, however, be taken not to alter these indicators, otherwise unpredictable results may occur.

If the passbook is present on a read (entry) operation, the TCTTEPCR indicator is turned on (set to a binary one) by CICS. In this case, the application program generally issues a write operation back to the passbook area to update the passbook. After the write operation, the application program must check the TCTTEPCW indicator to ensure that the passbook was present at the time the write occurred. If the TCTTEPCW indicator is off (set to a binary zero), the passbook was not present and the write operation did not occur. The data sent to the terminal (and not printed because of the "no passbook" condition) is, however, returned to the application program in its original form for subsequent retransmission.

When the "no passbook" condition occurs on a write, CICS allows an immediate write to the terminal. The application program should write 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 automatically causes the transaction to wait 23.5 seconds before continuing.

After regaining control from CICS following the writing of the error message, the application program can attempt another write to the passbook area 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. (The correct number of tabs can be acquired from TCTTETAB.)

If the TCTTEPCW indicator is off following 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").

In summary, all writes to the passbook area are conditional on a passbook being present before a write can be executed successfully. Therefore, a read operation cannot be combined with a passbook write. h a passbook write. For example, a DFHTC TYPE=(WRITE,READ,WAIT) macro is an invalid request for 2980 terminal services involving the passbook area. A DFHFC TYPE=PASSBK macro is permissible because it implies only WRITE,WAIT.

Note: The application programmer should not insert shift characters in output data, because this is done automatically by CICS. CICS removes shift characters from input data.

SEGMENTED WRITES CONTROL

Segmented writes are supported for both the journal area and the passbook area. Journal area segmented writes are limited in length by the hexadecimal halfword value that the user stores in TIOATDL. Passbook segmented writes are limited to a one-line logical write to ensure positive control when spacing (indexing) past the bottom of the passbook.

For example, consider a 2972 buffer length of 48 and a 2980 Model 4 logical write (print) area of 100 characters per line. The application program can write a logical record (DFHTC TYPE=PASSBK) of 100 characters to this area; CICS automatically segments the record to adjust to the buffer size. The application program must insert the passbook indexing character (X'25') as the **last** character written in one logical write to the passbook area. This is done to control passbook indexing and thereby achieve positive control of passbook presence.

If the message contains embedded passbook index characters and segmentation is necessary because of the logical length of the message, the write terminates if the passbook spaces beyond the bottom of the passbook; the remaining segments are not printed.

DATA HANDLING

SHIFT CHARACTERS: Shift characters are handled by the terminal control program and are of no concern to the application programmer. They are stripped from input messages and added to output messages as required. Data can be written in any mix of uppercase, lowercase, or special characters. (See the 2980 Translate Tables in Appendix D.)

JOURNAL INDEXING: Journal indexing is the responsibility of the application programmer. Carriage returns (X'15') may be inserted anywhere in the logical message. For further information, see the appropriate SNA Guide.

PASSBOOK INDEXING: Passbook indexing necessitates special consideration by the application programmer to control bottom-line printing on the passbook. (See "Passbook Control" and "Segmented Writes Control"; the two preceding sections.)

TAB CHARACTERS: The tab character (X'05') is controlled by the application programmer. As stated above, the number of tabs required to position the print element to the first position of the passbook is available at TCTTETAB. This value is specified by the system programmer when generating the terminal control table and may be unique to each terminal. Other tab characters are inserted as needed to control output format.

MISCELLANEOUS CHARACTERS: Turn page, message light, open chute, and special banking characters can be used by the application programmer as needed. (See the 2980 Translate Tables in Appendix D.)

AUDITOR KEY MODEL 2: Presence of the Auditor key is controlled through use of the DFHTC TYPE=PASSBK macro and may be used in a manner similar to that for passbook control. (See "Passbook Control", earlier in the Chapter.)

2980 MODEL NUMBER: TCTTETM contains the 2980 model number expressed as a hexadecimal value (X'01', X'02', X'04'). Since CICS uses the model number to select the correct translate table for each of the 2980 models, the application program should not alter this field.

COMMON BUFFER: Common buffer writes (DFHTC TYPE=CBUFF) are translated to the receiving TCTTE model character set. 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.

EXAMPLE OF APPLICATION PROGRAM FOR THE 2980

The following examples show how the various facilities described above for the 2980 are used.

In the following COBOL example, the structure DFH2980 is copied into the working storage section.

The application program is also expected to test the TCTTEPCF field to determine whether a passbook is present on a read or write. TCTTEPCR and TCTTEPCW are located in DFH2980 to aid in testing.

To test the TCTTEPCF field, statements such as the following might be used:

```
MOVE TCTTEPCF TO HOLDPCF.
IF HOLDPCFB=(HOLDPCFB/TCTTEPCW)*TCTTEPCW
THEN GO TO BOOK-FOR-PRESENT-WRITE.
```

Substituting TCTTEPCR for TCTTEPCW tests for the presence of a passbook on a read. (HOLDPCF and HOLDPCFB are also part of DFH2980.)

```
DATA DIVISION
WORKING STORAGE SECTION.
01 DFH2980 COPY DFH2980.
      .
      .
LINKAGE SECTION.
01 DFHBLLDS COPY DFHBLLDS.
   02 TCTTEAR PIC S9(8) COMP.
   02 TIOABAR PIC S9(8) COMP.
      .
      .
01 DFHTCTTE COPY DFHTCTTE.
01 DFHTIOA COPY DFHTIOA.
   02 DATA PIC X(20).
   02 FILLER REDEFINES DATA.
      03 TAB1-1 PIC X.
      03 DATA1 PIC X(19).
   02 FILLER REDEFINES DATA.
      03 TAB1-2 PIC X.
      03 TAB2-2 PIC X.
      03 DATA2 PIC X(18).
      .
      .
PROCEDURE DIVISION.
      .
      .
IF TCTTETAB = TAB-ONE GO TO ONETBCH.
IF TCTTETAB = TAB-TWO GO TO TWOTBCH.
      .
      .
ONETBCH.
    MOVE TABCHAR TO TAB1-1.
    MOVE TOTAL TO DATA1.
      .
      .
TWOTBCH.
    MOVE TABCHAR TO TAB1-2, TAB2-2.
    MOVE TOTAL TO DATA2.
      .
      .
```

In the following PL/I example, DFH2980 is included following the %INCLUDE statement for the based structures. DFH2980 contains constants that may be used when writing application programs for the 2980. To test the TCTTEPCF field, statements such as the following might be used:

```
IF (TCTTEPCF|TCTTEPCW)
THEN GO TO BOOK_PRESENT_WRITE;
```

Substituting TCTTEPCR for TCTTEPCW tests for the presence of a passbook on a read.

```
%INCLUDE DFHTIOA;
    2 DATA CHAR(20);
DCL 1 USERTIOA_1 BASED(TIOABAR),
    2 TIOAFILL CHAR(12),
    2 TAB1_1 CHAR(1),
    2 DATA1 CHAR(19);
DCL 1 USERTIOA_2 BASED(TIOABAR),
    2 TIOAFILL CHAR(12),
    2 TAB1_2 CHAR(1),
    2 TAB2_2 CHAR(1),
    2 DATA2 CHAR(18);
         .
         .
%INCLUDE DFH2980;
         .
         .
    IF (TCTTETAB=TAB_ONE)
    THEN GO TO ONETCBH;
    IF (TCTTETAB=TAB_TWO)
    THEN GO TO TWOTBCH;
         .
         .
ONETBCH: TAB1_1=TABCHAR;
         DATA1=AMOUNT;
         .
         .
TWOTBCH: TAB1_2=TABCHAR;
         TAB2_2=TABCHAR;
         DATA2=AMOUNT;
          .
          .
```

To test the station identification and to determine whether the normal station or alternate station is being used, values of the forms shown below are predefined in DFH2980:

```
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 values are one-byte character values and can be compared to TCTTESID in an IF statement.

To test the teller identification on a 2980 Model 4, the TCTTETID field is defined as a one-byte character value. It can be tested in an IF statement.

Thirty special characters are defined in DFH2980. Twenty-three of these can be referred to by the name SPECCHAR-X or SPECCHAR_X (for COBOL or PL/I) where X is an integer (1 through 23). The seven other characters are defined with names that imply their usage, for example, TABCHAR. For further information on these thirty characters, see Appendix D.

The names defined in DFH2980 for COBOL are:

```
STATION-0-N         TCTTEPCR
STATION-0-A         TCTTEPCW
STATION-1-N         TABCHAR
STATION-1-A         OPENCH
STATION-2-N         JRNLCR
STATION-2-A         PSBKCR
STATION-3-N         MSGLITE
STATION-3-A         BCKSPACE
STATION-4-N         TRNPGE
STATION-4-A         SPECCHAR-1
STATION-5-N         SPECCHAR-2
STATION-5-A         SPECCHAR-3
STATION-6-N         SPECCHAR-4
STATION-6-A         SPECCHAR-5
STATION-7-N         SPECCHAR-6
STATION-7-A         SPECCHAR-7
STATION-8-N         SPECCHAR-8
STATION-8-A         SPECCHAR-9
STATION-9-N         SPECCHAR-10
STATION-9-A         SPECCHAR-11
TAB-ZERO            SPECCHAR-12
TAB-ONE             SPECCHAR-13
TAB-TWO             SPECCHAT-14
TAB-THREE           SPECCHAR-15
TAB-FOUR            SPECCHAR-16
TAB-FIVE            SPECCHAR-17
TAB-SIX             SPECCHAR-18
TAB-SEVEN           SPECCHAR-19
TAB-EIGHT           SPECCHAR-20
TAB-NINE            SPECCHAR-21
HOLDPCFB            SPECCHAR-22
DFHFILL             SPECCHAR-23
HOLDPCF
```

The names defined in DFH2980 for PL/I are the same, except the underline character is used in place of the hyphen, and the names HOLDPCFB, DFHFILL, and HOLDPCF are not defined for PL/I.

3270 INFORMATION DISPLAY SYSTEM (BTAM AND TCAM)

```
DFHTC
     TYPE=({READ|READB}[,WAIT]
           [,SAVE][,TEXT])

READB not available under TCAM
```

```
DFHTC
    TYPE=({WRITE|COPY|PRINT|ERASEAUP}
   [,WAIT][,SAVE][,ERASE],[STRFIELD])
   [,CTLCHAR={hex-number|YES}]
   [,DEST={symb-addr|YES}]

COPY and PRINT not TCAM

DEST is TCAM only
```

When input is to be received from a terminal of the 3270 Information Display System, the application programmer can use

```
DFHTC TYPE=(READ,TEXT)
or
DFHTC TYPE=TEXT
DFHTC TYPE=READ
DFHTC TYPE=WAIT
```

to request a temporary override of the uppercase translation features of CICS, thus allowing a message containing both uppercase and lowercase data to be received from a terminal.

If the 3270 print request facility is included in the terminal control program at CICS system initialization, the application program can issue a DFHTC TYPE=PRINT to cause the data currently displayed on a 3270 display to be printed on the first available eligible 3270 printer.

For a printer to be available for printing from a display, it must be in service and not currently attached to a task. For it 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 had FEATURE=PRINT specified for it in the TCT by the system programmer.

If the 3270 display is a 3275 with an attached printer, and FEATURE=PRTADAPT has been specified in the TCTTE; the data will be printed on the attached printer.

Some 3270 displays have the facility to copy a screen image to a printer that is attached to the same control unit, without host intervention. This is a hardware facility, and is not under the control of CICS. For further details see "printer authorization matrix", in An Introduction to the IBM 3270 Information Display System.

For those devices with switchable screen sizes, the size of the screen that can be used and the size to be used for a particular transaction are defined at CICS system generation. These values are available to the application programmer in fields in the TCTTE. These fields are listed in Appendix C.

3270 LOGICAL UNIT

```
DFHTC TYPE=({READ|READB}
            [,WAIT][,SAVE][,TEXT])
            [,EOC=symb-addr]
```

```
DFHTC
    TYPE=({WRITE|PRINT|COPY|ERASEAUP}
   [,WAIT][,SAVE][,ERASE],[STRFIELD])
   [,CTLCHAR={hex-number|YES}]
   [,CCOMPL=NO]
   [,DEFRESP=YES]
   [,DEST={symb-addr|YES}]

DEST is TCAM only
```

In general, programming for a 3270 logical unit is the same as programming for a 3270 via BTAM, that is, the COPY, PRINT, READB, ERASE, and ERASEAUP are supported as before. The additional operand (DEFRESP) has been added to the DFHTC terminal control macro, and there are some restrictions:

- ASCII code is not supported (but, for BSC 3270, code translation can be carried out by NCP translation tables in the 3704/3705 communications controller).
- DFHTC TYPE=COPY must specify a symbolic terminal identification; a physical device address cannot be specified.

If the 3270 print request facility is included at system initialization, the DFHTC TYPE=PRINT macro will enable the data displayed on the screen to be printed on the first available printer that is eligible.

An available printer is one that is in service and that is not attached to a task.

An eligible printer is one for which the PRINTTO or ALTPRT option has been specified in the TCT.

If COPY has also been specified with these options, the printer must be attached to the same 3270 control unit as that used for the display.

If an eligible printer is unavailable, the data in the display buffer is captured, a message is sent to the master terminal operator by the terminal abnormal or node abnormal condition program (DFHZNAC) and control is passed to a user-written terminal or node error program which provides an appropriate action, for example, if the printer is already attached to a task, the user-written error program can direct the data to another printer or hold the data until the busy printer becomes available.

If the 3270 display is a 3275 with an attached printer, and FEATURE=PRTADAPT has been specified in the TCTTE; the data will be printed on the attached printer.

Some 3270 displays have the facility to copy a screen image to a printer that is attached to the same control unit, without host intervention. This is a hardware facility, and is not under the control of CICS. For further details see "printer authorization matrix", in An Introduction to the IBM 3270 Information Display System.

For those devices with switchable screen sizes, the size of the screen that can be used and the size to be used for a particular transaction are defined at CICS system generation. These values are available to the application programmer in fields in the TCTTE. These fields are listed in Appendix C.

3270 LUTYPE2 LOGICAL UNIT

```
DFHTC TYPE=({READ|READB}
        [,WAIT][,SAVE][,TEXT])
        [,EOC=symb-addr]
```

```
DFHTC
   TYPE=({WRITE|PRINT|COPY|ERASEAUP}
  [,WAIT][,SAVE][,ERASE],[STRFIELD])
  [,CTLCHAR={hex-number|YES}]
  [,CCOMPL=NO]
  [,DEFRESP=YES]
  [,DEST={symb-addr|YES}]

DEST is TCAM only
```

```
DFHTC TYPE=SIGNAL
   {,SIGADDR=symb-addr|,WAIT=YES}
```

Logical unit type 2 (LUTYPE2) is a logical unit defined by SNA, and which accepts a 3270 display data stream.

Support and macro syntax are the same as for the 3270 logical unit except that TYPE=COPY is not supported.

Some 3270 displays have the facility to copy a screen image to a printer that is attached to the same control unit, without host intervention. This is a hardware facility, and is not under the control of CICS. For further details see "printer authorization matrix", in An Introduction to the IBM 3270 Information Display System.

For those devices with switchable screen sizes, the size of the screen that can be used and the size to be used for a particular transaction are defined at CICS system generation. These values are available to the application programmer in fields in the TCTTE. These fields are listed in Appendix C.

3270 LUTYPE3 LOGICAL UNIT

```
DFHTC TYPE=({WRITE|PRINT|ERASEAUP}
   [,WAIT][,SAVE][,ERASE][,STRFIELD])
   [,CTLCHAR={hex-number|YES}]
   [,CCOMPL=NO]
   [,DEFRESP=YES]
   [,DEST={symb-addr|YES}]

DEST is TCAM only
```

```
DFHTC TYPE=SIGNAL
   {,SIGADDR=symb-addr|,WAIT=YES}
```

Logical unit type 3 (LUTYPE3) is a logical unit defined by SNA, and which accepts a 3270 display data stream.

Support and macro syntax are the same as for the 3270 logical unit except that TYPE=READ, READB and COPY are not supported, but TYPE=WRITE,WAIT,READ is supported for STRFIELD to issue QUERY.

3270 SCSPRT LOGICAL UNIT

```
DFHTC TYPE=(WRITE[,WAIT]
    [,SAVE][,LAST])
    [,CCOMPL=NO]
    [,DEFRESP=YES]
    [,DEST={symb-addr|YES}]

DEST is TCAM only
```

```
DFHTC TYPE=(READ[,WAIT][,SAVE])
        [,EOC=symb-addr]
```

```
DFHTC TYPE=SIGNAL
    {,SIGADDR=symb-addr|,WAIT=YES}
```

The SCS printer logical unit (SCSPRT) accepts an SCS data stream. SCS is defined by SNA. Certain devices connected as SCSPRT have input capability (for example, PA keys on 3287), in which case TYPE=SIGNAL should be used to detect operator input, followed by TYPE=READ to obtain the input. Alternatively, TYPE=(READ,WAIT) can be issued alone, in which case the program will wait for operator input.

3600 FINANCE COMMUNICATION SYSTEM (BTAM)

```
DFHTC TYPE=(READ[,WAIT][,SAVE])
```

```
DFHTC TYPE=(WRITE[,WAIT][,SAVE]
          [,TRANSPARENT])
```

INPUT

The unit of transmission from a 3601 to CICS is a segment consisting of the start-of-text data link control character (STX), the one byte identification of the 3600 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 TIOA passed to the user-written application program consists of the data only. The one-byte field TCTTEDLM contains flags describing the data-link control character (ETB, ETX, or IRS) that ended the segment. The application program can issue terminal control macros to read the data until it receives a segment ending with ETX. If blocked data is transmitted, it is received by CICS 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 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 nontransparent. Blocked input is nontransparent.

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 terminal control write, 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 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. 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.

The terminal control program sends data in transparent mode when the user-written application program issues a DFHTC TYPE=TRANSPARENT macro. Otherwise, data is sent in nontransparent mode.

CICS system output messages begin with "DFH" followed by a four-byte message number and the message text. These messages are sent in nontransparent mode. It is suggested that CICS 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 can be used by the 3600 application program to define a convention to associate responses received from CICS with transactions sent to the host, for example, sequence numbers could be used.

If a CICS user-written application program has already issued a terminal control write 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 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 in transparent mode. Otherwise it is sent in nontransparent mode.

3600 (3601) LOGICAL UNIT

```
DFHTC TYPE=(READ[,WAIT][,SAVE])
          [,EOC=symb-addr]
          [,INBFMH=symb-addr]
```

```
DFHTC TYPE=(WRITE[,WAIT][SAVE][LAST])
      [,LDC={mnemonic|YES}]
      [,FMH={NO|YES}]
      [,CCOMPL=NO]
      [,DEFRESP=YES]
      [,DEST={symb-addr|YES}]

DEST is TCAM only
```

```
DFHTC TYPE=SIGNAL
    {,SIGADDR=symb-addr|,WAIT=YES}
```

3600 PIPELINE LOGICAL UNIT

```
DFHTC TYPE=(WRITE[,WAIT][,SAVE]
               [,LAST])
```

3600 (3614) LOGICAL UNIT

```
DFHTC TYPE=(READ[,WAIT][,SAVE])
```

```
DFHTC TYPE=(WRITE[,WAIT][,SAVE])
```

3630 PLANT COMMUNICATION SYSTEM

The 3630 Plant Communication System is supported as a 3600. Two types of logical unit can be defined for a 3630: the 3600 (3601) logical unit and the 3600 pipeline logical unit. The macro syntax is as shown above for these logical units.

3650 HOST COMMAND PROCESSOR LOGICAL UNIT

```
DFHTC TYPE=(READ[,WAIT][,SAVE])
          [,EOC=symb-addr]
```

```
DFHTC TYPE=(WRITE[,WAIT][,SAVE])
          [,FMH=YES]
          [,CCOMPL=NO]
```

3650 HOST CONVERSATIONAL (3270) LOGICAL UNIT

```
DFHTC TYPE=(READ[,WAIT][,SAVE])
         [,EOC=symb-addr]
```

```
DFHTC TYPE=({WRITE|PRINT|ERASEAUP}
      [,WAIT][,SAVE][,ERASE][,LAST])
      [,CTLCHAR={hex-number|YES}]
      [,CCOMPL=NO]
      [,DEFRESP=YES]
      [,FMH=YES]
```

OUTPUT DEVICE CONTROL

Device control characters for 3650 devices can be inserted by CICS application programs into output data streams. To avoid designing such device-dependent CICS application programs, device responsibility can be moved to the 3650 application programs. Thus, the CICS application programs would be concerned with data content, while data format would be the responsibility of the 3650 application program.

Another alternative is available for handling device-dependent matters. Basic mapping support (BMS) can be used to write data to logical units (except for pipeline). BMS can be used to format data and insert the necessary 3650 device control characters.

THE ERASE FUNCTION

The erase option is supported by the DFHTC macro when this macro is issued for a host conversational (3270) logical unit. The erase function for this logical unit is controlled as a device-dependent character. The erase function can be obtained using BMS.

3650 PIPELINE LOGICAL UNIT

```
DFHTC TYPE=(WRITE[,WAIT][,SAVE]
                [,LAST])
```

3650 HOST CONVERSATIONAL (3653) LOGICAL UNIT

```
DFHTC TYPE=(READ[,WAIT][,SAVE])
          [,EOC=symb-addr]
```

```
DFHTC TYPE=(WRITE[,WAIT][,SAVE]
          [,LAST])
          [,CCOMPL=NO]
          [,DEFRESP=YES]
```

3650 INTERPRETER LOGICAL UNIT

```
DFHTC TYPE=PROGRAM
       ,PRGNAME=name
      [,VALID=address]
      [,NONVAL=address]
      [,CONNECT={ACTIVATE|CONVERSE}]
      [,NORESP=address]
```

```
DFHTC TYPE=(READ[,WAIT][,SAVE])
      [,EODS=symb-addr]
      [,EOC=symb-addr]
      [,INBFMH=symb-addr]
```

```
DFHTC TYPE=(WRITE[,WAIT]
      [,SAVE][,LAST])
      [,FMH={YES|NO}
      [,DEFRESP=YES]
```

```
DFHTC TYPE=EODS       VTAM only
```

3660 SUPERMARKET SCANNING SYSTEM (BTAM)

Support and macro syntax as for System/3, except that the 3660 cannot initiate communications; the host system initiates all transactions.

3735 PROGRAMMABLE BUFFERED TERMINAL

```
DFHTC TYPE=(READ[,WAIT][,SAVE])
          [,EOF=symb-addr]
```

```
DFHTC TYPE=(WRITE[,WAIT]
   [,SAVE][,NOTRANSLATE])
   [,DEST={symb-addr|YES}]

DEST is TCAM only
```

The 3735 Programmable Buffered Terminal may be serviced by CICS in response to terminal-initiated input, or as a result of an automatic or time-initiated transaction. Both are explained below.

AUTOANSWER

The 3735 transaction is attached by CICS upon receipt of input from a 3735. Data is passed to the application program in 476-byte blocks; each block (one buffer) may contain multiple 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 multiple logical records, the application program must perform any necessary deblocking functions and the gathering of partial logical records from consecutive reads.

It is recommended that the user spool input data from a 3735 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 EOF=symb-addr operand must be used with each DFHTC TYPE=READ request. When the EOF branch is taken, the user may begin to write FDPs or data to the 3735, or, optionally, request CICS to disconnect the line.

It is possible that the 3735 will transmit the end-of-file condition immediately upon connection of the line. For this reason the user must code the initialization request (DFHTC EOF=symb-addr) before issuing any other terminal control requests.

The user is responsible for formatting 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 NOTRANSLATE operand must be included in the DFHTC TYPE=WRITE request for each block of FDP records.

The user must issue a DFHTC TYPE=DISCONNECT macro when all output has been transmitted to the 3735. If the application program ends during batch write mode prior to issuing the DISCONNECT request, CICS forces a 3735 "receive abort" condition and all data just transmitted is ignored by the 3735.

AUTOCALL AND TIME-INITIATED

In automatic and time-initiated transactions, all considerations stated above except use of the DFHTC EOF=symb-addr macro apply when CICS dials a 3735. The DFHTC EOF=symb-addr macro is not used.

CICS connects the line and allows the user to indicate the direction of data transfer by means of the first terminal control request. If this first request is a WRITE and the 3735 has data to send, the 3735 causes the line to be disconnected.

3740 DATA ENTRY SYSTEM

```
DFHTC TYPE=(READ[,WAIT][,SAVE])
      [,ENDFILE=symb-addr]   not TCAM
      [,ENDINPT=symb-addr]   not TCAM
```

```
DFHTC TYPE=(WRITE[,WAIT][,SAVE]
 [,ENDFILE][,ENDOUTPUT]
 [,TRANSPARENT])
 [,DEST={symb-addr|YES}]

DEST is TCAM only
```

The 3740 Data Entry System may be serviced by CICS as a batch or inquiry mode application. Considerations for both modes are described in the following paragraphs.

BATCH MODE APPLICATIONS

In batch mode, the 3740 sends multiple files of data to CICS during a single transmission. All input data files must be sent to CICS before the 3740 is able to receive data from CICS. When able to receive, the 3740 accepts multiple files of data in a single transmission. To communicate in this manner, a means is provided in the DFHTC macro for identifying end-of-file, end-of-input, and end-of-output conditions.

When sending data to the 3740, the DFHTC TYPE=ENDFILE macro must be issued after each file to signal the end-of-file (EXT) condition to the 3740. The DFHTC TYPE=ENDOUTPUT macro should be issued after all data has been sent to the 3740 (EOT) and must be immediately preceded by a DFHTC TYPE=ENDFILE macro. Once end-of-output is signaled in this manner, no additional WRITEs should be issued. The WRITE, ENDFILE, and ENDOUTPUT operands may be combined in the DFHTC macro. For example, a DFHTC TYPE=(WRITE,ENDFILE) causes a write operation followed by an end-of-file signal.

A DFHTC TYPE=(WRITE,ENDFILE,ENDOUTPUT) causes a write operation, an end-of-file signal, and then an end-of-output signal. A DFHTC TYPE=(ENDFILE, ENDOUTPUT) causes an end-of-file signal followed by an end-of-output signal. The placement of the operand within the macro has no effect on the sequence.

Note: If ENDFILE is combined with any other operand and SAVE is also present, the TIOA used to write the end-of-file record will be the current TIOA after return from terminal control.

3767 INTERACTIVE LOGICAL UNIT

```
DFHTC TYPE=(READ[,WAIT][,SAVE])
        [,EOC=symb-addr]
```

```
DFHTC TYPE=(WRITE[,WAIT]
     [,SAVE][,LAST])
     [,FORCE=YES]
     [,CCOMPL=NO]
     [,DEFRESP=YES]
     [,DEST={symb-addr|YES}]

DEST is TCAM only
```

```
DFHTC TYPE=SIGNAL
     {,SIGADDR=symb-addr|,WAIT=YES}
```

3770 INTERACTIVE LOGICAL UNIT

```
DFHTC TYPE=(READ[,WAIT][,SAVE])
        [,EOC=symb-addr]
```

```
DFHTC TYPE=(WRITE[,WAIT]
      [,SAVE][,LAST])
      [,FORCE=YES]
      [,CCOMPL=NO]
      [,DEFRESP=YES]
      [,DEST={symb-addr|YES}]

DEST is TCAM only
```

```
DFHTC TYPE=SIGNAL
      {,SIGADDR=symb-addr|,WAIT=YES}
```

3770 BATCH AND BATCH DATA INTERCHANGE LOGICAL UNIT

```
DFHTC TYPE=(READ[,WAIT][,SAVE])
      [,EODS=symb-addr]
      [,EOC=symb-addr]
      [,INBFMH=symb-addr]
```

```
DFHTC TYPE=(WRITE[,WAIT]
      [,SAVE][,LAST])
      [,FMH={NO|YES}]
      [,CCOMPL=NO]
      [,DEFRESP=YES]
      [,DEST={symb-addr|YES}]

DEST is TCAM only
```

3770 FULL FUNCTION LOGICAL UNIT

```
DFHTC TYPE=(READ[,WAIT][,SAVE])
      [,EOC=symb-addr]
      [,INBFMH=symb-addr]
```

```
DFHTC TYPE=(WRITE[,WAIT]
     [,SAVE][,LAST])
     [,FMH={NO|YES}]
     [,CCOMPL=NO]
     [,DEFRESP=YES]
     [,DEST={symb-addr|YES}]

DEST is TCAM only
```

3780 DATA COMMUNICATIONS TERMINAL

Support and macro syntax as for System/3.

3790 INQUIRY LOGICAL UNIT

```
DFHTC TYPE=(READ[,WAIT][,SAVE])
      [,EOC=symb-addr]
```

```
DFHTC TYPE=(WRITE[,WAIT]
      [,SAVE][,LAST])
      [,FMH={NO|YES}]
      [,CCOMPL=NO]
      [,DEFRESP=YES]
      [,DEST={symb-addr|YES}]

DEST is TCAM only
```

3790 FULL FUNCTION LOGICAL UNIT

```
DFHTC TYPE=(READ[,WAIT][,SAVE])
      [,EOC=symb-addr]
      [,INBFMH=symb-addr]
```

```
DFHTC TYPE=(WRITE[,WAIT]
      [,SAVE][,LAST])
      [,FMH={NO|YES}]
      [,CCOMPL=NO]
      [,DEFRESP=YES]
      [,DEST={symb-addr|YES}]

DEST is TCAM only
```

3790 (SCS PRINTER) LOGICAL UNIT

```
DFHTC TYPE=(WRITE[,WAIT]
      [,SAVE][,LAST])
      [,CCOMPL=NO]
      [,DEFRESP=YES]
      [,DEST={symb-addr|YES}]

DEST is TCAM only
```

3790 (3270-DISPLAY) AND 3790 (3270-PRINTER) LOGICAL UNITS

These logical units are sometimes referred to collectively as the 3270 compatibility logical unit. Support and macro syntax are the same as for the 3270 logical unit, apart from the following exceptions:

- DFHTC TYPE=READB is not supported for the 3270-printer logical unit.
- DFHTC TYPE=COPY is not supported for the 3270-display logical unit.
- When using the DFHTC TYPE=PRINT macro, if FEATURE=PTRADAPT has been specified in the TCT, allocation of the printer is controlled by the 3790. If FEATURE=PTRADAPT has not been specified, allocation of printers is governed by the PRINTTO and ALTPRT options specified in the TCT.

3790 BATCH DATA INTERCHANGE LOGICAL UNIT

Support and macro syntax as for 3770 Batch Logical Unit.

7770 AUDIO RESPONSE UNIT

```
DFHTC TYPE=(READ[,WAIT][,SAVE])
```

```
DFHTC TYPE=(WRITE[,WAIT][,SAVE])
```

Although CICS does not distinguish between special codes (characters) entered at an audio terminal (for example, the 2721 Portable Audio Terminal), an application program is not precluded from performing special functions upon encountering these codes. For example, the following special hexadecimal codes may be entered from a 2721:

Key	Code
CALL END	37 (see note)
CNCL	18
#	3B (see note) or 7B
VERIFY	2D
RPT	3D
EXEC	26 (see note)
F1	B1
F2	B2
F3	B3
F4	B4
F5	B5
00	A0
000	3B (see note) or B0
IDENT	11, 12, 13, or 14 plus two other characters

Note: These codes cause a hardware interrupt and are in the terminal input/output area (TIOA) immediately following the data; the codes are not included in the data length.

For further information concerning the 2721, see the publication 2721 Portable Audio Terminal Component Description.

The following special hexadecimal codes may be entered from a Touch-Tone telephone. (Touch-Tone is the trademark of the American Telephone and Telegraph Company.)

Key	Code
*	A0
#	3B or B0

The * and # characters of a Touch-Tone telephone correspond to the 00 and 000 characters, respectively, on a 2721 Portable Audio Terminal. The # and 000 characters cause an end-of-inquiry (EOI) hardware interrupt (X'3B') unless the EOI Disable feature (#3540) is installed on the 7770 Audio Response Unit Model 3. If this feature is installed, the user can elect that neither, or only one, of the # and 000 characters will cause a hardware interrupt. At the option of the user, either or both of the # and 000 characters do not cause a hardware interrupt, are presented in the TIOA with the rest of the data, and are included in the data length.

If, after receiving at least one character from a terminal, no other characters have been received by the 7770 for a period of five seconds, the 7770 automatically generates an EOI hardware interrupt that ends the read operation.

LUTYPE4 LOGICAL UNIT

```
DFHTC TYPE=(READ[,WAIT][,SAVE])
      [,EODS=symb-addr]
      [,EOC=symb-addr]
      [,INBFMH=symb-addr]
```

```
DFHTC TYPE=(WRITE[,WAIT]
      [,SAVE][,LAST])
      [,FMH={NO|YES}]
      [,CCOMPL=NO]
      [,DEFRESP=YES]
```

```
DFHTC TYPE=SIGNAL
      {,SIGADDR=symb-addr|,WAIT=YES}
```

The TYPE=SIGNAL macro is required to detect a hard request change direction (RCD) signal from the terminal. The application program should not issue a TYPE=WRITE macro following such a signal.

LUTYPE4 terminals can operate in unattended mode. The application programmer can detect unattended mode by testing the TCTTE field TCTEMOP under the mask TCTEMOPU.

OTHER CICS-SUPPORTED TERMINALS

```
DFHTC TYPE=(READ[,WAIT][,SAVE])
DFHTC TYPE=(WRITE[,WAIT][,SAVE])
  [,DEST={symb-addr|YES}]

DEST is TCAM only
```

TCAM SUPPORTED LOGICAL UNITS (CICS/OS/VS ONLY)

```
DFHTC TYPE=({READ|READL}[,WAIT]
      [,SAVE])
      [,INBFMH=symb-addr]
```

```
DFHTC TYPE=(WRITE[,WAIT]
      [,SAVE][,ERASE][,LAST]
      [,ERASEAUP])
      [,FMH={NO|YES}]
      [,CTLCHAR={hex number|YES}]
      [,LINEADR={number|YES}]
      [,DEST={symb-addr|YES}]
```

OPERANDS OF DFHTC MACRO

CCOMPL=NO

- Used with VTAM logical units only.

Indicates that the last request/response unit (RU) sent as a result of this write request will not complete the chain. If this operand is omitted, the last RU will terminate the chain.

Before this operand may be used, the system programmer must have specified that the application program may control outbound chaining indicators by coding a DFHPCT TYPE=OPTGRP macro with the MSGPOPT=CCONTRL operand. If CCOMPL=NO is used without this support, the task will be abnormally terminated.

There are a number of restrictions on the use of the CCOMPL=NO operand; these restrictions are as follows:

- If CCOMPL=NO is used without the authority (CCONTR) of the system programmer, the task will be abnormally terminated.
- CCOMPL=NO cannot be used if the DEFRESP=YES operand is specified.
- If CCOMPL=NO is specified, the application program must not issue a read request until a write request that does not specify CCOMPL=NO has been issued; failure to observe this restriction will lead to abnormal termination of the task.
- CCOMPL=NO is not valid for a combined write and read request, including conversational write operations. TYPE=LAST is ignored if it is not FOC or OC.

CONNECT=

- Used with 3650 interpreter logical units only.

This operand specifies the type of connection to be established.

ACTIVATE
specifies that the 3650 application program will not communicate with the host processor.

CONVERSE
specifies that the 3650 application program will communicate with the host processor.

CTLCHAR=

- Used for 3270 logical units, 3650 host-conversational (3270) logical units, 3790(3270-display), and 3790(3270-printer) logical units only.

This operand is used (1) in a DFHTC TYPE=WRITE macro to provide the hexadecimal representation of the write control character (WCC) that controls the requested write operation, or (2) except for the 3650 host conversational (3270) LU, in a DFHTC TYPE=COPY macro to provide the hexadecimal representation of the copy control character (CCC) that controls and defines the copy function to be performed.

hex-number
is the hexadecimal representation of the WCC or CCC required for the operation specified in the TYPE= operand of this DFHTC macro.

YES
indicates that the appropriate bit configuration has been placed in TIOACLCR.

For DFHTC TYPE=WRITE, if the functions defined by the WCC only are to be performed (that is, no data stream is to be supplied), TIOATDL must contain zero. If the CTLCHAR operand is omitted, all modified data tags are reset to zero, and the keyboard is restored. For DFHTC TYPE=COPY, if the CTLCHAR operand is omitted, the contents of the entire buffer (including nulls) are copied and the start printer flag is not on.

DEFRESP=YES

- Used with VTAM logical units only.

Indicates that a definite response is required when the write operation has been completed. DEFRESP=YES cannot be specified if the CCOMPL=NO operand is used.

This operand specifies, for this write operation only, that a definite response is required, even if neither the MSGINTEG operand nor the PROTECT operand has been specified in the DFHPCT TYPE=OPTGRP macro by the system programmer.

DEST=
indicates that the output message is to be sent to a TCAM destination other than the source TCAM terminal.

This operand is meaningful only for TCAM-supported terminals.

symbolic name
is the symbolic address of the storage area containing the TCAM destination to which the message must be sent.

YES
indicates that the application program has placed the four-byte message destination in TCTTEDES before issuing the WRITE. This can be used to allow dynamic selection of the message destination.

ENDFILE=symb-addr

- Used for 3740 Data Entry System only.

Indicates the label of the routine that is to receive control when end-of-file is encountered on batch input. It is set when a null block is received, indicating the end of a physical file. The task must continue reading.

ENDINPT=symb-addr

- Used for 3740 Data Entry System only.

Indicates the label of the routine that is to receive control when end-of-input is reached on batch processing. It is set by CICS when an end of transmission signal is received and the ENDFILE indicator was set. After this condition the task must not issue any further reads to the device but must return to CICS so that the 3740 can be set to receive a new batch of input.

ENDMSG=NO

- Used for BTAM terminals only.

Indicates that the block sent as a result of the write request does not complete the message. If this operand is omitted, the message will be regarded as complete when the write request has been fulfilled. This operand is valid only for assembler language application programs.

Before this operand may be used, the system programmer must have specified that the application program may control outbound chaining by coding a DFHPCT TYPE=OPTGRP macro with the MSGPREQ=CCONTRL operand. If ENDMSG=NO is used without this support, the task will be abnormally terminated.

There are a number of restrictions on the use of the ENDMSG=NO operand; these restrictions are as follows:

- If ENDMSG=NO is used without the authority (MSGPREQ=CCONTRL) of the system programmer, the task will be abnormally terminated.
- If ENDMSG=NO is specified, the application program must not issue a read request until a write request that does not specify ENDMSG=NO has been issued; failure to observe this restriction will lead to abnormal termination of the task.
- ENDMSG=NO is not valid for a combined write and read request, including conversational write operations.

EOC=symb-addr

- Used for logical units only.

Specifies the label of the routine that is to receive control if the request/response unit (RU) is received with the end-of-chain (EOC) indicator set. If this operand is specified, the WAIT parameter of the TYPE operand is assumed. If an inbound FMH is received, the INBFMH operand will override this operand. If an end-of-data-set FMH is also received, the EODS operand will override both this operand and the INBFMH operand. (Overridden operands can be specified in a DFHTC TYPE=WAIT macro.)

EODS=symb-addr

- Used for 3650 interpreter logical units, batch logical units, and LUTYPE4 logical units only.
- Cannot be used for 3650 Host Command Processor logical units.

Indicates the label of a user-written routine that is to receive control if an end-of-data-set FMH is received. The TIOA contains the EODS indicators. If EODS is specified, the WAIT parameter of the TYPE operand is assumed. If EODS is specified, and end-of-data-set is received, the EOC and INBFMH operands are overridden; they can be specified in a DFHTC TYPE=WAIT macro within the end-of-data-set routine.

Symbolic address is the address to which control is to be given if the CICS EODS indicator is set on. The indicator is set when a READ is issued and there is no data remaining for this data set.

EOF=symb-addr
indicates the label of the routine that is to receive control when end-of-file is encountered on batch input. This operand can be used in a special initialization macro, DFHTC EOF=symb-addr, to test for the end-of-file condition upon initial connection to a 3735. It must be included in the initialization section of the application program that handles 3735 input, preceding other DFHTC macros.

Note: When the EOF condition occurs, TIOATDL is set to binary zeros to indicate that the TIOA for the input operation contains no valid data.

FMH=

- Used for 3600 (3601), 3650 host-conversational (3270), 3650 host-command processor, LUTYPE4, 3770 batch, 3790 full function, 3790 inquiry, and 3790 batch data interchange logical units only.

This operand indicates whether the function management header (FMH) has been placed in the TIOA by the application program. If FMH is omitted, NO is assumed.

For the 3600 (3601) and 3790 inquiry logical units, an FMH is required and is provided as described below. For the 3650 host-conversational (3270) logical unit, the FMH is required if outboard maps are to be used; the FMH in such cases can be provided by BMS, if BMS is being used, or otherwise, by the application program. For LUTYPE4 and batch logical units, the FMH is required for device selection and is provided as described below.

<u>NO</u>
indicates that the application program has not placed the FMH in the TIOA. For the 3600 (3601) and 3790 inquiry logical units, CICS is responsible for placing the FMH in the TIOA; if NO is specified, space must be reserved in the TIOA for the FMH. For the 3650 host-conversational (3270) logical unit, CICS does not build an FMH, and the data is transmitted unmodified. For all other logical units, no FMH is sent; refer to the appropriate CICS subsystem guides for details of when an FMH is necessary.

YES
indicates that the application program has placed the FMH into the TIOA. Refer to the appropriate CICS subsystem guides for size and format of the FMH for a specific terminal. The FMH=YES and LDC=YES options are mutually exclusive.

FORCE=YES

- Used for interactive logical units only.

This operand indicates that the write operation is to be preceded by an outbound SIGNAL data-flow-control command to force the terminal into receive mode. This operand is used only for interactive logical units operating in contention mode, and is ignored otherwise.

INBFMH=symb-addr
specifies the label of the routine that is to receive control if the request/response unit (RU) contains an FMH, and CICS has passed this

FMH to the application program. The presence of an inbound FMH means that, if this operand is specified, the EOC operand is overridden. If an end-of-data-set FMH is received, the EODS operand will override the INBFMH operand. (Overridden operands can be specified in a DFHTC TYPE=WAIT macro.)

For this operand to be effective, the system programmer must have specified INBFMH=ALL or EODS in the PCT entry for the transaction. If INBFMH=NO is specified, inbound FMHs will not be passed to the application program, and the INBFMH operand will never be operative.

LDC

- Used for the 3601 logical unit (but not for the 3614, even if attached to the 3601) only.

This operand specifies the mnemonic to be used by CICS to determine the logical device code (LDC) that is to be transmitted to the logical unit in the function management header.

mnemonic
is the two-character mnemonic used to determine the appropriate LDC numeric value. The mnemonic represents a LDC entry in the DFHTCT TYPE=LDC macro.

YES
indicates that the application program has placed the mnemonic in TCATPLDM. The LDC=YES and FMH=YES options are mutually exclusive.

NONVAL=address

- Used with 3650 application programs only.

This operand indicates the label of the user-coded routine to receive control if the name specified in the PRGNAME operand is invalid.

NORESP=address

- Used with 3650 logical units only.

This operand indicates the label of a user-coded routine to receive control if there is a no error response.

PRGNAME=name

- Used with 3650 logical units only.

This operand indicates the name of the 3650 application program. The name (up to eight characters) is transmitted to the 3651 for verification by the 3650 control program.

RDATT=symb-addr
indicates the label of the routine to which control is to be transferred if the read operation that responds to a DFHTC TYPE=READ macro is terminated by pressing the attention (ATTN) key rather than the return key.

Note: This operand is meaningful only if 2741 Read Attention support has been generated in the CICS system. See "Read Attention" and "Write Break" under "2741 Communication Terminal" earlier in the chapter.

SIGADDR=symb-addr

- VTAM only

Specifies the symbolic address of the routine to be given control if SIGNAL is received.

TYPE=
describes the terminal or logical unit operations required, as follows:

TYPE=CBUFF

- Used with 2980 General Banking Terminal only.

This is a stand-alone parameter used to place a message in the common buffer of the 2972 terminal control unit; the 2972 associated with the current TCTTE receives the output message. Both write and wait are implied.

Note: The output message is translated according to the model of 2980 described by the current TCTTE. If more than one model is attached to a 2972 Terminal Control Unit, the contents of the common buffer are intelligible only to the model for which the message was translated. Since shift characters are added to the message by CICS during translation, the length of the message is dependent upon the contents of the message. Up to 23 characters, including shift characters, can be transmitted.

TYPE=COPY

- Valid only for BSC-connected devices which have the copy feature, that is, BTAM remote connection, or VTAM non-SNA remote connection.

This parameter is used to copy the format and data contained in the buffer of one terminal into the buffer of another terminal attached to the same remote 3270 control unit. The terminal from which data is to be copied can be identified in either of two ways:

1. Set TIOATDL to a value of 1, and the first byte of the output data area (TIOADBA) to the physical address of the terminal to be copied; or

2. Set TIOATDL to a value of 4 and the first four bytes of the output data area (TIOADBA) to the terminal identification of the terminal to be copied. If the terminal identification is less than four bytes, it must be left-justified with blank padding on the right.

The copy control character (CCC), which controls and defines the copy function to be performed, must be supplied in the CTLCHAR operand of the DFHTC macro.

Note: For VTAM-supported 3270 logical units, it is not possible to supply the physical address of the terminal to be copied; the terminal identification must be supplied.

TYPE=DISCONNECT

- Switched lines and logical units only.

For switched lines, DISCONNECT is used to break the line connection between the terminal and the computer; if the terminal is a buffered device, the data in the buffer(s) is lost.

- CICS does not automatically disconnect a 3270 display at the end of a transaction. A disconnection occurs at the request of a terminal operator, at the request of the application program (through this macro), or after a specified number of time-outs are encountered by DFHTEP for the terminal. (Refer to the appropriate _CICS Customization Guide_ for information about DFHTEP.)

- When used with a TCAM terminal or logical unit, DISCONNECT sets the X'08' bit in the communication control byte (CCB) sent to TCAM. The message handler should provide the necessary function (that is, issue IEDHALT, to terminate the logical-unit session) for disconnect.

- When used with VTAM logical units, DISCONNECT, which does not become effective until the task has been terminated, terminates the session, without causing a physical disconnection.

TYPE=ENDFILE

- Used for 3740 Data Entry System only.

Indicates that an end-of-file record is to be written to the terminal.

TYPE=ENDOUTPUT

- Used for 3740 Data Entry System only.

Indicates that an end-of-output record is to be written to the terminal.

TYPE=EODS

- Used with 3650 interpreter logical units only.

Causes an end of data set FMH to be sent on behalf of the task. An I/O area need not be supplied by the CICS application program. Refer to the appropriate _CICS IBM 3650/3680 Guide_ for details about communicating with a 3650 application program.

Note: If the application receives the FMH, the FMH may have been presented on completion of a previous read request. The end of the data set is not until the CICS EODS indicator is set on.

TYPE=ERASE

- Used with 2260 Display Station, 3270 Information Display System, 3270 logical units, 3650 host-conversational logical units, 3790 (3270-display), and 3790 (3270-printer) logical units only.

This parameter is used with the WRITE or WRITEL operand. It blanks out the screen and sets the cursor to the upper left corner. Normally, TYPE=ERASE would be used on the first output request of a transaction to prepare the screen for new output data.

TYPE=ERASE also sets the screen size to that specified for the transaction that issues the command. Therefore when switching from one screen size to another between transactions, a TYPE=ERASE must be issued to set the screen size of a new transaction. If one is not issued, the screen size will remain unchanged from a previous transaction's setting.

The CLEAR key, if used within a transaction, sets the screen size to its default. However, CICS will reset the transaction specified size following a CLEAR operation.

Note: To erase the screen,

1. Place the address of a TIOA into TCTTEDA,
2. Place a data length of 0 into TIOADTL, and
3. Issue a DFHTC TYPE=(WRITE, ERASE) macro.

TYPE=ERASE and DEFRESP=YES are mutually exclusive.

TYPE=ERASEAUP

- Used with 3270 logical units, 3650 host-conversational (3270) logical units, 3790 (3270-display) and 3790 (3270-printer) logical units only.

This parameter issues an "erase all unprotected" command command and causes the following functions to be 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.

Neither WRITE, ERASE, nor COPY can be specified in a DFHTC macro that includes the ERASEAUP parameter. No data stream is supplied.

TYPE=LAST
signals CICS that the WRITE is the last output for a transaction and, therefore, the end of a bracket. Specifying this parameter can improve system performance for VTAM logical units except when used with the 3270 logical unit.

- This parameter has no effect when used with a 3270 logical unit.

TYPE=NOTRANSLATE
prevents translation of form description program (FDP) records which are to be transmitted to a 3735 using ASCII transmission code. (For further information, see "3735 Programmable Buffered Terminal", earlier in the chapter.)

TYPE=PASSBK

- Used with 2980 General Banking Terminal only.

This is a stand-alone parameter used to cause output to be printed on a banking passbook. Both WRITE and WAIT are implied. If a passbook is not present, no printing occurs. An error message can be sent to the operator of the terminal associated with the requesting task.

TYPE=PRINT

- Used with 3270 logical units, 3650 host-conversational (3270) logical units, 3790 (3270-display), and 3790 (3270-printer) logical units only.

This parameter specifies that the data currently displayed on a 3270 display is to be printed on an eligible 3270 printer.

TYPE=PROGRAM

- Used with 3650 devices only.

This parameter is used to request the loading of a 3650 application program. If the program is loaded, control is returned to the next sequential instruction following the DFHTC TYPE=program macro unless NORESP=program is specified. Otherwise, control is returned to an address specified by one of the other operands of the macro as listed below.

TYPE=PSEUDOBIN
indicates that the data being read is to be translated from System/7 pseudobinary representation to hexadecimal. (For more information about System/7 programming, see "System/7", earlier in the chapter.)

TYPE=READ
indicates that the data is to be read from a terminal or logical unit.

When the contents of a 3270 buffer are read the programmer should be aware that the attention identifier byte and the cursor address are made available at TCTTEAID and TCTTECAD respectively. A set of standard symbolic names for testing the 3270 attention identifier is provided in a copy book called DFHAID. For further details refer to "Standard Attention Identifier List (DFHAID)" in "Chapter 4.3. Basic Mapping Support" on page 143.

TYPE=READB

- Used with BTAM 3270 and 3270 and 3790(3270-display) logical units only.

This parameter reads the contents of the 3270 buffer, beginning at buffer location 0 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. READB cannot be specified for TCAM-supported terminals nor can it be used for 3790 (3270-printer) logical units.

Note: Because of the relatively long transmission times required to transmit the entire contents of a remote 3270 buffer, the READB parameter should be used primarily for testing and diagnosing; the COPY parameter, which permits a selective transfer of buffer contents should be used in all other cases.

TYPE=READL

- Used with 2260 only.

Indicates that the keyboard is to remain locked at the completion of a data transfer. This parameter is applicable only to CICS/OS/VS, but may be used on a CICS/DOS/VS application if compatibility with CICS/OS/VS is desired.

TYPE=RESET

- Used with binary synchronous devices only.

This operand is used to relinquish use of a communication line; the next BTAM operation will be a read or write initial. RESET is not supported by TCAM, because line control is performed by TCAM in the MCP.

TYPE=SAVE

in the case of a read operation, it indicates that the TIOA used in a previous terminal operation is not to be used as an input area; a new TIOA is acquired. For a write operation, it indicates that the TIOA whose address is in TCTTEDA is not to be released upon completion of the write operation; however, there is no guarantee that TCTTEDA will remain unchanged.

TYPE=SIGNAL

- Used with VTAM interactive and LUTYPE2, LUTYPE3, LUTYPE4 and SCSPRT logical units, and VTAM 3600 (3601) logical units, only.

Indicates that this macro specifies the action to be taken by the application program when an inbound SIGNAL data-flow-control command is received from the logical unit.

The four-byte field TCTESIDI in the terminal control table terminal entry (TCTTE) is set to the signal code received from the logical unit. If a hard request change direction (RCD) signal is received (signal code X'00010000') from an LUTYPE4 unit, the transaction should either end or read from the unit. An attempt to follow the signal with a write would be an error.

Most logical units will send a signal with a code of X'00010000' when an attention key is pressed.

TYPE=STRFIELD

- Assembler language only.

Specifies that the TIOA contains structured fields. If this operand is specified, the contents of all structured fields must be handled by the application program. (Structured fields are described in the appropriate <u>CICS IBM 3270 Data Stream Device Guide</u>.) CTLCHAR and ERASE are mutually exclusive with STRFIELD and their use will generate an MNOTE .

TYPE=TEXT

- Used with 3270 only.

Is meaningful only when used in conjunction with a READ request. It specifies a temporary override of the uppercase translation feature of CICS to allow the task to receive a message containing both uppercase and lowercase data.

TYPE=TRANSPARENT

- Applicable to System/3 when it 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).

- Applicable to System/7 when it indicates that the data being read is not to be translated.

TYPE=WAIT
ensures that the terminal or logical unit operation requested in the macro is completed before starting subsequent processing. WAIT can be coded separately from a READ to accomplish overlapping of logical unit I/O operations; or with the EOC, EODS, or INBFMH operand, for example, to give control to user-written routines from within an end-of-data-set routine entered as a result of specifying the EODS operand.

TYPE=WRITE
indicates that data is to be written to a terminal or logical unit.

VALID=address

- Used with 3650 devices only.

This operand indicates the label of a user-coded routine to receive control if the name specified in the PRGNAME operand is valid but sufficient resources are not available in the 3651 to initiate the 3650 application program. This routine can determine whether a DFHIC TYPE=INITIATE or DFHIC TYPE=PUT macro is to be issued in order to restart the 3650 application program later.

WAIT=YES
specifies that the task is to be suspended until SIGNAL is received. This request is ignored if the logical unit cannot send a SIGNAL command; the contents of field TCTESIDI will be set to X'00000000' in these circumstances.

WRBRK=symb-addr
is the symbolic address to which control is transferred if a write operation started in response to this DFHTC TYPE=WRITE macro is interrupted by the terminal operator pressing the Attention (ATTN) key.

This operand is meaningful only if 2741 Write Break support has been generated into the system, an option available only under CICS/OS/VS. See "Read Attention" and "Write Break" under "2741 Communication Terminal" earlier in the chapter.

CHAPTER 4.3. BASIC MAPPING SUPPORT

Basic mapping support (BMS) provides the CICS application programmer with various formatting services that assist in interpreting input data streams from and preparing output data streams to the terminal network. These formatting services are provided by BMS modules that act as an interface between the user's application program and the CICS terminal control program.

The application program passes data to BMS and receives data from BMS in a device-independent format. BMS macros are issued by the application program to control formatting of the data and to initiate input from and output to the terminal network.

You should refer to the appropriate CICS Application Programmer's Reference Manual (Command Level) for descriptions of the additional attributes field outlining (OUTLINE), mixed DBCS and EBCDIC fields (SOSI), and background transparency (TRANSP) that are new for this release.

ADVANTAGES OF BMS

The two principal advantages to be obtained by using BMS are device independence and format independence.

DEVICE INDEPENDENCE

Device independence permits the application program to send data to a terminal or to receive data from a terminal without regard for the physical characteristics of the terminal. BMS can be used for communication with any of the following devices and logical units:

```
1050
2740
2741
2770
2780
2980 Models 1 and 2
2980-4 (keyboard and printer only)
3270
3780
TWX
Tape storage devices
Disk storage devices
CRLP (a device declared as
     card-reader-in/line-printer-out)
TCAM-connected terminals (defined by
     TRMTYPE=TCAM in DFHTCT
     TYPE=TERMINAL macro)
TCAM logical units (defined by
     TCAMFET=SNA in DFHTCT TYPE=LINE
     macro and SESTYPE=3600|3767|3770
     |3790|BCHLU|INTLU in DFHTCT
     TYPE=TERMINAL macro)
VTAM logical units:
     3270
     LUTYPE2
     LUTYPE3
     LUTYPE4
     SCSPRT
     3600
     3650 (host-conversational (3270)
          and interpreter LUs only)
     3767
     3770
     3790 (all except inquiry LU)
```

Some special BMS programming considerations that apply only to particular terminal subsystems are described in the various CICS subsystem guides (for example, the IBM 4700/3600/3630 Guide). These guides are listed in the Bibliography.

With BMS, a CICS installation with more than one type of terminal need provide only one program for each application transaction to support all terminal types in the installation. BMS identifies which terminal type is requesting use of the application program and provides for the conversion of the device-dependent data stream to and from the device-independent format used by the application program. A CICS installation using only one type of terminal may nevertheless wish to use the formatting services of BMS to facilitate the addition of other terminal types or the conversion to another terminal type in the future.

FORMAT INDEPENDENCE

Format independence permits the application program to provide data to one or more terminals or to receive data from a terminal without regard for the physical placement of fields within the data stream or on the terminal.

All references to data by the application program are through symbolic field names. The placement of fields within the data stream is accomplished by BMS through the use of information stored in data format tables called **maps**. A CICS installation in which BMS is used may rearrange the fields to be included in a terminal message by simply changing some values stored in the map that defines the format of the message. The application program that causes the message to be written need not be modified. Programming maintenance can thus be considerably simpler than if BMS were not used.

Format independence also permits certain constant information, such as headings, field-identifying keywords, and 3270 screen formats, to be stored in maps. These constants can be modified simply by changing their values in the maps. Any 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 may be compared with the independence provided by DL/I for data bases. Both remove from the application program the requirement to know the physical placement of fields within the data record or message. Fields may be physically rearranged, removed, or added without necessitating program maintenance on all application programs using the record or message.

FACILITIES OF BMS

The facilities that BMS provides are data mapping and formatting, terminal paging, and message routing.

DATA MAPPING AND FORMATTING

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

The application program may select any of three standard data formats in which to provide or accept data from BMS: field data format, block data format, or text data format.

When field data format is used, data is passed to BMS as separate fields. Each field is given a symbolic field name by the application programmer. This name 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 single attribute byte (used for 3270 output operations only, but present for all terminal types), and the data area. A map describing the position of the field when displayed or printed, the data length, and other information about each field is created to control the mapping function.

When block data format is used, data is passed to BMS as line segments. Fields positioned within the line segments may be given symbolic field names to aid the application program in positioning the fields. Each field provides for a single attribute byte and the 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 positions when displayed or printed, data lengths, and other necessary information.

When text data format is used, output data consisting of a data stream with optional new-line (X'15') characters is passed to BMS. BMS divides the data stream into lines no longer than those defined for the particular terminal to which the data stream is related. BMS will only allow a line break to occur where it encounters a blank (X'40'). If a word will not fit into the space remaining in a line, BMS places the whole word on a new line. If new-line characters are included in the data stream, they too are honored. CICS 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 the necessary new-line characters. Maps are not used with text data format.

Field data format is the commonest data format 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 that is 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.

TERMINAL PAGING

Terminal paging permits the application program to (1) combine several small mapped data areas into one or more pages of output, or (2) prepare more output than can be contained in one page of output. By definition, a page is the physical area of a terminal on which data is displayed or printed at one time. The size of the area (in numbers of lines and columns) is specified for the particular terminal in the CICS terminal control table (the TCT) by the system programmer.

Since a page of output may be constructed by BMS from several small maps, it is convenient to generate these maps together in a map set. A map set is a collection of maps generated and stored together in the CICS program library. A reference to one map in the map set causes the entire map set to be loaded into storage for the duration of the task or until another map set is referred to by the task. DFHMSD, DFHMDI, and DFHMDF macros, described later in this chapter, are used in constructing the map set.

During execution, the application program issues DFHBMS TYPE=PAGEBLD macros to position portions of an output page. If all the data cannot be contained on one page, BMS recognizes an overflow condition and can transfer control to an overflow routine within the application program. This routine normally causes 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 the output message is completed, the page is written to temporary storage to await completion of the **logical message**. A logical message is the result of one or more BMS requests for output services all of which have the same disposition (OUT, STORE, or RETURN, as explained later in this chapter). To cause the logical message to be completed, the application program issues a DFHBMS TYPE=PAGEOUT macro. Alternatively, the logical message is completed upon termination of the application program unless a short-on-storage condition exists, in which case the logical message is deleted.

Terminal paging provides the additional function of building a logical message without the use of maps. A DFHBMS TYPE=TEXTBLD macro is issued to request this type of page building. The data is passed to BMS as text data, 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. The formatting of the logical message can be controlled through the data itself by embedding new-line characters (X'15') within the data. To cause the TEXTBLD logical message to be completed, the application program issues a DFHBMS TYPE=PAGEOUT macro or terminates execution.

DFHBMS TYPE=PAGEBLD and TYPE=TEXTBLD macros 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 DFHBMS TYPE=PURGE macro. This instruction deletes the portions of the message already built in main storage or on temporary storage.

MESSAGE ROUTING

Message routing permits an application program to build and route a logical message to one or more terminals. The message is automatically scheduled for each designated terminal, to be delivered as soon as the terminal is available to receive messages or at some future time.

The page building facility of BMS is used for message routing, so the design of application programs is very similar for the two facilities. Message routing allows application-built messages to be sent to any prescribed terminals.

To initiate a routing operation, the application program issues a DFHBMS TYPE=ROUTE macro followed by DFHBMS TYPE=(PAGEBLD,STORE) or TYPE=(TEXTBLD,STORE) instructions to build the logical message that is to be routed. A DFHBMS TYPE=PAGEOUT macro 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 sent as one message.

A parameter of the DFHBMS TYPE=ROUTE macro points to a list of terminals to receive the routed message. The list may contain the terminal identification and operator identification of each terminal designated to receive the message. If only a terminal identification is specified, the message is routed to that terminal, regardless of who is signed on at the terminal. If both the terminal identification and the operator identification are specified, the message is routed to the terminal but delivered only when the specified operator is signed on. If only the operator identification is specified, BMS scans the terminal control table and delivers the message to the first terminal at which the operator is signed on.

Another parameter of the DFHBMS TYPE=ROUTE macro is a specific operator class code. If specified, only an operator signed on with that class code may receive the routed message. One to twenty-four class codes may be assigned to operators in the CICS sign-on table.

The DFHBMS TYPE=ROUTE macro further designates whether the message is to be delivered as soon as possible or at a specific time or after some interval of time. If the routed message cannot be delivered within a specified length of time, an error message may be returned to the terminal sending the message or to some designated alternative terminal. The message may be deleted, or it may be retained indefinitely, until delivered or until deliberately deleted by an operator at the receiving terminal.

If a message is to be routed to more than one terminal type, BMS builds a device-dependent message for each terminal type. Each such message is stored on temporary storage until all terminals for which it is destined 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 time period (specified at system generation) has elapsed, causing the message to be deleted by BMS.
- The message is deleted by the destination terminal.

Another consideration of routing to different terminal types is the handling of overflow conditions. Since different terminal types 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 terminal type caused the overflow and how many pages have been created for that terminal type.

If a message is routed to terminals with alternate screen size capabilities, the selection of the screen size to be used is taken from the SCRNSZE parameter in the PCT for the routing transaction. (The SCRNSZE parameter is described in the appropriate CICS Resource Definition manual.)

The message routing facility of BMS is an ideal tool for developing message switching and broadcasting applications. CICS provides a generalized message switching transaction that uses the message routing facility of BMS. Use of the message switching transaction is described in the appropriate CICS-Supplied Transactions book.

MAPPING CONCEPTS AND TECHNIQUES

Most of the facilities of BMS (text data format is the exception) require two forms of map to be defined by CICS macros and assembled offline in advance of running the application program. The two forms are: (1) a physical map used by BMS to convert data to or from the format desired by the application programmer, and (2) a symbolic description map used by the application programmer to symbolically refer to the data in the terminal buffer. The physical map is a table of information about each field, and is stored in the CICS program library to be loaded by BMS at execution time. The symbolic description map is a set of source statements that are cataloged into the appropriate source library (Assembler, COBOL, or PL/I) and copied into the application program when it is assembled or compiled.

The programmer defines and provides names for fields and groups of fields that may be written to and received from the devices supported by BMS. The symbolic description map can be copied into each application program that uses the associated physical map. Data can thus be passed to and from the application program under the field names in the symbolic description map. Since the application program is written to manipulate the data under the field names, altering the map format by adding new fields or rearranging old fields does not necessarily alter the program logic.

If the map format is altered, it is necessary in most cases to make the appropriate changes to the macros that describe the map and then reassemble both the physical map and the symbolic description map. The new symbolic description map must then be copied into the application program and the program reassembled. There are certain map alterations that can be made without necessitating reassembly of the symbolic description map.

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

By using BMS to construct and interpret data streams, application programmers can insulate application programs from the device-dependent considerations required to handle the data streams. If necessary, the application program has the facility to temporarily modify 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 progressively 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.

MAP DEFINITION

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 libraries.

Map definition is accomplished through the use of three different macros: DFHMSD, DFHMDI, and DFHMDF.

The DFHMSD macro

- Defines a map set
- Indicates whether a particular set of macros is for a physical map or for a symbolic description map
- Specifies whether the map is for input, output, or both
- Can specify the data format: field or block.

The DFHMDI macro

- Defines a map
- Defines the position of the map on the page, either absolutely or in relation to other maps
- Specifies the size of the map
- Can specify the data format: field or block.

The 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 are given later in this chapter. An example of their use and of the symbolic storage definitions generated is given in Appendix B.

The map definition macros are assembled twice, once to produce the map used by BMS, and once to produce the symbolic storage definition (or DSECT) that will be copied into the application program.

INPUT MAPPING

For an input map, the maximum data length and the starting position of each field must be defined.

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 "fieldname.L" or "groupname.L".

The flag area is a one-byte field and is addressed by the name "fieldname.F" or "groupname.F".

The data area 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 area addressed by the name "groupname.I" or "fieldname.I". For fields contained within a group, there are no intervening length or flag areas (only "groupname.L" exists) but each field is addressed by a name "fieldname.I".

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 "fieldname D", and the first byte of the next occurrence of the field is named "fieldname N". 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, "fieldname D" is the name of the array of minor structures containing the length, flag, and data areas of the field.

Note that "." is a concatenation symbol used here only to show how the symbolic names are suffixed; the period is never actually coded. For example, in the case of field name XYZ, the length area is referenced as XYZL; the flag area is referenced as XYZF; and the data area is referenced as XYZI.

The length specified for a field may differ from the number of characters that are entered for 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.

With a 3270 or similar type of device, the length of the input field will be the number of nonnull characters contained in the field. Note that a previous output mapping operation may have entered nonnull characters into the field.

The flag area 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 all nulls), the length area is set to zeros and the flag area is set to X'80'.

Specifying ATTRB=FSET on the DFHMDF macro (see page 163) causes the field to be returned with the same length, flag, and data areas as if the operator entered it, unless the field was originally nulls, in which case, length is set to 0, flag is set to X'80', and data is set to nulls,

Any fields that are entered as input but are not defined in the map are discarded. The length and data areas of any fields defined but not keyed are set to nulls (X'00').

For a 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 pen-detectable field contains a binary one if selected or a binary zero if not.

OUTPUT MAPPING

For each output field, the starting location, length, field characteristics, and default data (if desired) must be defined.

The fields of an output map are assigned names in the DFHMDF macro. The characteristic or attribute byte is named "fieldname.A" or "groupname.A". For a field contained within a group, the data area is given the name "fieldname.O", 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 "groupname.O" or "fieldname.O."

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 "fieldname D", and the first byte of the next occurrence of the field is named "fieldname N". 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, "fieldname D" 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.

Note that "." is a concatenation symbol used here only to show how the symbolic names are suffixed; the period is never actually coded. For example, in the case of field name XYZ, the data is referred to as XYZO; the attribute byte is referred to as XYZA.

If output maps are to be used by application programs coded at command level, 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 he has specified a name. 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.

Note: 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').

Pen-detectable fields should be "auto skip" to prevent data from being keyed into them. Because of the nature of pen-detectable 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 blank character; otherwise, the field is no longer 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 (INOUT) 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. These limits are stated in the description of the DFHMDF macro later in this chapter.

MAP RETRIEVAL

Map sets placed in the CICS program library are accessed by BMS through program control DFHPC TYPE=LOAD macros. Therefore, each map set name must be entered in the processing program table (PPT) by the system programmer. When device-dependent map sets are placed in the CICS 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.)

To avoid having to load a map set during execution, an assembler language programmer using the macro level interface may include the map set in the program, place the address of the map set at TCAMSMSA, and code MSETADR=YES in the DFHBMS macro. Alternatively, the programmer may code MSETADR=symb-addr, where the symbolic address is the label of the map set. The MAP=map-name specification must also be provided with the MSETADR parameter to locate a specific map within the map set. Similarly, the MAPADR operand enables an assembler language programmer to specify, directly or indirectly, the address of an individual map.

COPYING SYMBOLIC DESCRIPTION MAPS

The symbolic description maps must be copied into the application program as shown in the following examples. These examples use the macro level interface, examples using the command level interface are given in the appropriate CICS Application Programmer's Reference Manual (Command Level).

In the following examples, mapsetname1, mapsetname2, and mapsetname3 are the names of members that contain the assembly of a BMS symbolic storage definition.

1. Assembler language COPY instructions for each symbolic storage definition. To ensure that each definition overlays the same area, the second and subsequent COPY instructions must be preceded by an ORG instruction to reposition the assembler to the start of the TIOA data area.

```
COPY DFHTIOA
COPY mapsetname1
ORG  TIOADBA
COPY mapsetname2
ORG  TIOADBA
COPY mapsetname3
```

2. COBOL COPY statements for each symbolic storage definition. In this example, mapname1I, mapname2I, and mapname3I are the names of the first maps in the map sets.

```
LINKAGE SECTION.
01 DFHBLLDS COPY DFHBLLDS.
   02 TIOABAR PIC S9(8) COMP.
    .
    .
01 DFHCSADS COPY DFHCSADS.
01 DFHTCADS COPY DFHTCADS.
01 DFHTIOA COPY DFHTIOA.
01 mapname1I COPY mapsetname1.
01 mapname2I COPY mapsetname2.
01 mapname3I COPY mapsetname3.
```

Note: For MODE=IN and MODE=INOUT the format of the COPY statement is:

```
01 mapname1I COPY mapsetname1
```

For MODE=OUT the format of the COPY statement is:

```
01 mapname1O COPY mapsetname1
```

3. PL/I %INCLUDE statements.

```
%INCLUDE DFHTIOA;
    2 DUMMY CHAR(1);
%INCLUDE mapsetname1;
%INCLUDE mapsetname2;
%INCLUDE mapsetname3;
.
.
.
```

In addition to providing the BMS symbolic storage definition for the TIOA, the application programmer must establish addressability for this storage definition. Depending on the programming language used, this is accomplished as follows:

1. Assembler language L instruction to set up TIOABAR, normally from TCASCSA. For example:

```
COPY DFHTIOA
COPY mapsetname1
ORG  TIOADBA
COPY mapsetname2
ORG  TIOADBA
COPY mapsetname3
.
.
.
DFHSC TYPE=GETMAIN,
      NUMBYTE=mapname.E-TIOADBA,
      CLASS=TERMINAL,
      INITIMG=00
L     TIOABAR,TCASCSA
```

Note: BMS offline macros generate a label at the beginning and end of each map description and a label at the end of each map set description; these labels have the form "mapname.S", "mapname.E", and "mapsetname.T", respectively, where "." is a concatenation symbol used only for documentational purposes. The start of each map, or map set, can be referred to by the label TIOADBA. Thus an assembler language programmer can specify the amount of storage required in the way shown in the example above. The last L instruction establishes TIOA addressability.

2. COBOL 02 level statements immediately following the COPY statement for the Linkage Section Base Locator (BLL). These 02 statements must be coded in the same order as the corresponding 01 statements. For example:

```
LINKAGE SECTION.
01 DFHBLLDS COPY DFHBLLDS.
.
.
.
    02 TIOABAR PIC S9(8) COMP.
    02 MAPBASE1 PIC S9(8) COMP.
    02 MAPBASE2 PIC S9(8) COMP.
    02 MAPBASE3 PIC S9(8) COMP.
.
.
.
01 DFHTIOA COPY DFHTIOA.
01 mapname1 COPY mapsetname1.
01 mapname2 COPY mapsetname2.
01 mapname3 COPY mapsetname3.
.
.
.
PROCEDURE DIVISION.
.
.
.
DFHSC TYPE=GETMAIN,NUMBYTE=120,
    CLASS=TERMINAL,INITIMG=00
    MOVE TCASCSA TO TIOABAR.
    ADD 12 TIOABAR
    GIVING MAPBASE1.
    MOVE MAPBASE1
    TO MAPBASE2 MAPBASE3.
```

3. Set up the PL/I based pointer variable (BMSMAPBR) on which the map structures are based. For example:

```
%INCLUDE DFHTIOA;;
%INCLUDE mapsetname1;
%INCLUDE mapsetname2;
%INCLUDE mapsetname3;
.
.
.
DFHSC TYPE=GETMAIN,
      NUMBYTE=120,
      CLASS=TERMINAL,
      INITIMG=00
TIOABAR=TCASCSA;
BMSMAPBR=ADDR(TIOADBA);
```

Note that this code assumes that the TIOAPFX operand of the DFHMSD and DFHMDI macros has been omitted or coded as TIOAPF. Each of the maps (mapsetname 1-3) is based on the same pointer variable - BMSMAPBR.

MAP DEFINITION MACROS

The syntax and operand descriptions of the three map definition macros (DFHMSD, DFHMDI, and DFHMDF) are given below.

You should refer to the appropriate CICS Application Programmer's Reference Manual (Command Level) for descriptions of the additional attributes field outlining (OUTLINE), mixed DBCS and EBCDIC fields (SOSI), and background transparency (TRANSP) that are new for this release.

DEFINING A MAP SET (DFHMSD MACRO)

BMS generates and stores map sets in the CICS program library under the names selected by the application programmers. A reference to one map in the map set causes the entire map set to be loaded into storage for the duration of the task, or until another map set is referred to by the task.

Information pertaining to an entire map set is specified in the DFHMSD macro, which always appears at the beginning and end of each map set generation. The one at the beginning indicates whether physical maps or symbolic description maps are being generated; the one at the end indicates the end of the map set.

All operands other than the TYPE operand of a DFHMSD macro are the same for a physical map generation run and for the corresponding symbolic description map generation run. The application programmer should specify TYPE=MAP for the former, and TYPE=DSECT for the latter. Alternatively, physical maps and symbolic description maps can be assembled in the same job by the use of job control language options, as described in the appropriate CICS Installation and Operations Guide.

The format of the DFHMSD macro is shown in Figure 15.

The operands of the DFHMSD macro are as follows:

mapset

is the one- to seven-character name of the map set, to be specified in the MAPSET operand of any DFHBMS macro that refers to the map set. The name must begin with an alphabetic character and, if the map is to reside in the CICS program library, must differ from other map names or program names.

A suffix specified by the SUFFIX operand, or based on the terminal type specified in the TERM operand of the DFHMSD macro is appended to the map set name during assembly. This suffixed name is the name that should be used in the NAME card (CICS/OS/VS) or the PHASE card (CICS/DOS/VS) in cataloging the map set (see the appropriate CICS Installation and Operations Guide for further details), and the name that should be specified by the system programmer in the PPT entry (see the appropriate CICS Resource Definition manual). The suffixes are tabulated in the description of the TERM operand, below.

When a mapping operation is requested by means of a DFHBMS macro in an application program, CICS automatically appends a similar suffix to the map set name specified in that macro, and attempts to load a map set with the suffixed name. If the load is unsuccessful, that is, the suffixed map set name cannot be found in the library, CICS will load a map set with an unsuffixed name (equivalent to being suffixed with a blank). CICS obtains the suffix from the TCT terminal entry for the appropriate terminal (either the terminal associated with the transaction or, for routing, the destination terminal), and this suffix depends 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) 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 SUFFIX operand of the DFHMSD macro. If this load is unsuccessful, normal map set selection will occur.

For example, if two maps are assembled, one with TERM=CRLP and the other with TERM=ALL, the first will be suffixed with A and the second with blank (that is, unsuffixed). The system programmer should use these suffixed names in the PHASE/NAME cards and in the PPT entry. If a CICS transaction now routes a message to two terminals, one of which has TRMTYPE=CRLP and the other TRMTYPE=L3277, TRMMODL=2, BMS will attempt to load mapset.A and mapset.M to do the mapping in the two cases. The second of these will be unsuccessful, so BMS will then look for the unsuffixed map set name for routing to the 3277.

TYPE=

indicates the generation function of the macro. If both map and DSECT are to be generated in the same job, the SYSPARM option can be used in the assembler job execution step, as described in the appropriate CICS Installation and Operations Guide.

DSECT

indicates that this is a symbolic description map generation run to create the list of field names to be copied into an application program. If a single map set is to be used by application programs written in different languages, a separate DFHMSD TYPE=DSECT macro must be written for each language to put the table of field names into the copy library of the language.

MAP

indicates that this is a physical map generation run to create the control information block used by BMS to perform mapping. This physical map is stored in the CICS program library and loaded as required by BMS. The assembler language application programmer can, alternatively, generate the map in his program and pass the address of the map to BMS instead of using this facility to generate and store the map beforehand in the CICS program library.

FINAL

must be coded in the DFHMSD macro that marks the end of the map set. If other parameters are coded in the DFHMSD TYPE=FINAL macro, they will be ignored.

```
mapset  DFHMSD   TYPE={DSECT|MAP|FINAL}
                       [,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|YES|MAPONLY}]
                       [,HILIGHT={OFF|BLINK|REVERSE|UNDERLINE}]
                       [,HTAB=tab[,tab]...]
                       [,LANG={ASM|COBOL|PLI}]
                       [,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]...]
```

Figure 15. DFHMSD Macro (Define a Map Set)

BASE=name
is used to indicate that the same storage base will be used for the symbolic description maps from more than one map set. The same name is coded in the BASE operand for each map set that is to share the same storage base. Since all map sets with the same base describe the same storage, data related to a previously-used map set may be overwritten when a new map set is used. Furthermore, different maps within the same map set will also overlay one another.

This operand is not valid for assembler language programs.

As an example, assume that the following DFHMSD macros are used to generate symbolic description maps (symbolic storage definitions) for two map sets.

```
MAP1    DFHMSD TYPE=DSECT,
               TERM=2780,
               LANG=COBOL,
               BASE=DATAREA1,
               MODE=IN

MAP2    DFHMSD TYPE=DSECT,
               TERM=3270,
               LANG=COBOL,
               BASE=DATAREA1,
               MODE=OUT
```

The symbolic storage definitions 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 MAP1.
  01 name COPY MAP2.
```

MAP1 and MAP2 multiply redefine DATAREA1; only one 02 statement is needed to establish addressability. However, the program can only use the fields in one of the symbolic map areas at a time.

If BASE=DATAREA1 is deleted from this example, an additional 02 statement is needed to establish addressability for MAP2; the 01 DATAREA1 statement is not needed. The program could then refer to fields concurrently in both symbolic map areas.

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 storage definition 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 operand of a DFHMDI or DFHMDF macro. If this operand is specified when EXTATT=NO, a warning will be issued and the operand ignored. If this operand is specified, but EXTATT is not, EXTATT=MAPONLY will be assumed. Refer to the EXTATT operand for device dependencies.

CTRL=

is used to specify 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. To be effective, this operand must be specified on the last (or only) map of a page unless the CTRL operand of the DFHBMS macro is 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, <u>HONEOM</u>

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 line printer length to be used.

FREEKB

specifies that the keyboard should be unlocked after this map is written out. If omitted, the keyboard remains locked; further data entry from the keyboard is inhibited until this status is changed.

ALARM

activates the 3270 audible alarm feature. For a VTAM terminal ALARM signals BMS to set the alarm flag in the function management header; this feature is not supported by interactive and batch logical units.

FRSET

indicates that the modified data tags (MDTs) of all fields currently in the 3270 buffer are to be reset to a not-modified condition (that is, field reset) before any map data is written to the buffer. This allows the DFHMDF ATTRB specification for the requested map to control the final status of any fields written or rewritten in response to a DFHBMS macro.

DATA=

specifies the format of the data as seen by the application program.

<u>FIELD</u>

indicates 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 (this field is 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)," later in this chapter).

BLOCK

indicates that the data is passed as a continuous stream which is processed as line segments of the length specified in the map used to process this 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.

```
| A |data field|space...
```

The data type associated with any map depends on the DATA specifications, or lack thereof, in both the DFHMSD and DFHMDI macros:

1. A DATA operand in a DFHMDI macro will always override that in a DFHMSD macro.

2. If no DATA operand is coded in the DFHMDI macro, the DATA operand in the DFHMSD macro will apply.

3. If no DATA operand is coded in either macro, DATA=FIELD is the default.

EXTATT=
specifies whether the extended attributes (COLOR, HILIGHT, PS, and VALIDN) are supported.

<u>**NO**</u>
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, and DFHMDF macros will be invalid.

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).

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 the application program.

HILIGHT=
specifies the default highlighting attribute for all fields in all maps in a map set. See the EXTATT operand for device dependencies.

<u>**OFF**</u>
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, 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.

<u>**ASM**</u>
indicates that the symbolic description map is to be referred to by an assembler language program.

COBOL
indicates that the symbolic description map is to be referred to by a COBOL program.

PLI
indicates that the symbolic description map is to be referred to by a PL/I program.

RPG
indicates that the symbolic description map is to be referred to by an RPGII program. This parameter is valid for CICS/DOS/VS only.

LDC=mnemonic
specifies the mnemonic to be used by CICS 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
indicates an input map generation.

__OUT__
indicates an output map generation.

INOUT
indicates that the map definition is to be used for both input and output mapping operations.

Note: 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 DFHBMS TYPE=IN macro.

OBFMT=
specifies whether outboard formatting is to be used. This operand is available only for 3650 logical units. Refer to the appropriate CICS 3650/3680 Guide for details of 3650 logical units and of outboard formatting.

YES
indicates that all maps within this map set are eligible for use in outboard formatting, except those for which OBFMT=NO is specified in the DFHMDI macro.

__NO__
indicates that no maps within this map set are eligible for use in outboard formatting, except those for which OBFMT=YES is specified in the DFHMDI macro.

PS=
specifies that programmed symbols are to be used. see the EXTATT operand for device dependencies.

__BASE__
specifies that only the basic symbols are used.

psid
specifies a single EBCDIC character or a hexadecimal code on the form X'nn', that identifies the set of programmed symbols.

If PS is specified when EXTATT=NO, a warning is issued and the option ignored. If PS 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 storage definitions of the maps in the map set are to be separate (that is, not redefined) areas. This operand is used when the symbolic storage definitions are copied into the WORKING-STORAGE section of a program using the command level interface and the storage for the separate maps in the map set is to be used concurrently. (For information about the command level interface, see the appropriate CICS Application Programmer's Reference Manual (Command Level).

Specifies, for PL/I programs, that the symbolic storage definitions are to be declared as having the AUTOMATIC storage class. If not specified, the symbolic storage definitions are declared as having the BASED storage class.

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. The use of numeric suffixes would help prevent conflict.

TERM=terminal type
indicates the type of output device or logical unit associated with the map set. The parameters that may be coded after TERM= are given in the left-hand column of the table below.

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, the 3790 full function LU, and SCS-printer LUs (3270 and 3290).

[2] Use also for host conversational (3653) LU.

[3] Use also for interpreter LU.

[4] Use also 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.

The application programmer who specifies ALL in the TERM operand must be certain that device-dependent characters are not included in the map set and must ensure 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, a specification of ALL, indicating that one map is to be used for multiple terminals, 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 in the TERM operand. Alternatively, the BMS support for device-dependent map sets can be left ungenerated by specifying NODDS in the BMS operand of the DFHSIT system generation macro. (See the appropriate <u>CICS Customization Guide</u> for further details.)

TIOAPFX=

specifies whether BMS should include a filler in the symbolic TIOA description(s) to allow for the unused TIOA prefix. If this operand is coded, the same storage address may be used for TIOABAR and the map base.

YES

indicates that the filler should be included in the symbolic TIOA description(s). This operand is ignored unless TYPE=DSECT is coded. If TIOAPFX=YES is coded, all maps within the map set have the filler, except when TIOAPFX=NO is coded on the DFHMDI macro.

<u>**NO**</u>

is the default and indicates that the filler is not to be included. The filler may still be included for a specific map if TIOAPFX=YES is coded on the DFHMDI macro.

Note: In previous versions of CICS, it has not been valid to code TIOAPFX=YES for an assembler language application program. If this operand was coded in this way, CICS disregarded it and applied the default specification (TIOAPFX=NO). In CICS Version 1.4, it is valid to code TIOAPFX=YES for an assembler program: doing so will thus produce a different object program under CICS/VS 1.4 from that which would be produced under earlier versions.

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.

See the EXTATT operand for device dependencies.

VTAB=tab[,tab]...

specifies one or more tab positions for use with interactive and batch logical units having vertical forms control.

DEFINING A MAP (DFHMDI MACRO)

The DFHMDI macro is used to define a single map. It defines the size of the data to be mapped and its position within the input or output. When defining more than one map within a map set, 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 COBOL data areas and not to the size of the map on the screen). The format of the DFHMDI macro is shown in Figure 16 on page 158.

The operands of the DFHMDI macro are as follows:

map
: is the one- to seven-character name of the map, to be specified in the MAP operand of any DFHBMS macro that refers to the map.

  Map names within a map set, or within multiple map sets that are copied into one application program, should be unique.

COLOR=
: specifies the default color for all fields in a map unless overridden explicitly by the COLOR operand of a DFHMDF macro. If this option is specified when EXTATT=NO, 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 specification controls whether map and page margin selection and column counting are to be done with reference to 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.

  <u>SAME</u>
  : indicates that the left or right map margin is to be established in the same column as the last map used that specified COLUMN=number and the same JUSTIFY parameters as this macro.

  Refer to the section "Map Positioning" on page 170 for a more detailed discussion.

CTRL=
: is used to specify device characteristics related to terminals of the 3270 Information Display System. CTRL=ALARM is valid for TCAM SNA 3270 SDLC and VTAM-supported terminals (except interactive and batch logical units); all other parameters for CTRL are ignored. To be effective, this operand must be specified on the last (or only) map of a page unless the CTRL operand of the DFHBMS macro is 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 BMS output request is directed to a 3270 display without the Printer Adapter feature.

  L40, L64, L80, <u>HONEOM</u>
  : 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 line printer length to be used.

  FREEKB
  : specifies that the keyboard should be unlocked after this map is written out. If omitted, the keyboard remains locked; further data entry from the keyboard is inhibited until this status is changed.

  ALARM
  : activates the 3270 audible alarm feature. For a VTAM terminal, ALARM signals BMS to set the alarm flag in the function management header; this feature is not applicable to interactive and batch logical units.

  FRSET
  : indicates that the modified data tags (MDTs) of all fields currently in the 3270 buffer are to be reset to a not-modified condition (that is, field reset) before any map data is written to the buffer. This allows the DFHMDF ATTRB specification for the requested map to control the final status of any fields

```
 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}]
                  [,EXTATT={NO|YES|MAPONLY}]
                  [,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 16. DFHMDI Macro (Define a Map)

written or rewritten in response to a DFHBMS macro.

DATA=
specifies the format of the data as seen by the application program.

FIELD
indicates 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 (this field is 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" on page 181.

BLOCK
indicates that the data is passed as a continuous stream which is processed as line segments of the length specified in the map used to process this 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.

```
| A |data field|space...
```

A DATA specification in a DFHMDI macro overrides a DATA specification in a DFHMSD macro.

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, and DFHMDF macros will be invalid.

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).

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 the application program.

HEADER=YES
allows this map to be used during PAGEBLD overflow without terminating the overflow condition. See "PAGEBLD Overflow Processing" on page 173. 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, a warning will be issued and the option ignored.

JUSTIFY=
describes 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 DFHBMS requests 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 LEFT nor RIGHT is specified, LEFT is assumed. 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" on page 170 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, indicating a starting line number. A request to map data on a line and column that has been formatted in response to a preceding request 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
indicates 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
indicates that formatting of data is to begin on the same line as that used for a preceding DFHBMS request. 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" on page 170 for a more detailed discussion.

OBFMT=
specifies whether outboard formatting is to be used. This operand is available only for 3650 logical units. Refer to the appropriate CICS IBM 3650/3680 Guide for details of 3650 logical units and of outboard formatting.

If OBFMT is not coded in the DFHMDI macro, the OBFMT specification in the DFHMSD macro is used.

YES
indicates that this map is to be used with outboard formatting.

NO
indicates 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 on 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.

SIZE=
gives the dimensions of a map in terms of length and width.

line
is a value from 1 to 240, indicating the length of a map as a number of lines.

column
is a value from 1 to 240, indicating the width of a map as a number of characters per line. Space for the attribute byte should be included in the column specification.

The SIZE operand is required in the following cases:

- A POS=(line,column) specification is given in a DFHMDF macro defining a specific field within this map.
- This map is to be referred to in a DFHBMS TYPE=PAGEBLD macro.
- This map is to be used when referring to input data from other than a 3270 terminal in a DFHBMS TYPE=IN or DFHBMS TYPE=MAP macro.

TIOAPFX=
specifies whether or not BMS should include a filler in the symbolic TIOA description to allow for the unused TIOA prefix. If this operand is coded, the same storage address may be used for TIOABAR and the map base. If this operand is not coded, the TIOAPFX specification derived from the DFHMSD macro is used.

YES
indicates that the filler should be included in the symbolic TIOA description for this map. This operand is ignored unless TYPE=DSECT is coded on the DFHMSD macro.

NO
indicates the filler is not to be included for this map.

Note: In previous versions of CICS, it has not been valid to code TIOAPFX=YES for an assembler language application program. If this operand was coded in this way, CICS disregarded it and applied the default specification (TIOAPFX=NO). In CICS In CICS Version 1.4, it is valid to code TIOAPFX=YES for an assembler program: doing so will thus produce a different object program under CICS/VS 1.4 from that which would be produced under earlier versions.

TRAILER=YES
allows this map to be used during PAGEBLD overflow without terminating the overflow condition (see "PAGEBLD 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 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.

DEFINING A FIELD (DFHMDF MACRO)

The DFHMDF macro is used to define one field in a map. One DFHMDF macro is required for each field, giving information such as symbolic field name, field position, field length, attribute byte (for 3270 terminals), initial constant data, justification of input, and COBOL or PL/I data picture.

The maximum number of named fields that can be defined for a COBOL or PL/I input/output map is 1023.

The format of the macro is shown in Figure 17 on page 162.

The operands of the DFHMDF macro are as follows:

fld
is the 1 through 7-character name of the field, used as a symbolic reference to the field by the application program.

Field names within a map, or within multiple maps that are copied into one application program, should be unique.

Although specification of a field name is not required for every field within a map, a field name must be specified for at least one field of any map to be compiled under COBOL or PL/I. All fields within a group must have names.

If no name is specified for a field, an application program cannot access the field map 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, any data supplied by the user overlays data supplied by initialization (unless DATA=NO is specified or assumed by default).

POS=
is used to specify the individually addressable character location in a map at which the attribute byte that precedes this field is positioned. Specification of the DFHMDF macro must be sequenced by the POS operand except for output mapping operations using DATA=FIELD.

The POS operand defines the location of fields in a map. The location of data on the output medium is dependent on DFHMDI macro parameters as well.

For each field definition (DFHMDF macro), the first position is reserved for an attribute byte. When supplying data for input mapping from non-3270 devices, the actual 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 coded which represents the last character position in the 3270, then two special rules apply:

- The IC attribute should not be coded on that DFHMDF macro. The cursor may be set to location zero by using the cursor operand of the DFHBMS macro.
- 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.

number
is an absolute displacement (relative to zero) from the beginning of the map being defined.

(line,column)
are line and column specifications (relative to one) within the map being defined.

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

[fld]	DFHMDF	[,POS={number|(line,column)}] [,ATTRB=([{ASKIP|PROT|UNPROT[,NUM]}],{BRT|NORM|DRK}] [,DET][,IC][,FSET])] [,COLOR={<u>DEFAULT</u>|BLUE|RED|PINK|GREEN|TURQUOISE| YELLOW|NEUTRAL}] [,GRPNAME=group-name] [,HILIGHT={<u>OFF</u>|BLINK|REVERSE|UNDERLINE}] [,INITIAL='character data'|XINIT=hexadecimal data] [,JUSTIFY=([{LEFT|RIGHT}][,{BLANK|ZERO}])] [,LENGTH=number] [,OCCURS=number] [,PICIN='value'] [,PICOUT='value'] [,PS={<u>BASE</u>|psid}] [,VALIDN=([MUSTFILL][,MUSTENTER])]

Figure 17. DFHMDF Macro (Define a Field)

fields. (Refer to the publication An Introduction to the IBM 3270 Information Display System for a full explanation of the effects of the attribute byte settings.)

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 PROPT=NLEOM is specified on the DFHBMS TYPE=PAGEBLD macro 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 non-display 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
indicates that data cannot be keyed into the field and causes the cursor (current location pointer) to automatically skip over the field.

PROT
indicates 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. Refer to the publication An Introduction to the IBM 3270 Information Display System for further information.

UNPROT
indicates 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 An Introduction to the IBM 3270 Information Display System 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

indicates that the cursor is to be placed in the first position of this field. The IC attribute for the last field for which it is specified in a map is the one that takes effect. If not specified for any fields in a map, the default location is 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 operand for the BMS request that causes the write operation. See the descriptions of the DFHBMS TYPE=PAGEBLD, mEXTBLD, and OUT macros, later in this chapter.

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 color to be used. 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.

GRPNAME=group name

is the name used to generate symbolic storage definitions and to combine specific fields under one group name. The group name has a maximum length of seven characters. The same group name must be specified for each field that is to belong to the group.

If this operand is specified, the OCCURS operand cannot be specified.

The fields in a group must follow on; there can be 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.

Appendix B contains examples showing, amongst other things, the effect of the GRPNAME operand.

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 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.

INITIAL='character data'|XINIT=hexadecimal data
is used to specify constant or default data for an output field. The INITIAL operand is used to specify data in character form; the XINIT operand is used to specify data in hexadecimal form. INITIAL and XINIT are mutually exclusive.

For fields with the DET attribute, initial data that begins with a blank character, "&", ">", or "?" should be supplied.

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=
indicates 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.

Note: If a field is initialized by an output map or contains data from any other source, data that is keyed as input will only overwrite equivalent length existing data; surplus existing data will remain in the field and could cause unexpected interpretation of the new data.

LENGTH=number
indicates the length (from 1 to 256 bytes) of this 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 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 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 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, of course) 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.

Appendix B contains examples showing, amongst other things, the effect of the OCCURS operand.

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 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.

Note: A P S V X 9 / are the valid picture values for COBOL maps.

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     COMP PIC S9(4).
      02  F1A     PICTURE X.
      02  FILLER REDEFINES F1A.
       03  F1F    PICTURE X.
      02  F1I     PIC X(30).
      02  FILLER  PIC X.
      02  F2L     COMP PIC S9(4).
      02  F2A     PICTURE X.
      02  FILLER REDEFINES F2A.
       03  F2F    PICTURE X.
      02  F2I     PIC X(10).
      02  FILLER  PIC X.
      02  F3L     COMP PIC S9(4).
      02  F3A     PICTURE X.
      02  FILLER REDEFINES F3A.
       03  F3F    PICTURE X.
      02  F3I     PIC 9999V99.
      02  FILLER  PIC X.
  01  MAPO REDEFINES MAPI.
      02  FILLER PICTURE X(3).
      02  F1O     PIC X(30).
      02  FILLER  PIC X.
      02  FILLER PICTURE X(3).
      02  F2O     PIC $$$,$$0.00.
      02  FILLER  PIC X.
      02  FILLER PICTURE X(3).
```

```
         02  F30      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 programs.

PS=
specifies the programmed symbol set to be used for the display of the field.

<u>BASE</u>
specifies that only the basic symbols are used.

psid
specifies a single EBCDIC character or a hexadecimal code on 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.

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.

<u>I/O OPERATIONS USING BMS MACROS</u>

Input and output operations using the facilities of BMS are requested by issuing DFHBMS macros. Parameters provided by the application program indicate whether an input or an output operation is needed, the name of the map to be used by BMS, and other information to control the mapping function. Control is passed to BMS, which performs any required input/output operations through terminal control.

Initial terminal input, which causes a task to be initiated, is stored in the initial TIOA of the task as a native-mode data stream. The initial input data can be mapped into a particular format by issuing a DFHBMS TYPE=MAP macro. The format of this initial input data must correspond to that of the requested map. Input data to be mapped from a 3270 must contain 3270 device-dependent code (in particular, the data stream must contain an SBA). Similarly, the DFHBMS TYPE=MAP macro can be used to map further input data, obtained by means of a terminal control READ request, into a particular format.

Alternatively, the DFHBMS TYPE=IN macro can be issued; this macro causes a terminal control READ/WAIT operation to occur, and the resulting terminal input is mapped into a particular format. The data returned from an input mapping operation is in TIOA format. The address of the TIOA containing the mapped data is placed in TCTTEDA for a TYPE=IN operation; for a TYPE=MAP operation, or an output operation, the address will be placed in the location (TCTTEDA or TCAMSIOA) used to specify the input data area. (See the section, "Addressing Input/Output Areas," below, for details of specifying input data areas.)

For an output mapping operation, if data is to be passed from the TIOA of an application program, the application program must have obtained, through storage control, a TIOA large enough to contain the symbolic storage definition of the map being used. Any fields for which data is not to be passed to the mapping operation must be set to nulls (X'00'); this is best achieved through use of the INITIMG=00 operand of the DFHSC TYPE=GETMAIN macro. The first position of a field to be sent must not contain a null; if it does, the field will be ignored.

Maps are defined in a map set, which permits the formatting of a page of output using one or more of the maps in the map set. If the map set has been placed in the CICS program library, the user should specify MAPSET=map-set-name and MAP=map-name in any DFHBMS macro requesting an operation in which the map is required. If preferred, the user may place the seven-character name of the map set at TCAMSMSN and the name of the map at TCABMSMN; the MAPSET=YES and MAP=YES operands inform BMS that the names have been supplied in this way.

Implied READ/WRITE

DFHBMS TYPE=IN or TYPE=OUT macros result in a terminal control READ or WRITE, respectively. Therefore, the user does not need to code any terminal control (DFHTC) macros.

However, nothing prevents the user from intermingling native mode and BMS operations. A DFHBMS TYPE=MAP macro can be used to format a native mode input TIOA. If a MAP operation is requested for input from an unformatted 3270 buffer, mapping is not performed and the unformatted native mode TIOA is returned to the application program.

It is nevertheless possible to use DFHBMS TYPE=MAP for the TIOA containing a transaction-initiating data stream. All that is necessary to do so is to format the screen with the preceding task.

Addressing Input/Output Areas

Before a task issues a DFHBMS TYPE=MAP, or any BMS output macro, the address of the data being passed must be set up in either TCTTEDA or TCAMSIOA. The rules for deciding which area to use are:

- If the task is not terminal-oriented, the address of the TIOA-like area being used must be put in TCAMSIOA. TCTTEDA cannot be referenced as the task has no TCTTE.

- If the task is terminal-oriented, but a TIOA is not being used, the address of the TIOA-like area containing the user data must be put into TCAMSIOA and TCTTEDA must be filled with binary zeros.

- If the task is terminal-oriented and the data is in a TIOA, the address of the TIOA may be put into either TCTTEDA or TCAMSIOA. If the address is put into TCAMSIOA, TCTTEDA must be filled with binary zeros. If the address is put into both TCTTEDA and TCAMSIOA, the address in TCTTEDA is used.

TCTTEDA is altered by BMS; the user must not assume that its contents are unchanged.

A BMS input operation places the data into a TIOA, and the address of the TIOA is returned in TCTTEDA.

Terminal-oriented tasks need not use actual TIOAs. Any task may pass data to BMS in any portion of dynamically acquired storage which looks like a TIOA in all respects except two:

- The storage class need not be terminal.

- The storage chain address need not refer to a TCTTE or other terminal storage.

Non-Terminal-Oriented Tasks

These tasks do not have a TIOA or a TCTTE; therefore such tasks cannot issue any BMS macros that use information in these areas. They can issue only DFHBMS TYPE=ROUTE, DFHBMS TYPE=PAGEBLD with a disposition of STORE or RETURN, and DFHBMS TYPE=TEXTBLD with a disposition of STORE or RETURN.

Technique for Setting TCTTEDA to Binary Zeros in PL/I

The NULL built-in function cannot be used to set TCTTEDA to binary zeros because this places hexadecimal 'FF' in the high-order byte of the address. Instead, the following statement can be used.

```
  UNSPEC(TCTTEDA)=32'0'B;
```

DFHBMS Macros

BMS macros are provided to enable the application programmer to:

- Map data that is already in a TIOA (without any terminal I/O taking place) (DFHBMS TYPE=MAP)

- Read in and map data from a terminal (DFHBMS TYPE=IN)

- Cumulatively build one or more pages of output data using maps (DFHBMS TYPE=PAGEBLD)

- Cumulatively build one or more pages of output data without using maps (DFHBMS TYPE=TEXTBLD)

- Terminate the accumulation of output data that has been logically combined and write it to an output device (DFHBMS TYPE=PAGEOUT)

- Write data (without accumulation) to an output device (DFHBMS TYPE=OUT)

- Discontinue the process of building a logical message (DFHBMS TYPE=PURGE)

- Define the terminal(s) or operator(s) that are to receive an output message (DFHBMS TYPE=ROUTE)

- Check the response to a BMS request (DFHBMS TYPE=CHECK).

In the sections that follow, the syntax of each type of DFHBMS macro is shown, and the use of the macro is explained. Parameters of the TYPE= operand are discussed separately under each macro. Descriptions of all other operands for the DFHBMS macros are gathered into a single section, arranged in alphabetical order, at the end of the chapter.

Output Operations

There are a variety of ways in which the various DFHBMS macros can be used, and combined, for output operations.

The simplest case is DFHBMS TYPE=OUT (without PAGEBLD or TEXTBLD). This macro results in a simple output operation similar to that resulting from a DFHTC TYPE=WRITE macro, but with a mapping operation probably, but not necessarily, included.

When an application programmer wishes to output data which may occupy more than one device output buffer he can build a single logical message using a series of DFHBMS TYPE=PAGEBLD macros (if he wants mapping to be included) or DFHBMS TYPE=TEXTBLD (if mapping is not required). When the logical message is complete, he terminates the process of accumulation and causes physical output to occur by issuing a DFHBMS TYPE=PAGEOUT macro.

The DFHBMS TYPE=ROUTE macro does not itself cause any output operation to occur; it defines the destination for ensuing BMS output macros. The effect of a ROUTE macro should be terminated by a PAGEOUT macro before another ROUTE macro is issued.

Output operations that do not send user-supplied data (TYPE=PAGEBLD, DATA=NO or TYPE=OUT, DATA=NO) do not require TIOAs.

INPUT MAPPING WITHOUT I/O (TYPE=MAP)

To request that data already in an input TIOA is mapped according to a specified map.

```
DFHBMS TYPE=(MAP[,SAVE])
       [,MAP={map-name|YES}]|
       [,MAPADR={symb-addr|YES}]
       [,MAPSET={mapset-name|YES}]|
       [,MSETADR={symb-addr|YES}]
       [,MAPFAIL=symb-addr]
       [,ERROR=symb-addr]
       [,INVMPSZ=symb-addr]
       [,NORESP=symb-addr]
```

TYPE=MAP
specifies an input mapping operation without any associated terminal I/O operation.

The application program must have placed the address of an input TIOA containing data to be mapped into TCTTEDA or TCAMSIOA. The data in the TIOA is positioned into a new TIOA using the map specified in the MAP operand of the DFHBMS macro instruction, but no terminal I/O operation occurs. An example of such a TIOA is the initial TIOA given to a transaction upon entering a transaction code. If data is included with the transaction code, the screen must have been formatted previously by another transaction, or the data is not mapped. The address of the new TIOA is returned to the application program in the location in which the original data area was specified (TCTTEDA or TCAMSIOA).

The following types of data are not mapped, but are left in the TIOA unaltered.

- Data from TCAM-supported 3600 or 3790
- Data from VTAM-supported 3600 or 3650 (except 3650 host conversation (3270) logical unit)
- Data from 3790
- Word processing data streams, that is, data received from a word processing medium type 1, 2, 3, or 4.

SAVE
When used with MAP, SAVE specifies that the user-supplied data area addressed by TCTTEDA or TCAMSIOA is not to be altered, and that a new TIOA is to be acquired for the operation. The address of the new TIOA is returned to the application program in the location in which the original data area was specified (TCTTEDA or TCAMSIOA).

The use of the SAVE operand merely stops CICS overwriting a data area that you want to retain. It is still necessary to store the address of any such area elsewhere, so that it can be accessed later, because the location containing the address is overwritten.

INPUT OPERATIONS WITH MAPPING (TYPE=IN)

To request BMS services for input operations, a DFHBMS macro of the following format is used:

```
DFHBMS TYPE=(IN[,SAVE][,TEXT])
     [,MAP={map-name|YES}]|
     [,MAPADR={symb-addr|YES}]
     [,MAPSET={mapset-name|YES}]|
     [,MSETADR={symb-addr|YES}]
     [,EOC=symb-addr]
     [,EODS=symb-addr]
     [,ERROR=symb-addr]
     [,INVMPSZ=symb-addr]
     [,MAPFAIL=symb-addr]
     [,NORESP=symb-addr]
     [,RDATT=symb-addr]
```

TYPE=IN
specifies an input mapping operation. Input is accepted from the terminal through a terminal control READ/WAIT request. The input data is then mapped into the TIOA and made available to the application program by placing the TIOA address at TCTTEDA. The fields entered as part of the input data stream are available to the application program under the field names specified in the DFHMDF macros by which they are defined, suffixed with the letter I to correspond to the name generated by CICS in the definition of the area.

The following types of data are not mapped, but are left in the TIOA unaltered.

- Data from TCAM-supported 3600 or 3790
- Data from VTAM-supported 3600 or 3650 (except 3650 host conversation (3270) logical unit)
- Data from 3790
- Word processing data streams, that is, data received from a word processing medium type 1, 2, 3, or 4.

If the terminal used for input is a printer (for example, a 2740), the output must be entered into the correct lines and columns as if it were a map on a screen. If the first field to be entered is defined as POS(3,16), two blank lines must be entered followed by 16 blanks (including one blank for the attribute byte which is the first character of the field), followed by the data for the field.

If DFHBMS TYPE=IN macros are used to read data from a VTAM batch logical unit, the inbound function management headers (FMHs) will be removed before the data is placed in the TIOA. If an FMH is required, the application programmer should issue a DFHTC TYPE=READ macro, deal with the FMH, and then issue a DFHBMS TYPE=MAP macro to map the data. Inbound FMHs are applicable only to batch logical units.

SAVE
when used with IN, specifies that the data area addressed by TCTTEDA is not to be altered; a new TIOA is to be acquired for the operation, and its address returned in TCTTEDA.

The use of the SAVE operand merely stops CICS overwriting a data area that you want to retain. It is still necessary to store the address of any such area elsewhere, so that it can be accessed later, because the location containing the address is overwritten.

TEXT
indicates that uppercase and lowercase characters are contained in the input data stream.

This parameter is used to override a FEATURE=UCTRAN specification in the DFHTCT macro used by the system programmer for the input terminal. (See the appropriate CICS Resource Definition manual.)

BUILDING OUTPUT PAGES USING MAPS (TYPE=PAGEBLD)

To build an output page cumulatively, using maps, the application program uses the DFHBMS TYPE=PAGEBLD macro. This causes the data in the area defined by a specified symbolic description map to be mapped according to the physical map. The mapped data is positioned within an area large enough to contain one page of output. The application programmer issues another DFHBMS TYPE=PAGEBLD macro to map and position the next portion of the output page. The mapping operation continues until the application program has completed the message to be sent to the terminal.

Because of CICS terminal paging facilities, the application programmer need not keep track of when a page is full. He can either let BMS force a new page automatically or include the OFLOW operand in the DFHBMS TYPE=PAGEBLD macros to cause BMS to transfer control to an overflow routine (which the programmer must provide) when a page of data cannot contain the data to be mapped.

The format of the DFHBMS TYPE=PAGEBLD macro is as follows:

```
DFHBMS TYPE=(PAGEBLD[,{OUT[,WAIT]|
            STORE|RETURN}]
       [,SAVE][,ERASE][,ERASEAUP]
       [,LAST])
       [,DATA={NO|YES|ONLY}]
       [,MAP={map-name|YES}]
       [,MAPSET={mapset-name|YES}]|
          [,MSETADR={symb-addr|YES}]
       [,CTRL=([PRINT][,{L40|L64|L80|
        HONEOM}]
       [,FREEKB][,ALARM][,FRSET])]
       [,CURSOR={number|YES}]
       [,FMHPARM={parameter|YES}]
       [,LDC={mnemonic|YES}]
       [,PROPT=NLEOM]
       [,REQID={prefix|YES}]
       [,ERROR=symb-addr]
       [,IGREQID=symb-addr]
       [,INVLDC=symb-addr]
       [,INVMPSZ=symb-addr]
       [,INVREQ=symb-addr]
       [,NORESP=symb-addr]
       [,OFLOW=symb-addr]
       [,RETPAGE=symb-addr]
       [,TSIOERR=symb-addr]
       [,IGREQCD=symb-addr]¹
       [,WRBRK=symb-addr]²

¹ ASM only
² CICS/OS/VS only
```

where:

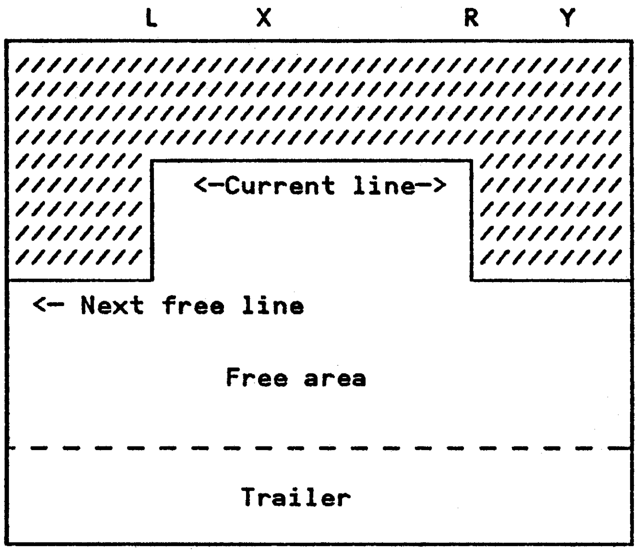
TYPE=PAGEBLD
indicates that one page of data is to be accumulated and formatted from data submitted through multiple PAGEBLD requests. In each PAGEBLD request, a map defines the number of lines and columns that the data is to occupy on the page. When a page is complete, as defined by one or more maps, it is written according to an OUT, STORE, or RETURN disposition.

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 (L), and next column from right (R).

Three other pointers are maintained that are relevant to map placement though they do not affect the area available: left reference column (X) and right reference column (Y), which are used when COL=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 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 nontrailer map, this is immediately above the trailer area; for a trailer map, it is 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 PAGEBLD, 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 nontrailer 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 in the leg of the inverted T, 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 COL=number was specified, by setting the left reference column to the specified column number.

Page Building Examples

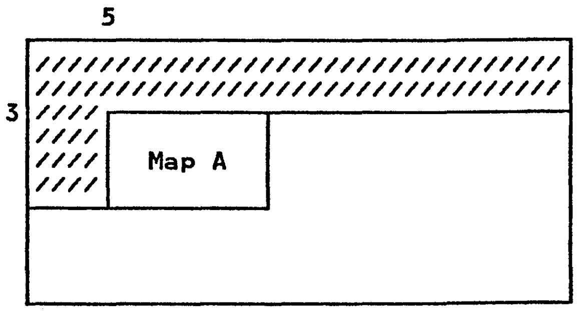
The effects of the mechanisms described above are illustrated by the following examples. The examples show the interactions between SIZE, LINE, COLUMN, and JUSTIFY=LEFT or RIGHT; header and trailer maps and JUSTIFY=FIRST or LAST are not brought into the examples.

In processing a PAGEBLD request, 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 PAGEBLD request. "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.

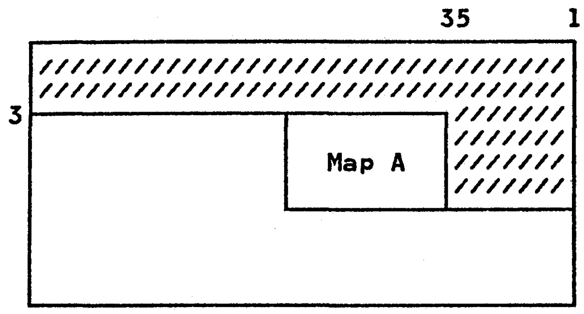
2. When JUSTIFY=LEFT is coded (or applied by default), all columns to the left of the leftmost map column, for the full depth of the map, are unavailable.

```
MAPA DFHMDI   ...,LINE=3,COL=5,
                  JUSTIFY=LEFT,...
```



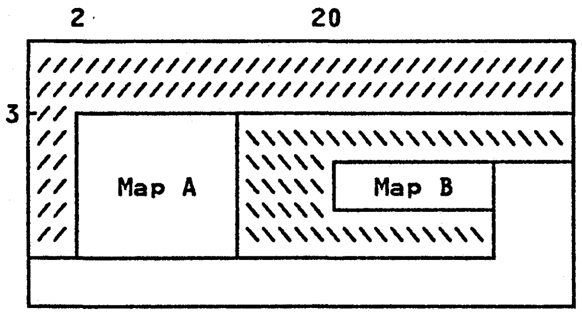
3. When JUSTIFY=RIGHT is coded, all columns to the right of the rightmost map column, for the full depth of the map, are unavailable.

```
    MAPA DFHMDI   ...,LINE=3,COLUMN=35,
                      JUSTIFY=RIGHT,...
```

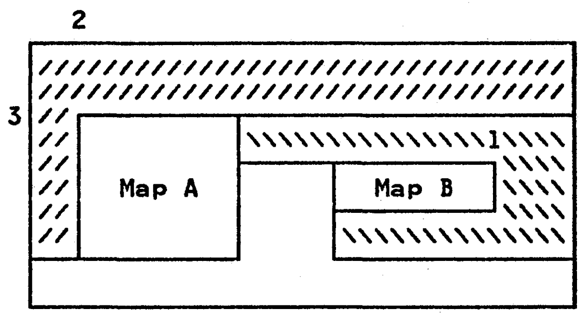


4. When two or more maps are placed so that they share certain lines, all columns beneath a map that ends higher are unavailable to the depth of the map that ends lowest. Similarly unavailable are all columns to the left (if the higher map is left justified) or to the right (if the higher map is right 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,...
```

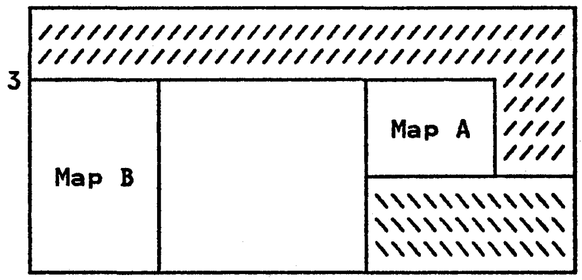


Figure 18 on page 173 shows the effect of several different maps on one page.

If BMS discovers that an area of the page directly specified for a map has already been used by a previous map, it raises the overflow condition, described below under "PAGEBLD Overflow Processing."

Handling Returned Pages

Whenever one or more pages have been completed and the programmer has specified TYPE=RETURN, TCAMSRLA contains the address of a list of completed pages. Since more than one page of output may result from a single BMS output request, there may be more than one entry in the list for a given terminal type. All entries for a particular terminal type immediately follow one another in the list. The list is laid out as shown in (a) of Figure 19 on page 174.

The page buffer pointer points to an area of USER-class storage which has a 12-byte prefix similar to that of a terminal input/output area (TIOA), as shown in (b) of Figure 19.

At this point, page buffers are on the USER-class storage chain and are disassociated from BMS control blocks; it is therefore the user's responsibility to release page buffers

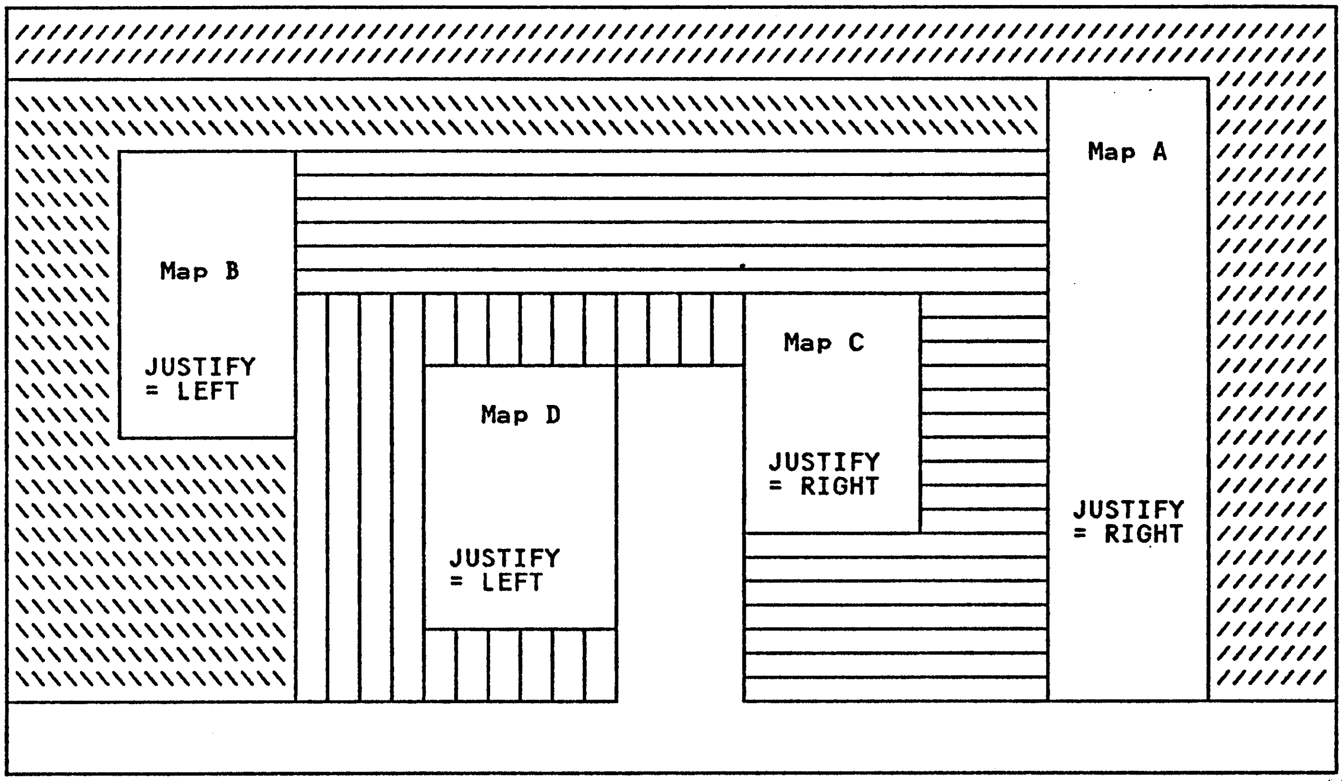


Figure 18. Map Positioning for More than One Map

when they are no longer needed. The storage containing the list of buffers should not be freed by the programmer; it is the intention of BMS to reduce processing time by reusing the list. This list will be altered by the next BMS request. Therefore, the programmer must save the contents before issuing the next BMS request.

Subsequent output of pages should normally be done using BMS. The use of the DFHTC macro to handle the output of pages is not recommended. However, if a DFHTC TYPE=WRITE macro is used, storage must be obtained by a DFHSC TYPE=GETMAIN macro with the CLASS=TERM operand included, and the output pages moved to the TIOA so acquired. The DFHTC TYPE=WRITE macro can then be used to transmit the pages from this new TIOA.

When terminals of the 3270 Information Display System are used, the write control character (WCC) containing the CTRL specification can be found at TIOACLCR in the page buffer after addressability to the area has been established. (TIOACLCR is a defined field in DFHTIOA and is addressable if the buffer address is loaded into TIOABAR.)

PAGEBLD 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 TCAM and VTAM terminals 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 requests. The LDC mnemonic is passed to the user's overflow routine in TCAMSLDM, and the LDC numeric value is passed in TCAMSLDC. PAGEBLD overflow can occur on a logical message being built for a ROUTE environment. If the ROUTE environment was created with a route list containing more than one LDC mnemonic, the returned LDC mnemonic and numeric value is the first LDC mnemonic resolved in the route list.

The routine to which control is transferred 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 in the TIOA.

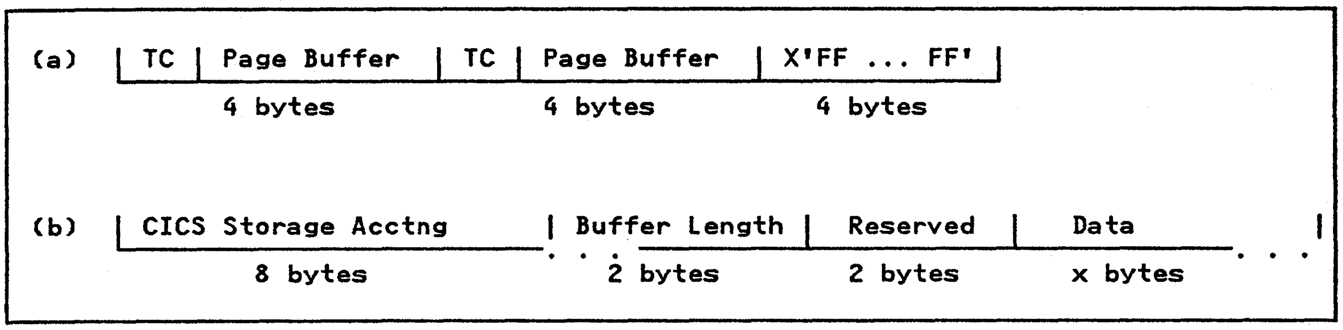


Figure 19. Page Address List

If a DFHBMS TYPE=ROUTE macro has not been previously issued, there is only one destination. If a DFHBMS TYPE=ROUTE macro has been issued, the logical message is probably being built for a multiple-destination environment. Since the application programmer has the capability of concurrently building pages for terminals that have different-sized output, overflow may occur at different times for different terminal groups.

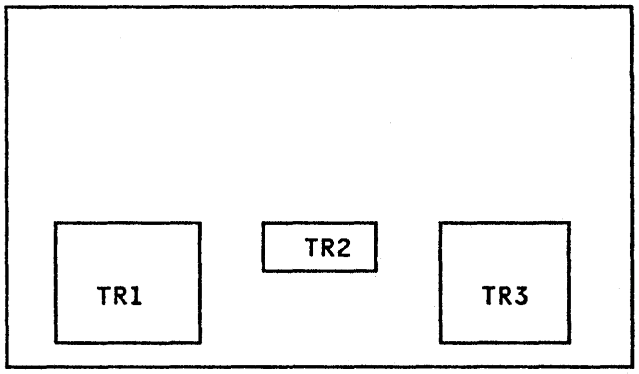
The overflow routine gets control every time any one of the destinations or groups of destinations encounters an overflow condition, that is, every time a specified map will not fit a page. The application program overflow routine must determine which destination or group of destinations has encountered the overflow.

Upon return to the application program from a DFHBMS TYPE=ROUTE macro, a count (relative to one) of the number of destinations or groups of destinations is available in TCAMSOCN. This overflow control count tells the application programmer how many overflow control areas (for example, accumulators) he may want to keep.

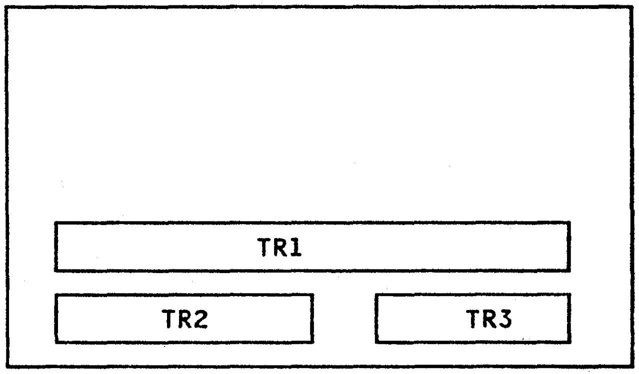
Whenever the overflow routine gets control, TCAMSOCN indicates the relative overflow control number of the destination that has encountered the overflow. This number indicates which control area should be output, perhaps through one or more trailer maps. In addition to the relative control count, BMS returns the current page number for the destination that has encountered the overflow. This page number is located at TCAMSPGN.

To place trailer data on a page, the programmer codes DFHBMS TYPE=PAGEBLD requests to process the trailer data. The map(s) 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. If the programmer tries to place more lines of trailer data on the page than are available, that trailer data is placed on a separate page by itself. Still another page is built to continue mapping with or without a header map.



No dummy trailer required.



Dummy trailer required.

To place header data on a page, the programmer codes DFHBMS TYPE=PAGEBLD request(s) to process the header data. The map(s) 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. JUSTIFY=FIRST is ignored if BMS is positioned at the top of a new page. If the programmer tries to place more header data on the page than the page can contain, multiple pages are created.

After overflow has been raised, the first map to be used in a TYPE=PAGEBLD request must be one that specified JUSTIFY=FIRST. Failure to do this will result in overflow being raised again immediately.

When all trailer and/or header data has been processed, the programmer must reissue the DFHBMS request that caused the overflow, since this data has not yet been mapped for all destinations.

If the user does not specify an overflow routine while issuing PAGEBLD requests, 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 OFLOW parameter would not be used.

A general overview of overflow processing is given in the flowchart in Figure 20 on page 176.

BUILDING OUTPUT PAGES WITHOUT USING MAPS (TYPE=TEXTBLD)

To request the building of pages of data without the use of maps, the application program issues DFHBMS TYPE=TEXTBLD macros. These macros cause BMS terminal paging to create pages containing application-program-supplied text data. The length of the data each macro is to process must be supplied in TIOATDL, prior to issuing the macro. Completion of a logical message is signaled by a DFHBMS TYPE=PAGEOUT macro. The beginning and ending of pages are handled by BMS and need be of no concern to the application program.

The format of the DFHBMS TYPE=TEXTBLD macro is as follows:

```
DFHBMS TYPE=(TEXTBLD[,{OUT[,WAIT]|
          STORE|RETURN}]
            [,SAVE][,ERASE][,LAST])
       [,HEADER={symb-addr|YES}]
       [,JUSTIFY={FIRST|LAST|
            line-number|YES}]
       [,TRAILER={symb-addr|YES}]
       [,CTRL=([PRINT][,{L40|L64|
            L80|HONEOM}]
        [,FREEKB][,ALARM])]
       [,CURSOR={number|YES}]
       [,FMHPARM={parameter|YES}]
       [,LDC={mnemonic|YES}]
       [,PROPT=NLEOM]
       [,REQID={prefix|YES}]
       [,ERROR=symb-addr]
       [,IGREQID=symb-addr]
       [,INVLDC=symb-addr]
       [,INVREQ=symb-addr]
       [,NORESP=symb-addr]
       [,RETPAGE=symb-addr]
       [,TSIOERR=symb-addr]
       [,IGREQCD=symb-addr]¹
       [,WRBRK=symb-addr]²

¹ ASM only
² CICS/OS/VS - 2741 only
```

where:

TYPE=TEXTBLD
indicates that (1) one page of output is to be formed from data submitted through multiple TEXTBLD requests, or (2) multiple pages of output are to be formed from one TEXTBLD request. When TEXTBLD is specified, no map is used. When no more data can fit on a page, the page is written according to the OUT, STORE, or RETURN disposition (see below), and another page is started if necessary.

DIRECT OUTPUT (TYPE=OUT)

An output request in which neither TEXTBLD nor PAGEBLD is specified can be issued by the application program. Such a request may cause multiple pages to be written as output, but multiple requests cannot be issued to accumulate and format data within one page. One map may be used to format data on one page, and that page may be written directly to the terminal (TYPE=OUT). The rules governing this type of output are as follows:

- Multiple requests cannot be accumulated to build one page, whether mapped or unmapped.
- When using maps, one request cannot build more than one page.

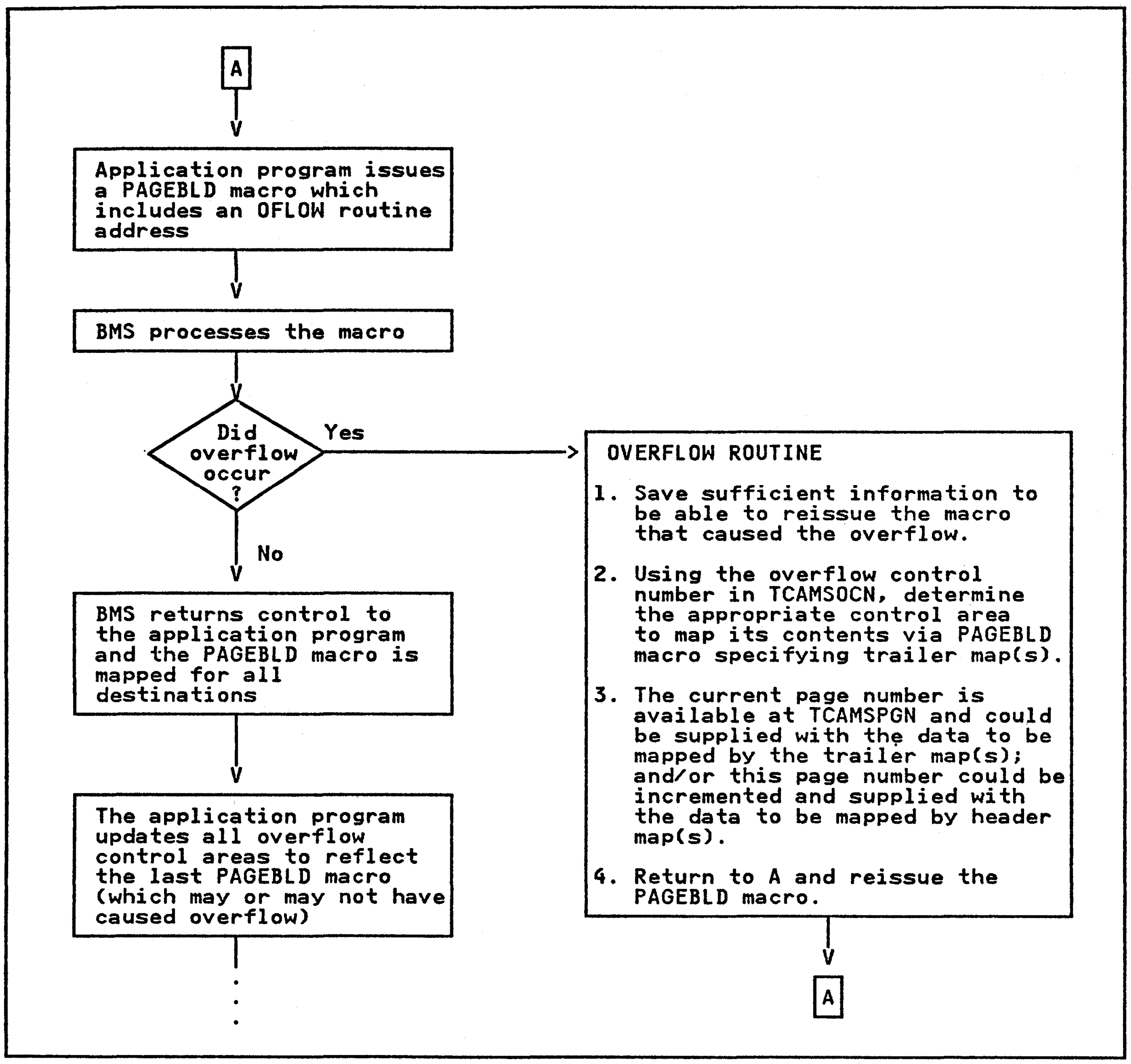

A
Application program issues a PAGEBLD macro which includes an OFLOW routine address
BMS processes the macro
Did overflow occur ?
Yes
No
OVERFLOW ROUTINE
1. Save sufficient information to be able to reissue the macro that caused the overflow.
2. Using the overflow control number in TCAMSOCN, determine the appropriate control area to map its contents via PAGEBLD macro specifying trailer map(s).
3. The current page number is available at TCAMSPGN and could be supplied with the data to be mapped by the trailer map(s); and/or this page number could be incremented and supplied with the data to be mapped by header map(s).
4. Return to A and reissue the PAGEBLD macro.
BMS returns control to the application program and the PAGEBLD macro is mapped for all destinations
The application program updates all overflow control areas to reflect the last PAGEBLD macro (which may or may not have caused overflow)
A

Figure 20. Overflow Processing by Application Programs under BMS

- When not using maps, a single request can result in more than one page.
- If the disposition is STORE, multiple requests can cause multiple pages (each request starting a new page) to be included in one logical message.
- For both mapping and nonmapping operations, if the disposition is STORE, a DFHBMS TYPE=PAGEOUT request must be issued to terminate the logical message.

The format of the DFHBMS TYPE=OUT macro is as follows:

```
DFHBMS TYPE=([{OUT[,WAIT]|STORE|
         RETURN}][,NOEDIT][,SAVE]
          [,ERASE][,ERASEAUP][,LAST])
       [,DATA={NO|YES|ONLY}]
       [,MAP={map-name|YES}]|
        ,MAPADR={symb-addr|YES}]  |
       [,MAPSET={mapset-name|YES}]|
       [,MSETADR={symb-addr|YES}]
       [,CTRL=([PRINT][,{L40|L64|
         L80|HONEOM]
         [,FREEKB][,ALARM][,FRSET])]
       [,CURSOR={number|YES}]
       [,FMHPARM={parameter|YES}]
       [,LDC={mnemonic|YES}]
       [,PROPT=NLEOM]
       [,REQID={prefix|YES}]
       [,ERROR=symb-addr]
       [,IGREQID=symb-addr]
       [,INVLDC=symb-addr]
       [,INVMPSZ=symb-addr]
       [,INVREQ=symb-addr]
       [,NORESP=symb-addr]
       [,RETPAGE=symb-addr]
       [,TSIOERR=symb-addr]
       [,IGREQCD=symb-addr]¹
       [,WRBRK=symb-addr]²

¹ ASM only
² CICS/OS/VS only
```

where:

TYPE=OUT
indicates that the output is to be written to the originating terminal at once if that terminal is to receive it.

Once a DFHBMS macro with OUT disposition has been issued, the application program must not issue a DFHSC TYPE=FREEMAIN,RELEASE=ALL macro until either a DFHBMS TYPE=PAGEOUT or DFHBMS TYPE=PURGE macro has been issued.

TERMINATING A LOGICAL MESSAGE (TYPE=PAGEOUT)

When the combining of pieces of data to form a logical message has been requested by means of DFHBMS TYPE=PAGEBLD or TYPE=TEXTBLD macros, such combining must be terminated by means of a DFHBMS TYPE=PAGEOUT macro. A logical message created by means of one or more noncumulative output requests with STORE disposition must be terminated by a DFHBMS TYPE=PAGEOUT macro.

The format of the DFHBMS TYPE=PAGEOUT macro is as follows:

```
DFHBMS TYPE=(PAGEOUT[,LAST])
       [,CTRL=([{PAGE|AUTOPAGE}]
        ,{RETAIN|RELEASE}])
       [,EODPURG={AUTO|OPER}]
       [,FMHPARM={parameter|YES}]
       [,TRAILER={symb-addr|YES}]
       [,TRANSID=transaction code]
       [,WRBRK={symb-addr|CURRENT|
          ALL}]
       [,ERROR=symb-addr]
       [,NORESP=symb-addr]
       [,RETPAGE=symb-addr]¹
       [,IGREQCD=symb-addr]
       [,TSIOERR=symb-addr]

¹ ASM only
```

where:

TYPE=PAGEOUT
specifies the termination of a logical message. No data is formatted in response to this request. Any remaining data in the page buffer is processed according to the OUT, STORE, or RETURN described in the previous macro. If a logical message is being built for a routing environment, PAGEOUT completes the logical message under route. An additional PAGEOUT macro is required to complete a logical message to the originating terminal.

If an error occurs during PAGEOUT processing, control is returned to the application program, and the RETAIN or RELEASE specifications are ignored. The logical message is not considered complete. The application program should either retry the PAGEOUT operation or PURGE the message.

Any logical message that has been started but not completed when a DFHSP (sync point) macro is issued is forced to completion by an implied TYPE=PAGEOUT macro.

DELETING A LOGICAL MESSAGE (TYPE=PURGE)

To discontinue the process of building a logical message, a DFHBMS TYPE=PURGE macro is issued. This instruction causes the portions of the message already built in main storage or on temporary storage to be deleted and returns control to the application program at the instruction following the DFHBMS TYPE=PURGE macro expansion. The TYPE=PURGE instruction is not to be used if TYPE=RETURN was used in the BMS PAGEBLD or TEXTBLD request.

The format of the DFHBMS TYPE=PURGE macro is as follows:

```
DFHBMS  TYPE=PURGE
```

where:

TYPE=PURGE
specifies that all data prepared for a logical message but not yet transmitted to a terminal is to be deleted from the system.

MESSAGE ROUTING (TYPE=ROUTE)

A DFHBMS TYPE=ROUTE request defines the terminal and/or operator to receive the message created by subsequent DFHBMS output requests. The message may be directed to any or all BMS-supported terminals. The ROUTE macro defines the destination of the message; it does not cause transmission to occur. The ROUTE macro must be followed by one or more BMS output macros. A DFHBMS TYPE=PAGEOUT request causes the logical message to be completed and terminates the effect of the DFHBMS TYPE=ROUTE macro.

If a ROUTE request followed by one or more BMS output requests is not terminated by a PAGEOUT request before a subsequent ROUTE request is issued or before the application program terminates, the message is forced to completion. Since the application program did not issue the PAGEOUT request, BMS applies the PAGEOUT defaults to the message. A ROUTE request may be issued immediately following another ROUTE request. In this case, the first ROUTE request is nullified, and the second one determines the routing environment.

A message is considered undeliverable to a destination if it cannot be delivered within a certain interval after the requested delivery time. This interval is specified in the PRGDLAY operand of the DFHSIT PROGRAM=BMS macro by the system programmer. If the PRGDLAY operand is not included, no action is taken for undelivered messages and the message awaits delivery indefinitely. If PRGDLAY is specified, the transient data destination CSMT is notified of the number of undeliverable messages purged for a destination; the application programmer can ensure that additional documentation is provided for an undeliverable message by including the ERRTERM operand in the DFHBMS TYPE=ROUTE macro.

Examples of situations causing undeliverable messages might occur, for example, when a message is routed to a terminal that is out of service, or when an operator identification is specified with a terminal identification and that operator is not signed on that terminal at the time the message is to be delivered.

Under CICS/DOS/VS only, operating in a DL/I environment, if it is required to route a message to more than 40 terminals, several TYPE=ROUTE macros must be issued, each with a LIST operand that specifies a list of terminals with no more than 40 entries. Each TYPE=ROUTE macro must be issued with all other DFHBMS macros relevant to the message.

The format of the DFHBMS TYPE=ROUTE macro is as follows:

```
DFHBMS TYPE=ROUTE
       [,ERRTERM={termid|ORIG|YES}]
       [,LIST={symb-addr|YES|ALL}]
       [,OPCLASS={decimal-value,
         ...|YES}]
       [,TITLE={symb-addr|YES}]
       [,INTRVAL={numeric-value|
         YES}]|[,TIME=
       {numeric-value|YES}]
       [,LDC={mnemonic|YES}]
       [,PROPT=NLEOM]
       [,REQID={prefix|YES}]
       [,ERROR=symb-addr]
       [,IGREQID=symb-addr]
       [,INVET=symb-addr]
       [,NORESP=symb-addr]
       [,RTEFAIL=symb-addr]
       [,RTESOME=symb-addr]
```

where:

TYPE=ROUTE
specifies the initiation of an output page routing operation.

Disposition and Message Routing

A routed logical message can be built using either of two dispositions: STORE or RETURN. The first BMS output request issued following the ROUTE request (with some exceptions noted below) determines the disposition of the logical message. This first request may specify STORE or RETURN; if neither is specified, the default is STORE. Once established, the disposition remains unchanged until the logical message is completed (PAGEOUT). It need not be repeated for subsequent requests. An output request specifying a disposition that is not in effect results in a return code of INVREQ.

A disposition of STORE is the normal disposition and finally results in the message either being delivered or deleted. A disposition of RETURN causes the routed logical message to be returned to the application program. It is the responsibility of the application program to deliver the logical message.

A task can converse with the terminal to which it is currently attached (assuming the task is terminal-oriented) during the time that it is building the logical message. That 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 input requests (DFHBMS TYPE=IN or TYPE=MAP) are encountered while the message is being built, they are processed as usual. To transmit output to the direct terminal while the routed logical message is being built, the task can issue non-TEXTBLD, non-PAGEBLD requests with an explicit disposition of OUT. The disposition of OUT isolates the output request to the direct terminal from the requests that are building the routed logical message.

The following points summarize the rules for conversation with the direct terminal while a routed logical message is being built:

- OUT must be specified in any output request that is to go to the direct terminal.
- TEXTBLD and PAGEBLD requests with a disposition of OUT are invalid and result in a return code of INVREQ.
- 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 had been issued from another terminal.

A list of "abridged" requests, in order of execution, is given below. The action taken by BMS for each is indicated.

- DFHBMS TYPE=OUT - Transmit to direct terminal.
- DFHBMS TYPE=ROUTE - Establish routing environment.
- DFHBMS TYPE=OUT - Transmit to direct terminal.
- DFHBMS TYPE=IN - Receive from direct terminal.
- DFHBMS TYPE=TEXTBLD - First output request eligible for routing establishes default disposition of STORE and TEXTBLD as mode of page building.
- DFHBMS TYPE=OUT - Transmit to direct terminal.
- DFHBMS TYPE=TEXTBLD,RETURN - INVREQ - routed logical message has already established a disposition of STORE.
- DFHBMS TYPE=TEXTBLD - Continue building routed logical message.
- DFHBMS TYPE=PAGEBLD,STORE - INVREQ - routed logical message being built with TEXTBLD requests cannot tolerate PAGEBLD request.
- DFHBMS TYPE=PAGEBLD,OUT - INVREQ - cannot issue PAGEBLD or TEXTBLD request to direct terminal while building a routed logical message.
- DFHBMS TYPE=TEXTBLD,STORE - Continue building routed logical message.
- DFHBMS TYPE=PAGEOUT - Terminate routed logical message and routing operation.
- DFHBMS TYPE=OUT - Transmit to direct terminal.

Status Flag Byte in User-Supplied Route List

Each route list entry contains a status flag byte used by BMS to indicate to the application program the status of the destination at the time the DFHBMS TYPE=ROUTE macro was issued. Upon return, the application program can investigate the status byte for each route list entry and take appropriate action.

ENTRY SKIPPED

A route list entry that is flagged as skipped was not included in the resolved routing environment. If an entry has been skipped, another flag indicating why the entry was skipped may be on in the status byte. This second flag could be **one** of the following:

- INVALID TERMINAL IDENTIFICATION
- TERMINAL NOT SUPPORTED UNDER BMS
- OPERATOR NOT SIGNED ON - **only** an operator identification was specified in the route list entry and that operator was not signed on **any** terminal
- OPERATOR SIGNED ON UNSUPPORTED TERMINAL
- INVALID LDC MNEMONIC

If **only** the ENTRY SKIPPED flag is on, neither a terminal identification nor an operator identification 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 IDENTIFICATION
This flag indicates that the terminal identification 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
This flag indicates that the terminal identification specified in the route list entry is for a terminal type that is not supported under BMS or the terminal table entry indicated that the terminal identification 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
This flag indicates that the specified operator is not signed on. Any one of the following conditions causes this flag to be set:

1. An operator identification was specified with a terminal identification, but the specified operator was not signed on the terminal. This entry is not skipped.

2. An operator identification was specified without a terminal identification, and the operator was not signed on any terminal. This entry is also flagged as ENTRY SKIPPED.

3. The OPCLASS operand was specified with the DFHBMS TYPE=ROUTE macro and a terminal identification was specified in the route list entry, but the operator signed on the terminal did not qualify under OPCLASS. 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 UNSUPPORTED TERMINAL
This flag indicates that only an operator identification 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 identification is returned in that route list entry at URLTRMID for informational purposes only. The settings are X'08' for ASM, 12-8-9 for COBOL, and 00001000 for PL/I.

INVALID LDC MNEMONIC
This flag indicates that one of the following conditions occurred:

1. The LDC mnemonic specified in the route list does not appear in the LDC list associated with the TCTTE.

2. 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.

A symbolic storage definition of the user-supplied route list is available on the source library(s) under the member name DFHURLDS. This symbolic storage 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 DFHBMS TYPE=ROUTE request that refers to a list. The symbolic base register is URLBAR. The settings are X'04' for ASM, 12-4-9 for COBOL, and 00000100 for PL/I.

CHECKING THE RESPONSE TO A BMS REQUEST (TYPE=CHECK)

```
DFHBMS TYPE=CHECK
       [,EOC=symb-addr]
       [,EODS=symb-addr]
       [,ERROR=symb-addr]
       [,IGREQID=symb-addr]
       [,INVET=symb-addr]
       [,INVLDC=symb-addr]
       [,INVMPSZ=symb-addr]
       [,INVREQ=symb-addr]
       [,MAPFAIL=symb-addr]
       [,NORESP=symb-addr]
       [,RETPAGE=symb-addr]
       [,RTEFAIL=symb-addr]
       [,RTESOME=symb-addr][1]
       [,IGREQCD=symb-addr]
       [,TSIOERR=symb-addr]

[1] ASM only
```

where:

TYPE=CHECK
indicates that the BMS response to a request for BMS services is to be checked.

Some response codes may appear in combination with other response codes. These combinations are: RTEFAIL and INVET, and RTESOME and INVET. The order used by BMS in checking for all conditions that the application programmer specifies is as follows:

NORESP, TSIOERR, INVREQ, RETPAGE, MAPFAIL, RTEFAIL, RTESOME, INVET, IGREQID, INVLDC, INVMPSZ, EODS, EOC, and ERROR. Thus, if the application programmer has specified INVET and RTEFAIL and both of these responses apply, BMS transfers control to the user-written exception-handling routine identified in the RTEFAIL operand. In this situation, the INVET operand is not acted upon.

BMS RESPONSE CODES

To test a BMS response code the application programmer must know the codes and their meanings. For this approach, the application programmer can access the response code(s) at TCAMSRC1, TCAMSRC2, and TCAMSRC3. Response codes and their associated conditions are shown in Figure 21 on page 182. The keywords are explained at the end of the chapter.

The examples in Figure 22 on page 183 show how to examine the response code provided by BMS at TCAMSRC1, TCAMSRC2, and TCAMSRC3, and transfer control to the appropriate user-written routine accordingly.

BMS MESSAGE RECOVERY

BMS provides message recovery for routed and nonrouted messages. To be recoverable, messages must satisfy the following requirements:

- The DFHBMS TYPE=STORE operand must have been specified on the BMS output requests that built the logical message.
- The BMS default REQID (**) or the specified REQID for the logical message must have been identified to 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 (TSP) and the interval control program (ICP) must also support recovery.

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 available at TCAMSRLA when the application programmer has specified that pages of output be returned (that is, RETURN is the disposition parameter in the output request). The code is available at TCAMSRI1 when an invalid map size (INVMPSZ) response is returned.

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 (without buffer receive)
G	2741
H	2740-2 (with buffer receive)
I	2770
J	2780
K	3780
L	3270 (40-column displays)
M	3270 (80-column displays)
P	Interactive LU (3767, 3770 Interactive); 3790 Full Function LU; and SCS Printer LUs (3270 and 3790)
Q	2980 Models 1 and 2
R	2980 Model 4
U	3601
V	Host Conversational (3653)
W	3650 User Program
X	3650/3270 Host Conver (3270)
Y	Batch LU (3770 Batch), Batch Data Interchange LU (3770, 3790, LUTYPE4)

STANDARD ATTRIBUTE LIST AND PRINTER

Control Characters (DFHBMSCA

The application programmer can obtain a set of commonly used 3270 field attributes and printer control characters by copying DFHBMSCA into his program. For COBOL, this definition must be copied into the working storage section. DFHBMSCA consists of a set of EQU statements in the case of assembler language, a set of 01 statements in the case of COBOL, and DECLARE statements defining elementary character variables in the case of PL/I. One possible use for DFHBMSCA is for the purpose of temporarily changing attribute characters in a map.

The field attributes/printer control characters and corresponding symbolic names are listed below. These attributes cannot be combined by the application programmer in any manner. If any combinations other than those listed are required, the application programmer must either use the ATTRB operand of the DFHMDF macro to obtain the desired combinations or generate new attribute combinations offline.

DFHBMS Service Request	Condition	Response Code			Response Code Location
		Assembler	COBOL	PL/I	
INPUT,OUTPUT, ROUTING,CHECK	NORESP (Normal response)	X'00'	LOW-VALUES	00000000	TCAMSRC1, TCAMSRC2, and TCAMSRC3
OUTPUT,CHECK	INVREQ (Invalid request)	X'01'	12-1-9	00000001	TCAMSRC1
OUTPUT,CHECK	RETPAGE (Return Page)	X'02'	12-2-9	00000010	TCAMSRC1
INPUT,CHECK	MAPFAIL (Mapping attempt failure)	X'04'	12-4-9	00000100	TCAMSRC1
INPUT,CHECK	EODS (End of data set)	X'04'	12-4-9	00000100	TCAMSRC3
INPUT,OUTPUT, CHECK	INVMPSZ (Invalid map size)	X'08'	12-8-9	00001000	TCAMSRC1
INPUT,CHECK	EOC (End of chain)	X'08'	12-8-9	00001000	TCAMSRC3
OUTPUT,CHECK	INVLDC (Invalid LDC mnemonic)	X'10'	12-11-1-8-9	00010000	TCAMSRC2
OUTPUT, ROUTING,CHECK	IGREQID (Ignore REQID specification)	X'10'	12-11-1-8-9	00010000	TCAMSRC3
ROUTING,CHECK	INVET (Invalid error terminal)	X'20'	11-0-1-8-9	00100000	TCAMSRC1
ROUTING,CHECK	RTESOME (Routing to only some terminals)	X'40'	No punches	01000000	TCAMSRC1
ROUTING,CHECK	RTEFAIL (Routing failure)	X'80'	12-0-1-8	10000000	TCAMSRC1
INPUT,OUTPUT ROUTING,CHECK	ERROR (Any response other other than NORESP)	See note	See note	See note	TCAMSRC1, TCAMSRC2, and TCAMSRC3
OUTPUT,CHECK	TSIOERR (Temporary storage I/O error)	X'80'	12-0-1-8	10000000	TCAMSRC2
OUTPUT,CHECK	IGREQCD (Request change direction ignored)	X'40'	No punches	01000000	TCAMSRC2

Note: The test for the ERROR response is satisfied by a **not equal** condition; that is, not X'00', not LOW-VALUES, or not 00000000 for assembler, COBOL, and PL/I, respectively.

Figure 21. BMS Response Codes

```
ASM:

       DFHBMS   TYPE=(TEXTBLD,STORE)     BUILD OUTPUT
       CLI      TCAMSRC1,X'00'           ANY UNUSUAL CONDITIONS, TEST 1
       BNE      ERROR                    ..YES, GO TERMINATE THE TASK
       CLI      TCAMSRC2,X'00'           ..NO, ANY UNUSUAL CONDITIONS, TEST 2
       BNE      ERROR                    ..YES, GO TERMINATE THE TASK
       CLI      TCAMSRC3,X'00'           ..NO, ANY UNUSUAL CONDITIONS, TEST 3
       BE       GOOD                     ..NO, GO CONTINUE PROCESSING
ERROR  DS       0H                       YES, TERMINATE THE TASK
       DFHPC    TYPE=ABEND               TERMINATE THE TASK
GOOD   DS       0H
       .
       .
       .
COBOL:

       DFHBMS TYPE=(TEXTBLD,STORE)      BUILD OUTPUT
       IF TCAMSRC1 NOT = ' ' THEN GO TO ERROR.
       IF TCAMSRC2 NOT = ' ' THEN GO TO ERROR.
       IF TCAMSRC3 = ' ' THEN GO TO GOOD.
ERROR.
       DFHPC TYPE=ABEND                 TERMINATE THE TASK
GOOD.
       .
       (the value specified within the quotes is an unprintable
        multipunch code for the hex value)
       .
PL/I:

       DFHBMS TYPE=(TEXTBLD,STORE)      BUILD OUTPUT
       IF TCAMSRC1 = '0'B & TCAMSRC2 = '0'B
           & TCAMSRC3 = '0'B THEN GO TO GOOD;
       ERROR:
       DFHPC TYPE=ABEND                 TERMINATE THE TASK
GOOD:
       .
       .
       .
```

Figure 22. How to Examine BMS Response Codes

Name	Attribute/Control Character
DFHBMPEM	3270 Printer end of message
DFHBMPNL	3270 Printer new-line char.
DFHBMASK	Autoskip
DFHBMUNP	Unprotected
DFHBMUNN	Unprotected and numeric
DFHBMPRO	Protected
DFHBMBRY	High intensity
DFHBMDAR	Dark, nonprint
DFHBMFSE	MDT on
DFHBMPRF	Protected and MDT on
DFHBMASF	Autoskip and MDT on
DFHBMASB	Autoskip and high intensity
DFHPS	Programmed symbols
DFHHLT	Highlighting
DFHERROR	Error character
DFHDFT	Default value
DFHDFCOL	Default color
DFHBLUE	Blue
DFHRED	Red
DFHPINK	Pink
DFHGREEN	Green
DFHTURQ	Turquoise
DFHYELLOW	Yellow
DFHNEUTR	Neutral
DFHBASE	Base PS
DFHDFHI	Default highlight
DFHBLINK	Blink
DFHREVRS	Reverse Video
DFHUNDLN	Underline
DFHMFIL	Mandatory fill
DFHMENT	Mandatory enter
DFHMFE	Mandatory fill and enter
DFHALL	Clear all settings
DFHCOLOR	Color
DFHVAL	Field validation

STANDARD ATTENTION IDENTIFIER LIST (DFHAID)

To test the method of initiating an incoming READ from the 3270 Information Display System, the application programmer is provided with a set of 3270 attention identifiers (single-character variables called AIDs) that can be used to test the value at TCTTEAID. He can obtain this set of attention identifiers by copying DFHAID into his program. For COBOL, this definition must be copied into the working storage section.

DFHAID consists of a set of EQU statements in the case of assembler language, a set of 01 statements in the case of COBOL, and DECLARE statements defining elementary character variables in the case of PL/I. The symbolic names for the attention identifiers and the corresponding 3270 functions are given as follows:

Name	3270 Function
DFHENTER	Enter key
DFHCLEAR	Clear key
DFHOPID	Operator Identification Card Reader
DFHPEN	Immediately detectable field
DFHPA1	PA1 key
DFHPA2	PA2 key
DFHPA3	PA3 key
DFHPF1	PF1 key
.	.
.	.
.	.
DFHPF24	PF24 key

PROGRAMMING CONSIDERATIONS FOR PAGING COMMANDS ON DISPLAY DEVICES

The commands used by terminal operators to communicate with CICS BMS are collectively known as terminal paging commands, or simply as paging commands. They are defined by the system programmer through the DFHSIT macro, which is described in the appropriate CICS Resource Definition manual. Their format and use are discussed in detail in the appropriate CICS-Supplied Transactions book.

The application programmer must be aware of the terminal paging commands in order to write applications that involve terminal operators. The use of BMS at map definition time and in executable programs can have a significant effect on terminal operator procedures.

It is important to note that when in a page retrieval session, that is, when using paging commands, all PA and PF keys are treated as paging commands, regardless of whether or not they have been defined in the SKRXXXX operand of the DFHSIT macro.

Cursor placement is an important consideration in programming for paging commands. Any of the following items can cause a paging command **not** to be the first data read by CICS and therefore not to be interpreted as a paging command.

- After a print operation on a 3270 display, 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 DATA=ONLY and no attribute byte in the TIOA is written into the buffer without an attribute byte. If the application programmer 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 hardware considers the data 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 DFHBMS macro 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'.

OPERANDS OF THE DFHBMS MACRO

CTRL=

PAGEBLD, TEXTBLD, and OUT Macros

In DFHBMS TYPE=PAGEBLD, TEXTBLD, and OUT macros, CTRL= is used to specify device characteristics related to terminals of the 3270 Information Display System (including VTAM 3270 logical units, 3650 host-conversational (3270) logical units, and 3790 (3270-display and 3270-printer) logical units). CTRL=ALARM is also valid for TCAM SDLC and VTAM-supported terminals (except interactive and batch logical units), for which all other parameters for CTRL are ignored.

To be effective, this operand must be specified in the DFHBMS TYPE=PAGEBLD macro that causes a page of output to be completed, or in the DFHMDI macro for the associated map, or in the DFHMSD macro for the associated map set. If the operand is specified in more than one of these macros, the specification in a DFHBMS macro will override that in a DFHMDI macro, which in turn overrides that in a DFHMSD macro.

If PROPT=NLEOM is specified, this operand is overridden; see the description of the PROPT operand later in this list of operands.

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 for 3270 displays without printer features.

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 cuses the default printer line length to be used.

FREEKB
specifies that the keyboard should be unlocked after this map is written out. If omitted, the keyboard remains locked; further data entry from the keyboard is inhibited until this status is changed.

ALARM
activates the 3270 audible alarm feature. For TCAM and VTAM terminals supporting function management headers (FMHs) (except interactive and batch logical units), ALARM signals BMS to set the alarm flag in the FMH.

FRSET
is valid only when mapping is used. FRSET indicates that the modified data tags (MDTs) of all fields currently in the 3270 buffer are to be reset to a not-modified condition (that is, field reset) before any map data is written to the buffer. This allows the DFHMDF ATTRB specification for the requested map to control the final status of any fields written or rewritten in response to a DFHBMS macro.

PAGEOUT Macro

In the DFHBMS TYPE=PAGEOUT macro, CTRL= specifies how pages are to be displayed at the terminal (when the disposition is OUT or STORE) and whether or not control is to be returned to the application program.

PAGE
specifies that pages are to be paged one at a time to the terminal. BMS writes the first page to the terminal when the terminal becomes available or upon request of the operator. All subsequent pages are written to the terminal in response to a terminal operator request. See the description of paging commands in the appropriate <u>CICS-Supplied Transactions</u> book. If automatic paging is specified for the terminal at system generation, this specification overrides the

automatic paging for this logical message. For TCAM SNA and VTAM-supported terminals, PAGE applies to all LDC page sets accumulated within the logical message.

AUTOPAGE
specifies that pages are to be paged automatically to the terminal. BMS writes each page of the logical message to the terminal when it becomes available. If paging upon request was specified for the terminal at system generation, this specification overrides it for this logical message, provided that the terminal is not a 3270 display terminal (AUTOPAGE cannot be specified for a 3270 display terminal). For TCAM SNA and VTAM-supported terminals, AUTOPAGE applies to all LDC page sets accumulated in the logical message.

A specification of PAGE for 3284 or 3286 devices is ignored. That is, AUTOPAGE is assumed for these devices. If neither PAGE nor AUTOPAGE is specified, the paging status specified for the terminal at system generation determines how pages are to be written to the terminal. For TCAM SNA and VTAM-supported terminals with LDC support, paging status for each LDC is obtained from the system LDC table.

RETAIN
indicates that BMS is to return control to the application program for further processing after it has written the page(s) to the terminal and has received data other than a purge, copy, or paging command from the operator.

RETAIN is intended to be used for a combination of page display from the page file (logical message built using the STORE disposition) and operator data entry. BMS issues a GET to the terminal after writing the appropriate page(s) to the terminal. BMS issues the GET **only** if the logical message was built with STORE disposition. If the logical message was **not** built with STORE disposition, BMS returns control to the application program after the last page is written to the terminal, and without issuing a GET 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 purged. The address of the newly acquired TIOA is in TCTTEDA. A chaining command is not valid at this point because it requests the creation of a new task for the terminal to which a task is already attached.

RELEASE
indicates that control is to be returned to the program at the next higher logical level after BMS has written the page(s) to the terminal. When RELEASE is specified, LAST is assumed for TCAM SNA and VTAM-supported terminals, except when the PAGEOUT is for a route operation.

Note: To ensure that a logical message appears at the receiving terminal at once, before any other transaction is initiated from the terminal and before any other messages that may have been routed to it, CTRL=RELEASE should be specified.

If neither RETAIN nor RELEASE is specified, and STORE is the disposition for the logical message, a new task is scheduled by CICS task control for writing the pages to the terminal, and control is returned to the application program at this time rather than after the pages are written. After the application program has terminated, the pages will be written to the terminal in response to terminal operator requests. See the description of paging commands in the appropriate CICS-Supplied Transactions book. If pages are being routed, a specification of either RELEASE or RETAIN is ignored.

If messages are being chained, and the second transaction uses BMS in paging mode, the use of RETAIN will prevent further chaining. RELEASE must be used to allow more than two transactions to be chained together.

CURSOR=
is used to position the cursor upon completion of a write operation to a 3270 device. This operand is valid in TYPE=OUT macros only when maps are used.

number
is an integer indicating a particular position relative to zero on the screen; the range of values that may be specified depends upon the screen size of the 3270 being used.

YES
indicates that a value indicating the desired cursor position has been placed in TCABMSCP. (Note, though, that TCABMSCP may be used by CICS for other purposes. The user should not rely on the cursor position specification remaining intact throughout a transaction.)

This operand overrides the IC option of the ATTRB operand of the DFHMDF macro, if it is specified in a macro that completes a pagebuilding operation and thus causes a write operation. Previous specifications of the IC option and of the CURSOR operand for the other maps making up the page are ignored.

Similarly, a CURSOR operand on a later TEXTBLD macro always overrides a CURSOR operand on an earlier TEXTBLD macro.

An alternate method may be used to dynamically position the cursor on the output screen. This method is called symbolic cursor positioning; it allows a field in the TIOA to be marked, symbolically, such that the cursor is placed under the first data byte of the field on the output screen.

Requirements for symbolic cursor positioning are as follows:

- MODE=INOUT must be specified on the DFHMSD macro for maps and DSECTs which will be used with symbolic cursor positioning.
- CURSOR=YES must be specified on the DFHBMS macro.
- Field TCABMSCP must be initialized with hexadecimal Fs; for example, MVC TCABMSCP,=X'FFFF'. (In COBOL move minus one into TCABMSCP which has been defined as PIC S9(4) COMP.)
- The length field, suffix "L", associated with the field under which the cursor is to be placed must be initialized with hexadecimal Fs. For example, MVC FIELD3L,=X'FFFF'. (In COBOL move minus one into FIELD3L which has been defined as PIC S9(4) COMP.)

The remainder of the TIOA may be built as desired by the user. Symbolic cursor positioning is operable only for devices which allow cursor placement to be performed independently of data placement; for example, 3604 and 3270. Symbolic cursor positioning is ignored for other devices.

DATA=
indicates one of the following three output mapping data selection modes.

NO
specifies that only default data is to be written from the selected map.

YES
specifies that data placed in the TIOA by the application programmer is to be 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.

ONLY
specifies that only data placed in the TIOA by the application programmer is to be written. The attribute characters (3270 only) must be specified for each field in the TIOA. Any default data or attributes from the map are ignored.

This operand is valid only when mapping is used. If it is omitted, DATA=NO is assumed. The first position of each field in data placed in the TIOA by the application program must contain a nonnull character. A suitable replacement character for a null character is a blank (X'40').

If this option is used to send data to a terminal defined in the TCT as supporting 3270 data stream extensions (color, programmed symbols, and extended highlighting), BMS transmits a data stream that modifies the existing fields. This data stream is valid only when the screen is formatted, so care must be taken not to send it to an unformatted screen on an extended data stream display unit. The problem does not arise with screens that do not support the extensions, because CICS then sends a data stream that simply overwrites the relevant part of the buffer.

EOC=symb-addr
specifies the symbolic address of the routine to be given control if the request/response unit (RU) is received, during a BMS input operation, with the end-of-chain indicator set. This operand is used only for VTAM interactive and batch logical units.

EODPURG=
specifies the manner in which CICS deletes the current message.

AUTO
specifies that CICS is to delete the message automatically if the operator enters a transaction that is not a paging command. Alternatively, the operator may delete the message with a purge command. See the description of the purge command in the appropriate CICS-Supplied Transactions book.

OPER
specifies that CICS is not to delete the message until the terminal operator explicitly requests deletion with a purge command.

Note: If temporary storage is reinitialized, all messages are lost, regardless of any other specifications.

EODS=symb-addr
indicates the label of a user-written routine to receive control if end-of-data-set (EODS) has been received during a BMS input operation. If this condition occurs, no data has been received (only a standalone function management header). No data is mapped and TCTTEDA is set to zero. This operand applies only to VTAM batch logical units.

ERROR=symb-addr
specifies the entry label of the user-written routine to which control is passed if any of the response conditions except NORESP occurs.

ERRTERM=
indicates the terminal to be notified if the message is purged because it is undeliverable. The message number, title identification, and destination of the message are indicated.

termid
is the terminal identification of the terminal to be notified.

ORIG
indicates that the originating terminal is to be notified.

YES
indicates that the terminal identification of the terminal to be notified has been placed in TCAMSTI prior to issuing the DFHBMS TYPE=ROUTE macro.

This operand is operative only if the PRGDLAY operand was specified in the DFHSG PROGRAM=BMS macro by the system programmer. If PRGDLAY was not specified, this operand has no effect.

FMHPARM=
specifies information to be included in a function management header (FMH) being transmitted to a 3650 logical unit. Refer to the appropriate CICS IBM 3650/3680 Guide for details of the FMH and of 3650 logical units.

This operand applies only to VTAM-supported 3650 logical units with outboard formatting. It specifies the name of the map to be used with this BMS request.

parameter
specifies the eight-character name of the map.

YES
indicates that the map name has been stored in the eight-character TCAMSFMP field

HEADER=
specifies that header data is to be placed at the beginning of each output page and points to that data.

symb-addr
is the symbolic address of the header record that will be used to place header information at the beginning of each page.

YES
indicates that the application programmer has placed the address of the header record in TCAMSHDR prior to issuing this DFHBMS macro.

If this operand is used in a DOS COBOL program, the label must not be longer than eight characters.

The record pointed to by HEADER or TRAILER operands has the following format:

```
|LL|P|C|<----Data------PPPPP------>|
```

where:

LL
: is a 2-byte field containing the length of the header or trailer information.

P
: is a one-byte field containing a character of the user's choice that indicates which, if any, are the embedded page number positions in the data area. The character chosen must, obviously, be one that does not otherwise appear in the data area. The embedded page number positions will initially contain this same character. The character must **not** be any of the following, which are reserved: X'0C', X'15', X'17', X'26', and X'FF'. If page-numbering is not required, P should be set to blank (X'40').

C
: is a reserved one-byte field.

Data and PPPPP
: is the header or trailer information to be placed at the beginning or end of each page of output. This information consists of a constant character string with, optionally, a page-number field of up to five characters embedded within it.

  The placement of the page-number field within the data area is entirely at the user's choice. If such a field is defined, BMS will place the current page number in it for each page built. The number is padded on the left with zeros if it does not fill the defined field; it is truncated on the left if it is too large for the defined field. Page numbering starts at 1 and can run up to 32,767. It is automatically reset to 1 after each DFHBMS TYPE=PAGEOUT request or if the output disposition is changed. The legibility of the code will be improved if the page-number field is separated from the constant data by blanks or other suitable characters, though such separation is not required by BMS.

  New-line characters (X'15') may be included in the constant data if a multiple-line header or trailer is required.

IGREQCD=symb-addr
: specifies the entry label of a user-written routine to which control is passed if an output operation is attempted after a signal command with a hard request change direction (RCD) code (X'00010000') has been received from an LUTYPE4 logical unit. Applies to output operations only. Valid in assembler language only.

IGREQID=symb-addr
: specifies the entry label of a user-coded routine to which control is to be passed if the prefix specified is different from the established (via a previous specification or default) REQID for this logical message.

INTRVAL=
: specifies the interval of time after which data being routed to the page file is to be transmitted to the terminal(s).

  numeric value
  : is of 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.

  YES
  : indicates that the interval of time has been placed in packed decimal form (0HHMMSS+) in TCAMSRTI prior to issuing the DFHBMS TYPE=ROUTE macro.

INVET=symb-addr
: specifies the entry label of the user-written routine to which control is passed if the terminal identification specified by the ERRTERM operand of a DFHBMS TYPE=ROUTE macro is invalid or is assigned to a terminal of a type not supported under BMS.

INVLDC=symb-addr
: specifies the entry label of the user-written routine to which control is passed if the LDC mnemonic specified by the LDC operand does not appear in the LDC list associated with the TCTTE.

INVMPSZ=symb-addr
: specifies the entry label of the user-written routine to which control is to be passed if (1) the specified map is too wide for a receiving terminal, or (2) OFLOW has been requested and the specified map is too long for the

receiving terminal. Upon entry to the user-written routine, TCAMSRI1 contains a terminal code that further identifies the receiving terminal (see "Terminal Code (TC) Table," earlier in this chapter).

INVREQ=symb-addr
specifies the entry label of the user-written routine to which control is passed if the request for BMS services is invalid.

This response may be caused by any of the following conditions:

- Changing the disposition of a routed logical message prior to its completion, through DFHBMS TYPE=PAGEOUT
- Issuing a separate TYPE=TEXTBLD or TYPE=PAGEBLD request to the direct (originating) terminal while in the process of building a routed logical message
- Mixing TYPE=TEXTBLD and TYPE=PAGEBLD requests when building a logical message
- Specifying NOEDIT with a TYPE=PAGEBLD or TYPE=TEXTBLD request
- Specifying the TRAILER operand with TYPE=PAGEOUT when terminating a logical message built using TYPE=PAGEBLD requests
- Issuing a DFHBMS request with DATA=YES or DATA=NO and specifying a map with no field specifications
- Issuing a DFHBMS request with TYPE=STORE from a CICS application program communicating with a host conversational (3653) logical unit.

JUSTIFY=
describes the positioning of the text data.

FIRST
indicates that this TEXTBLD data is to be positioned at the top of the page. Any partially formatted page from preceding DFHBMS requests is considered to be complete. If the HEADER operand is specified, the header precedes the TEXTBLD data.

LAST
indicates that this TEXTBLD data is to be positioned at the bottom of the page. If the TRAILER operand is specified, the trailer appears after the TEXTBLD data. The page is considered to be complete after the request is processed.

line number
indicates that this TEXTBLD data is to be positioned at line nnn of the page.

YES
indicates that the application programmer has placed a binary value from 1 to 255 in TCAMSJ prior to issuing this DFHBMS TYPE=TEXTBLD macro. A value in the range from 1 through 240 represents a line number; 254 represents LAST; and 255 represents FIRST. The values from 241 through 253 are reserved and should not be specified.

LDC=
specifies the mnemonic to be used by CICS to determine the logical device code that is to be used for the BMS operation and transmitted in the function management header (FMH) to the logical unit. This operand is meaningful only for TCAM and VTAM terminals with LDC support.

mnemonic
is the 2-character mnemonic used to determine the appropriate LDC numeric value. The mnemonic represents an LDC entry in the DFHTCT TYPE=LDC macro.

YES
indicates that the application program has placed the LDC mnemonic in TCAMSLDM.

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 operand of the DFHBMS macro is omitted, the LDC mnemonic specified in the DFHMSD macro is used, (except in TEXTBLD operations, when maps do not apply). If the

LDC operand has also been omitted from the DFHMSD macro, the action depends on the type of the logical unit.

For a 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 was specified in the DFHTCT TYPE=TERMINAL macro, or (1,40) if such a value was not specified.

For a batch 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. Note that 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 DFHBMS TYPE=ROUTE operations, the LDC operand of the ROUTE macro takes precedence over all other sources. If this operand is omitted and a route list is specified (LIST=symbolic address or YES), 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.

LIST=
specifies the terminals and/or operators to which paged data is to be directed.

symb-addr
is the label of a list of terminals and/or operators to which data is to be directed. If this parameter is used on a CICS/DOS/VS COBOL application program the label must not be longer than eight characters.

YES
indicates that the address of the list of terminals and/or operators to which data is to be directed has been placed in TCAMSRLA prior to issuing the DFHBMS TYPE=ROUTE macro.

ALL
indicates that all terminals supported by BMS are to receive the paged data.

There is a limit to the number of terminals to which a message can be sent. The maximum cannot be defined because it is dependent on the other operands specified on the routing command, but the transaction will be abended with an abend code of ABMC if the limit is exceeded.

The list of destination terminals and/or operators consists of 16-byte entries whose contents are as follows:

Bytes	**Contents**
1-4	4-character (including trailing blanks) terminal or logical unit id, or blanks
5-6	2-character LDC mnemonic for TCAM and VTAM terminals with LDC support, or blanks
7-9	operator id, or blanks
10	status flag for route entry See "Status Flag Byte in User-Supplied Route List," earlier in the chapter.
11-16	reserved; must contain blanks

The end of the list is designated as follows:

```
ASM:    DC AL2(-1)
COBOL:  PIC S9(4) COMP VALUE -1.
PL/I:   DCL FIXED BIN(15) INIT(-1);
```

It may be necessary for the application program to supply this list of destinations in noncontiguous areas called segments. If the list is supplied in segments, every segment except the last is terminated with (at least) an 8-byte entry as follows:

Bytes	**Contents**
1-2	ASM: DC AL2(-2) COBOL: PIC S9(4) COMP VALUE -2. PL/I: DCL FIXED BIN(15) INIT(-2);
3-4	reserved
5-8	chain address to the first entry of the next segment

The end of the list is designated as described above for an unsegmented list.

If, for any entry in the list,

1. The terminal identification is specified but the operator identification is omitted, the data is routed to that terminal

without regard to operator identification.

2. The operator identification is specified but no terminal identification is given, the data is routed to the "first" terminal at which the operator is signed on under the specified operator identification. The "first" is determined by the physical location of the terminal entry in the CICS terminal control table. If no operator is signed on under the specified operator identification when the DFHBMS TYPE=ROUTE macro is executed, the route list entry is ignored.

3. Both terminal identification and operator identification are specified, the data is routed to that terminal.

For either 2 or 3 above, the data is displayed only if the operator with the specified identification is signed on at the terminal when the data is ready to be displayed, or when the operator signs on after the data is ready to be displayed. Entries of all three types may be included in one segmented or unsegmented list.

It should be noted that the status flag in each route list entry is used to notify the application program of certain status conditions for that requested destination. Therefore, if the route list is contained within the application program and BMS alters the status flag, the application program can no longer be considered reentrant.

MAP=
specifies the name of the map to be used when mapping formatted pages.

map name
is the 1- through 7-character name of the map within a map set.

YES
indicates that the application programmer has placed the name of the map in TCABMSMN prior to issuing this DFHBMS macro. The name must be left-justified and padded with trailing blanks to 8 characters.

MAPADR=
specifies the address of the map to be used when mapping formatted pages. This operand is valid only when the map has been coded within an assembler language program.

symb-addr
is the 1- through 7-character symbolic label that has been assigned to the map.

YES
indicates that the application programmer has placed the address of the map in TCABMSMA prior to issuing this DFHBMS macro.

If MAPADR is specified, MAP, MAPSET, and MSETADR should not be used.

MAPFAIL=symb-addr
specifies the entry label of the user-written routine to which control is passed if the data to be mapped has a length of zero or does not contain a SBA (start buffer address) sequence. This response can occur only if TYPE=IN or TYPE=MAP is specified and data is mapped from a 3270 device. For TYPE=IN, the address of the erroneous TIOA is available at TCTTEDA. For TYPE=MAP, this address is wherever the user placed it prior to the request (either in TCTTEDA or TCAMSIOA).

MAPSET=
specifies the name of the map set to be used in the mapping operation.

map set name
is the 1- through 7-character name of the map set.

YES
indicates that the application programmer has placed the name of the map set in TCAMSMSN prior to issuing the DFHBMS macro. The name must be left-justified and padded with trailing blanks to 8 characters.

The map set established by this operand must reside in the CICS program library, and a corresponding entry for the map set must exist in the processing program table (PPT).

If MAPSET is coded, MAP must also be coded.

MSETADR=
specifies the address of the map set to be used in the mapping operation. This operand is valid only when the map has been coded within an assembler language program.

symb-addr
is the 1- through 8-character symbolic label that has been assigned to the map set.

YES
indicates that the application programmer has placed the address of the map set in TCAMSMSA prior to issuing this DFHBMS macro.

MAPSET and MSETADR are mutually exclusive operands. If MSETADR is coded, MAP must also be coded.

NORESP=symb-addr
specifies the entry label of the user-written routine to which control is passed if none of the other response conditions (whether checked for or not) occurs. NORESP signifies "normal response".

OFLOW=symb-addr
specifies the symbolic address of a routine to which control is to be transferred if the mapped data does not fit on the current page (see "PAGEBLD Overflow Processing," earlier in the chapter).

OPCLASS=
specifies the operator class or classes to which data is to be routed.

decimal value,...
consists of 1 or more decimal values in the range from 1 through 24, separated by commas, specifically identifying the operator class(es).

YES
indicates that values identifying operator classes have been placed in TCAMSOC (3-byte field) prior to issuing the DFHBMS TYPE=ROUTE macro.

A bit position corresponding to each value from 1 through 24 is established in a 3-byte field which is matched against the 3-byte operator class field in the CICS terminal control table terminal entry (TCTTEOCL) for a terminal. At least one pair of corresponding bits must match in order for the message to be routed to the terminal. The value in TCTTEOCL is set during sign-on according to the OPCLASS operand of the DFHSNT TYPE=ENTRY macro specified by the system programmer.

If data is to be routed to an operator class, the application programmer may do one of the following:

1. Specify OPCLASS and omit LIST. Data is routed to each terminal at which an operator is signed on with the specified OPCLASS at the time the DFHBMS macro is issued.

2. Specify OPCLASS and LIST=ALL. Data is routed to all terminals. However, it is not necessary for an operator to be signed on with the specified OPCLASS at the time the DFHBMS macro is issued.

In both cases, the data is not displayed on a terminal until an operator is signed-on with the specified OPCLASS. In general, LIST=ALL is specified with OPCLASS only when it is anticipated that someone will eventually sign on with the specified OPCLASS at every supported terminal.

If the application programmer specifies OPCLASS and LIST=symbolic address, and the list contains operator identifications, a specified operator identification overrides OPCLASS for that entry.

PROPT=NLEOM
requests BMS to build a logical message specifically for a 3270 printer or a 3270 display with the Printer Adapter feature. If used, this operand must be specified in the first DFHBMS macro for each logical message. If routing, this operand must be specified on the TYPE=ROUTE request. Specification of this operand overrides the CTRL operand, if present; CTRL=(PRINT,HONEOM,FREEKB,PRESET) is assumed.

Specification of this operand will cause the page to be formatted using new-line (NL) characters as for the other hard copy devices. An end-of-message (EM) character is placed at the end of the data. As the data is printed, a new-line character causes printing to continue on the next line. The end-of-message character terminates printing. The next print operation will start on a new line.

The following restrictions apply when using this parameter: buffer updating and attribute modification of fields previously written into the buffer are not allowed. BMS issues an ERASE with every write to the terminal.

When building a logical message, BMS will insert an NL character at the end of each line and an EM character at the end of the text. Each NL and the EM character occupies a 3270 buffer position;

therefore, to avoid possible wraparound due to excessive data in the buffer, the PGESIZE values defined in the DFHTCT system macro should be such that the remainder of the 3270 buffer will contain these additional characters.

This operand is ignored if the direct or a routing terminal is not a 3270 printer or display with the Printer Adapter feature.

RDATT=
specifies the address of a routine to receive control if the operator presses the ATTN key on a 2741 when input is being entered from the terminal in response to a DFHBMS TYPE=IN request. This operand can be specified only if 2741 Read Attention support, an option available under either CICS/DOS/VS or CICS/OS/VS, has been generated into the system (see "Read Attention" on page 121).

REQID=
specifies the prefix to be used with the temporary storage identification. The identification (including the prefix) is used by CICS when attempting message recovery.

BMS message recovery is provided for a logical message only if the STORE operand is specified in the BMS output request and if the logical end of task has been reached.

Only one prefix can be specified for each logical message. If the REQID operand is not specified, CICS assigns the prefix ** (two asterisks).

prefix
indicates the alphanumeric prefix to be used as the first 2 characters of a temporary storage identification.

YES
indicates that the prefix has been stored at TCAMSRID.

RETPAGE=symb-addr
specifies the entry label of the user-written routine to which control is passed if one or more completed pages are returned to the application program. This response can occur only if TYPE=RETURN is specified in the DFHBMS macro (see the description of TYPE=RETURN for further information).

RTEFAIL=symb-addr
specifies the entry label of the user-written routine to which control is passed if a DFHBMS TYPE=ROUTE request results in a null routing environment (that is, the message will be sent, by default, to only the originating terminal). (To determine why route list entries were skipped, refer to "Status Flag Byte in User-Supplied Route List" on page 179.)

RTESOME=symb-addr
specifies the entry label of the user-written routine to which control is passed if (1) some of the entries in the user-specified route list named in the LIST operand of a DFHBMS TYPE=ROUTE macro are excluded from the routing environment, or (2) LIST=ALL is specified and not all of the entries in the terminal control table are included in the routing environment. (To determine why some route list entries were skipped, refer to "Status Flag Byte in User-Supplied Route List" on page 179.)

TIME=
specifies the time of day at which data being routed to the page file is to be transmitted to the terminal(s).

numeric value
is of 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.

YES
indicates that the time of day has been placed in packed decimal form (0HHMMSS+) in TCAMSRTI prior to issuing the DFHBMS TYPE=ROUTE macro.

TITLE=
specifies the symbolic address of a record that contains a title to be associated with the logical message created under this routing environment.

symb-addr
is the symbolic address of the title length field that precedes the title in the title record. If this parameter is used in a CICS/DOS/VS COBOL program the label must not be longer than 8 characters.

YES
indicates that the address of the title length field in the title record has been placed in TCAMSTA prior to issuing the DFHBMS TYPE=ROUTE macro.

The title pointed to by the TITLE operand is displayed with the logical message ID when the terminal paging query command is entered. See the description of the page query command in the appropriate CICS-Supplied Transactions book. This title serves as an additional message identifier, displayed upon request with the message ID, not on the logical message. The value in the 2-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:

```
|X'001A'|MONTHLY INVENTORY REPORT|

 2-byte       24-byte
 length       title field
 field
```

TRAILER=
specifies that user-defined trailer data is to be placed at the foot of each page completed by the TEXTBLD macro in which the operand is coded, or at the foot of the last page if the operand appears on a PAGEOUT macro. The operand is ignored if no page is completed by the macro in which it appears. The operand is invalid in a PAGEOUT macro that is completing a message built using PAGEBLD macros; if TRAILER= is used in such circumstances, BMS returns an INVREQ return code.

The format of trailer data is the same as that for header data, described above (see "HEADER="). Page numbering can be accomplished automatically, as with header data.

symb-addr
is the symbolic address of the trailer record that will be used to place trailer data at the bottom of the last page. If this parameter is used in a CICS/DOS/VS COBOL program, the label must not be longer than 8 characters.

YES
indicates that the application programmer has placed the address of the trailer record in TCAMSTRL prior to issuing this DFHBMS macro.

TRANSID=transaction code
specifies a 1- through 4-character transaction identification to be used with the next input message entered from the terminal to which this task is attached.

This operand is valid only when CTRL=RELEASE is specified.

TSIOERR=symb-addr
specifies the entry label of the user-written routine to which control is to be passed if an unrecoverable temporary storage input/output error occurs.

TYPE=
The following TYPE= parameters distinguish macros with distinct purposes. As such, they are not treated as operands and so described in this section; instead, they are explained individually in earlier sections in this chapter.

CHECK	PAGEOUT
IN	PURGE
MAP	ROUTE
PAGEBLD	TEXTBLD

The SAVE and TEXT parameters have special meaning for TYPE=IN (both) and TYPE=MAP (SAVE only) macros, and are described with the individual macros.

The following four TYPE= parameters indicate the disposition of output data:

OUT
indicates that the output is to be written to the originating terminal when the page is complete.

Once a DFHBMS macro with OUT disposition has been issued, the application program must not issue a DFHSC TYPE=FREEMAIN,RELEASE=ALL macro until either a DFHBMS TYPE=PAGEOUT or DFHBMS TYPE=PURGE macro has been issued.

When the OUT parameter is not preceded by either PAGEBLD or TEXTBLD, it effectively distinguishes a macro with a different purpose. This usage is described earlier in this chapter under the heading "Direct Output (TYPE=OUT)" on page 175.

RETURN
indicates that the complete page(s) is to be returned to the application programmer. (See "Handling Returned Pages," earlier, for further information.) The application program regains control (1) immediately following the BMS instruction if the current page is not yet completed, or (2) at an alternative entry point specified through the RETPAGE operand of this macro if one or more pages have been completed.

STORE
indicates that the output is to be placed in temporary

storage to be displayed in response to paging commands entered by the terminal operator. For more information about these commands, see the appropriate CICS-Supplied Transactions book. If STORE is specified with a REQID that is defined in the temporary storage table (TST), CICS provides message recovery for logical messages if the task has reached logical end. CICS temporary storage is needed to hold messages awaiting delivery to terminals.

WAIT
indicates that BMS is to wait until all output operations are complete before returning control to the application program. WAIT must be specified with every output request except the following:

- The last output request prior to task termination
- The last output request prior to an input operation
- The last output request prior to issuing a DFHBMS TYPE=PAGEOUT macro that precedes task termination or an input operation.

If no disposition is specified, the output is sent to the originating terminal. Once the disposition has been established for a logical message, it is not necessary to repeat the disposition for that logical message. Any change of disposition specified while in the process of building a logical message forces that logical message to completion with its original disposition. Then a new logical message is started with a new disposition. The disposition parameter is handled differently under DFHBMS TYPE=ROUTE. See "Disposition and Message Routing" on page 178.

The remaining TYPE= parameters are:

ERASE
specifies that a 3270 buffer or 3604 screen is to be erased before this page of output is displayed. A printer buffer will contain meaningless data from prior messages if all positions are not filled with current data. 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 all unprotected character locations in a 3270 buffer are to be erased before this page of output data is displayed. There are no further effects of specifying this parameter.

LAST
signals to CICS that this is the last output for a transaction and, therefore, the end of a bracket operation. This operand is meaningful only for TCAM SNA terminals and for VTAM-supported terminals and is applicable only when OUT is the specified disposition. For TCAM, an indicator is set in the communication control byte (CCB) requesting that the message handler send end-of-bracket.

NOEDIT
specifies that CICS need not insert device-dependent control characters (carrier return, line feed, idle characters, and so on) into the output data stream. The application program, therefore, assumes responsibility for providing any required control characters. This parameter is ignored for all output operations specifying maps. This parameter cannot be used with 3601 devices.

SAVE
specifies that the user-supplied data area addressed by TCTTEDA or TCAMSIOA is to be saved. The location containing the address of the data area will be changed by BMS, so the address should be stored elsewhere before issuing the macro.

WRBRK=
is used to specify the action that is to occur if the ATTN key on a 2741 is pressed while data is being written to the terminal.

symb-addr
specifies the symbolic address of the routine to receive control when the ATTN key on a 2741 is pressed during the actual write to the terminal.

This operand is operative when 2741 Write Break support has been generated into CICS (available only under OS/VS) and when the task would normally have regained control. It is not valid on BMS macros where TYPE=STORE or TYPE=RETURN is specified, or on a PAGEOUT macro when CTRL=RELEASE is specified.

CURRENT
specifies that transmission of the current page to the terminal is to cease, but, if autopaging has been requested, transmission of the next page (if any) begins.

ALL
specifies that transmission of the current page to the terminal is to cease and that no additional pages are to be transmitted. The logical message is purged.

Both CURRENT and ALL are meaningful only if 2741 Write Break support has been generated into CICS (available only under OS/VS), and if TYPE=STORE was specified in preceding DFHBMS requests, or data has been sent to terminals other than the originating terminal. In these cases, data has been placed in temporary storage and is being displayed by a program other than the one associated with the originating terminal.

CHAPTER 4.4. BATCH DATA INTERCHANGE (DFHDI MACRO)

The CICS Batch Data Interchange program provides for communication between an application program and a named data set (or destination) or a selected output medium. The named data set (or destination) must be part of a batch data interchange logical unit in an outboard controller; the selected output medium must be part of either such a logical unit or an LUTYPE4.

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 appropriate CICS/VS IBM 3767/3770/6670 Guide and the appropriate CICS/VS IBM 3790/3730/8100 Guide.)

The batch data interchange macro (DFHDI) is used to specify ADD, ERASE, REPLACE, QUERY, END, ABORT, SEND, RECEIVE, and CHECK operations on data sets in an outboard controller. Where the controller is an LUTYPE4 logical unit, only the END, ABORT, SEND, RECEIVE, and CHECK operations are supported.

The DFHDI macro can be used only with assembler language application programs. It is not available for COBOL or PL/I programs, which must use the command level interface if they require these facilities.

ADDITION OF RECORDS TO A DATA SET (TYPE=ADD)

```
DFHDI  TYPE=(ADD[,{SAVE|NOSAVE}]
       [,{WAIT|NOWAIT}]))
        ,DNADDR={symb-addr|YES}
       [,NUMREC={integer|YES}]
       [,DEFRESP=YES]
       [,VOLADDR={symb-addr|YES}]
       [,NORESP=symb-addr]
       [,FUNCERR=symb-addr]
       [,SELNERR=symb-addr]
       [,UNEXPIN=symb-addr]
```

This macro specifies that a record in the current TIOA, as indicated by the TCTTEDA, is to be added to the sequential or keyed direct data set corresponding to the destination name specified in the DNADDR operand.

The SAVE parameter specifies that the contents of the TIOA is to be saved; however, there is no guarantee that TCTTEDA will remain unchanged.

The WAIT parameter indicates that task activity is to be suspended until the DFHDI macro has been executed.

DELETION OF RECORDS FROM A DATA SET (TYPE=ERASE)

```
DFHDI  TYPE=(ERASE[,{WAIT|NOWAIT}])
        ,DNADDR={symb-addr|YES}
       {,KEYADDR={symb-addr|YES}|
        ,RRNADDR={record-id|YES}}
       [,DEFRESP=YES]
       [,VOLADDR={symb-addr|YES}]
       [,NORESP=symb-addr]
       [,FUNCERR=symb-addr]
       [,SELNERR=symb-addr]
       [,UNEXPIN=symb-addr]
```

This macro specifies that a record, identified by the KEYADDR or RRNADDR operand, is to be deleted from the keyed direct data set corresponding to the destination name specified in the DNADDR operand.

The WAIT parameter indicates that task activity is to be suspended until the DFHDI macro has been executed.

REPLACEMENT OF RECORDS IN A DATA SET (TYPE=REPLACE)

```
DFHDI  TYPE=(REPLACE[,{SAVE|NOSAVE}]
       [,{WAIT|NOWAITIT}])
        ,DNADDR={symb-addr|YES}
       {,KEYADDR={symb-addr|YES}|
        ,RRNADDR={record-id|YES}}
       [,NUMREC={integer|YES}]
       [,DEFRESP=YES]
       [,VOLADDR={symb-addr|YES}]
       [,NORESP=symb-addr]
       [,FUNCERR=symb-addr]
       [,SELNERR=symb-addr]
       [,UNEXPIN=symb-addr]
```

This macro specifies that a record identified by the RRNADDR or KEYADDR operand, in the current TIOA, is to replace a record in the addressed direct data set corresponding to the destination name specified in the DNADDR operand.

Where more than one record is to be replaced, the second and subsequent records are replaced consecutively, starting with the one specified in the RRNADDR or KEYADDR operand. The number of records to be replaced is specified in the NUMREC operand.

The SAVE parameter specifies that the contents of the TIOA are to be saved; however, there is no guarantee that TCTTEDA will remain unchanged.

The WAIT parameter indicates that task activity is to be suspended until the DFHDI macro has been executed.

INTERROGATION OF DATA SET (TYPE=QUERY)

```
DFHDI TYPE=QUERY
       ,DNADDR={symb-addr|YES}
      [,VOLADDR={symb-addr|YES}]
      [,NORESP=symb-addr]
      [,FUNCERR=symb-addr]
      [,SELNERR=symb-addr]
      [,UNEXPIN=symb-addr]
```

This macro specifies that the name of the data set corresponding to the destination name specified in the DNADDR operand is to be solicited to allow the outboard batch program to transmit the data set to the host. The program must issue input requests to receive the records from the data set.

TERMINATION OF OPERATIONS ON A DATA SET (TYPE=END)

```
DFHDI TYPE=END
      {{,DNADDR={symb-addr|YES}}
      |{,SELECT={(CONSOLE|PRINT|CARD|
          WPMEDIA1|WPMEDIA2|
          WPMEDIA3|WPMEDIA4[,nn])|
          YES}}}
      [,VOLADDR={symb-addr|YES}]
      [,NORESP=symb-addr]
      [,FUNCERR=symb-addr]
      [,SELNERR=symb-addr]
      [,UNEXPIN=symb-addr]
```

This macro specifies that operations on a data set are to be terminated normally. The current outboard destination is deselected normally.

ABNORMAL TERMINATION OF OPERATIONS ON A DATA SET (TYPE=ABORT)

```
DFHDI TYPE=ABORT
      {{,DNADDR={symb-addr|YES}}
      |{,SELECT={(CONSOLE|PRINT|CARD|
          WPMEDIA1|WPMEDIA2|
          WPMEDIA3|WPMEDIA4[,nn])|
          YES}}}
      [,VOLADDR={symb-addr|YES}]
      [,NORESP=symb-addr]
      [,FUNCERR=symb-addr]
      [,SELNERR=symb-addr]
      [,UNEXPIN=symb-addr]
```

This macro specifies that operations on a data set are to be terminated abnormally. The current outboard destination is deselected abnormally.

TRANSMISSION OF DATA FROM HOST TO OUTPUT DEVICES (TYPE=SEND)

```
DFHDI TYPE=(SEND[,{SAVE|NOSAVE}]
        [,{WAIT|NOWAIT}]}])
      {{,DNADDR={symb-addr|YES}}
      |{,SELECT={(CONSOLE|PRINT|CARD|
          WPMEDIA1|WPMEDIA2|WPMEDIA3|
        WPMEDIA4[,nn])|YES}}}
      [,VOLADDR={symb-addr|YES}]
      [,DEFRESP=YES]
      [,FUNCERR=symb-addr]
      [,SELNERR=symb-addr]
      [,UNEXPIN=symb-addr]
      [,NORESP=symb-addr]
```

Data for an output medium is transmitted to the logical unit from the TIOA, as indicated by the TCTTEDA. The SAVE parameter indicates that the TIOA is to be saved; however, there is no guarantee that TCTTEDA will remain unchanged.

The WAIT parameter indicates that task activity is to be suspended until the previous DFHDI macro has been executed.

TRANSMISSION OF DATA FROM DATA SET TO HOST (TYPE=RECEIVE)

```
DFHDI TYPE=(RECEIVE[,{SAVE|NOSAVE}])
      [,NORESP=symb-addr]
      [,EODS=symb-addr]
      [,DSSTAT=symb-addr]
      [,UNEXPIN=symb-addr]
```

This macro specifies that DFHTC TYPE=READ macros are to be generated to obtain records from the inbound data stream. These records are returned to the application program in a TIOA addressed by TCTTEDA. The number of records returned by the TYPE=READ macro depends upon whether the chain assembly or logical read options have been specified by the system programmer, which in turn depend upon the data format transmitted by the outboard controller.

When an FMH is encountered it is removed from the data stream and a response code is set to inform the application program of the change in destination selection status. Response codes are discussed below.

When the FMH is for BEGIN or RESUME DESTINATION, and no data is obtained from the READ, a further READ is issued so that the request can complete with user data.

When the FMH is for SUSPEND, END, or ABORT destination, the data, if present, is presented first to the application program with a normal response code; on the next request, the appropriate response code is set. The response code indicating the change of destination status is presented to the application program with no user data. If a name is sent, TCADIDNA is set, on completion of each request, to point to a field describing the host destination as a one byte length field followed by the destination name. If no destination name is sent, the field TCADISEL is set to the medium and sub-address sent. For descriptions of the formats and codes used, see the SELECT operand later in the chapter.

When reading from multiple data sets on an LUTYPE4, the DSSTAT condition will be raised by any attempted read after an end-of-data-set FMH has been received. The condition indicates that the logical unit has currently no more data to send.

The SAVE operand specifies that the contents of the TIOA are to be saved; however, there is guarantee that the TCTTEDA will remain unchanged.

<u>OBTAINING THE RELATIVE RECORD NUMBER OF NEXT RECORD (TYPE=NOTE)</u>

```
DFHDI TYPE=NOTE
      ,DNADDR={symb-addr|YES}
      [,VOLADDR={symb-addr|YES}]
      [,NORESP=symb-addr]
      [,FUNCERR=symb-addr]
      [,SELNERR=symb-addr]
      [,UNEXPIN=symb-addr]
```

This macro specifies that the relative record number of the position in the data set of the next available record is to be returned to the application program in a fullword field whose address is placed in the TCA at TCADIRNA after execution of the macro. The outboard destination is a user-defined addressed direct data set.

<u>SUSPENSION OF EXECUTION OF TASK (TYPE=WAIT)</u>

```
DFHDI  TYPE=WAIT
```

This macro specifies that task activity is to be suspended until the previous DFHDI macro has been executed. This macro is meaningful only following a DFHDI TYPE=ADD, TYPE=ERASE, TYPE=REPLACE, or TYPE=SEND.

<u>TESTING RESPONSE TO A REQUEST FOR DATA INTERCHANGE SERVICES (TYPE=CHECK)</u>

```
DFHDI TYPE=CHECK
      [,NORESP=symb-addr]
      [,EODS=symb-addr]
      [,DSSTAT=symb-addr]
      [,FUNCERR=symb-addr]
      [,SELNERR=symb-addr]
      [,UNEXPIN=symb-addr]
```

This macro specifies that the response code from the previous DFHDI macro is to be tested and, where necessary, a branch made to the user-written routine whose address is specified in one of the following operands: NORESP, EODS, DSSTAT, FUNCERR, SELNERR, or UNEXIN.

<u>BATCH DATA INTERCHANGE RESPONSE CODES</u>

Response codes are grouped into categories according to the operands. Each category is given a code, for example NORESP has category code X'00' which is placed in field TCADIRC1 in the TCA. Each category is subdivided into response codes that indicate the success or failure of a specified operation, for example "End destination FMH received" in category X'04' is 11. These response codes are placed in field TCADIRC2 in the TCA.

The categories, operands, response codes and their causes are shown in Figure 23 on page 202.

Category	Operand	Condition	Response Code
X'00'	NORESP	Successful	00
		Begin destination FMH received Resume destination FMH received	01 02
X'04'	DSSTAT,EODS	End destination FMH received Suspend destination FMH received Abort destination FMH received Currently no data to send	11 12 13 15
X'08'	FUNCERR	Request invalid for data set organization Record too long Data set full Invalid keyword or record identifier Resource not available Invalid NUMREC Insufficient resource. Request for change direction (RCD) signaled Transient Data error during logging	21 22 23 24 25 26 28 2B 60
X'0C'	SELNERR	Data set not found Destination does not exist Media not supported Invalid destination name Transient Data error during logging	29 41 43 44 60
X'10'	UNEXPIN	Unexpected sense Unexpected FMH Unsupported input	F1 F2 F3

Figure 23. Batch Data Interchange Response Codes

OPERANDS OF DFHDI MACRO

Response codes are described above.

DEFRESP=YES
All DFHTC TYPE=WRITE macros issued as a result of the current invocation of DFHDI will request a definite response from the outboard batch program, irrespective of the specification of message integrity for the CICS task.

DNADDR=
specifies the name of an outboard destination. If a destination with a different name is currently selected, it is deselected before this one is selected. If the current destination is being respecified then no selection is performed. This operand cannot be used with the SELECT operand.

symbolic-address
is the address of a field defining the destination name. This field consists of a one byte name length followed by the name itself.

YES
indicates that the application program has set this address into the word field TCADIDNA. The current implementation gives a maximum length of 8 characters for the destination name.

DSSTAT=symb-addr
specifies the entry label of the user-written routine to which control is passed when testing of the response code indicates a discontinuity in the inbound data stream.

EODS=symb-addr
specifies the entry label of the user-written routine to which control is passed when testing of the response code indicates the end of the data stream.

FUNCERR=symb-addr
specifies the entry label of the user-written routine to which control is passed when testing of the response code indicates a function error.

KEYADDR=
This identifies a record of a keyed direct data set.

symb-addr
specifies the address of a field defining the primary key of the record in a keyed direct data set to be erased. This field consists of a one

byte key length followed by the key itself.

YES
indicates that the application has set this address into fullword TCADIKYA. The current implementation gives a maximum length of 24 characters for the record key.

Note: This operand is not required when adding records to a 3790 keyed direct data set as the key value is embedded in the data specified by the DATA operand.

See the IBM 3790 Host System Programmer's Guide for a specification of valid keys.

NUMREC=
specifies the number of records affected by the current request. The 3790 will accept values greater than 1 only for the REPLACE operation on an addressed direct data set. The value is not meaningful to CICS but is conveyed to the outboard batch program as part of the function selection information. Records are replaced sequentially starting with the one identified by the RRNADDR operand.

integer
specifies the number of records, in the range 1 through 255, that are to be replaced.

YES
specifies that the application has set the binary value into the one byte field TCADINRS. If omitted this operand defaults to the value 1.

NORESP=symb-addr
specifies the entry label of the user-written routine to which control is passed when testing of the response code indicates normal response, that is, no errors have occurred during the processing specified by a DFHDI macro.

RRNADDR=
identifies a record of an addressed direct data set for the function REPLACE.

record-id
is the address of a one word field containing the relative record number of the record being replaced.

YES
indicates that the application has set this address into the fullword TCADIRNA.

Note: Record identifiers begin with the value 1.

SELECT=
specifies the type of output medium for the function SEND. This operand cannot be used with the DNADDR operand.

CONSOLE
specifies the medium provided for messages to the operator.

PRINT
specifies a printer.

CARD
specifies a card reader/punch.

WPMEDIA1 through WPMEDIA4
Specify, respectively, word processing media 1, 2, 3, and 4.

nn
specifies a medium subaddress in the range 00 to 15, where 15 means any available subaddress. The default is 00.

YES
specifies the medium code and subaddress have been placed in the one-byte field TCADISEL by the application program.

The first half of this field must contain a hexadecimal code indicating the type of medium, as shown below. The second half must contain the hexadecimal value of the subaddress (X'00' through X'15').

Code	Meaning
X'00'	CONSOLE
X'20'	CARD
X'30'	PRINT
X'80'	WPMEDIA1
X'90'	WPMEDIA2
X'A0'	WPMEDIA3
X'C0'	WPMEDIA4

SELNERR=symb-addr
specifies the entry label of the user-written routine to which control is passed when testing of the response code indicates errors during destination selection.

UNEXPIN=symb-addr
specifies the entry label of the user-written routine to which control is passed when testing of the response code indicates that unexpected or unrecognizable input or response is received in reply to a DFHDI macro. Response codes are described earlier in the chapter.

VOLADDR=
specifies, for the 3770 programmable subsystem only, the name of the diskette volume containing the data set named in the DNADDR operand. This name is used to qualify the data set name for destination selection. Subsequent specifications of the same data set name without a diskette volume name, or with a different diskette volume name, will cause a new destination to be selected. In the former case, all mounted diskette volumes will be searched for the data set named in the DNADDR operand.

symb-addr
specifies the address of the field defining the diskette volume name (to a maximum of six characters). The field consists of a one-byte name length followed by the name itself.

YES
indicates that the application program has set this address into the fullword field TCADIVNA.

PART 5. CONTROL OPERATIONS

CHAPTER 5.1. INTRODUCTION TO CONTROL OPERATIONS

This part of the manual describes the CICS macros that control the execution of tasks within a CICS system. The macros are associated with appropriate control programs and the specification of the various TYPE= operands invokes a range of operations.

The control programs and the macros associated with each are as follows:

- Interval Control Program (DFHIC Macro). This macro specifies operations that depend on the time of day and can have nine types of operations associated with it: GETIME, WAIT, POST, INITIATE, PUT, GET, CANCEL, RETRY, and CHECK. These operations are described in "Chapter 5.2. Interval Control (DFHIC Macro)" on page 209.

- Task Control Program (DFHKC Macro). This macro specifies operations that affect task activity or the control of resources. It can have eight types of operation associated with it: ATTACH, SCHEDULE, CHAP, WAIT, ENQ, DEQ, PURGE, and NOPURGE. These operations are described in "Chapter 5.3. Task Control (DFHKC Macro)" on page 221.

- Program Control Program (DFHPC Macro). This macro specifies operations that affect the flow of control between application programs. It can have ten types of operation associated with it: LINK, XCTL, LOAD, RETURN, DELETE, ABEND, SETXIT, RESETXIT, COBADDR, and CHECK. These operations are described in "Chapter 5.4. Program Control (DFHPC Macro)" on page 231.

- Storage Control Program (DFHSC Macro). This macro specifies operations that affect the acquisition and release of areas of main storage. It can have two types of operation: GETMAIN and FREEMAIN. These operations are described in "Chapter 5.5. Storage Control (DFHSC Macro)" on page 241.

- Transient Data Control Program (DFHTD Macro). This macro specifies operations that affect the queuing and retrieval of data in main storage or auxiliary storage. It can have five types of operation: PUT, GET, FEOV, PURGE, and CHECK. These operations are described in "Chapter 5.6. Transient Data Control (DFHTD Macro)" on page 245.

- Temporary Storage Control Program (DFHTS Macro). This macro specifies operations that affect the temporary storage of data in main storage or auxiliary storage. It can have seven types of operation: PUT, PUTQ, GET, GETQ, RELEASE, PURGE, and CHECK. These operations are described in "Chapter 5.7. Temporary Storage Control (DFHTS Macro)" on page 251.

CHAPTER 5.2. INTERVAL CONTROL (DFHIC MACRO)

CICS maintains the current time of day in two formats:

- a binary value, in CSACTODB, which is updated automatically during task dispatching to reflect the time of day maintained by the operating system
- a packed value, in CSATODP, which is updated when control returns from an operating system WAIT or when a DFHIC TYPE=GETIME,FORM=PACKED macro is executed.

The accuracy of these values at a given moment depends upon the task mix and the frequency of task switching operations.

Time management provides the capability of controlling various task functions based on the time of day or on intervals of time. The services available are listed below and are available to the application programmer through the interval control macro (DFHIC).

1. Provide the time of day in binary or packed decimal representation.
2. Provide task synchronization based on time-dependent events.
3. Provide automatic time-ordered task initiation with associated data retention and recovery support.

The application programmer must specify parameter values when using the DFHIC macro. The values can be specified in either of two ways:

1. By including the parameters in operands of the DFHIC macro by which time services are requested, or
2. By coding instructions that place the parameter values in fields of the TCA prior to issuing the DFHIC macro.

The second of these approaches provides flexibility in that the parameter values of a single DFHIC macro can be altered at execution time.

The application programmer can check the CICS response to a request for time services as explained under "Test Response to a Request for Time Services", later in the chapter. If the programmer does not check for a particular response, and the condition corresponding to that response occurs, program flow proceeds to the next sequential instruction in the application program. All operands that can be included in the DFHIC macro are discussed in detail at the end of the chapter.

Expiration Times

The time at which a time-controlled function is to be performed is called the **expiration time**. Expiration times can be specified in two ways, absolutely as the 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 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 adds 24 hours, that is, the requested function is performed at the time specified but on the next day.

Examples of the DFHIC TYPE=INITIATE macro specifying absolute time-of-day requests, are as follows:

- DFHIC TYPE=INITIATE,TIME=123000 issued at 1000 hours on Monday will expire at 1230 hours on the same Monday.
- DFHIC TYPE=INITIATE,TIME=090000 issued at 1000 hours on Monday will expire immediately because the specified time is within the preceding six hours
- DFHIC TYPE=INITIATE,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.
- DFHIC TYPE=INITIATE,TIME=330000 issued at 1000 hours on Monday will expire at 0900 hours on Tuesday.

TIME-OF-DAY UPDATING (TYPE=GETIME)

```
DFHIC TYPE=GETIME
      [,FORM={BINARY|PACKED }]
      [,TIMADR={symb-addr|YES}]
      [,NORESP=symb-addr]
      [,INVREQ=symb-addr]
      [,ERROR=symb-addr]
```

In the course of normal operation, CICS maintains the current time of day in binary form at CSACTODB and in packed decimal form at CSATODP. The binary representation is expressed as a four-byte positive value in hundredths of a second. The packed decimal representation is expressed as a four-byte positive signed value of the form HHMMSSt+ where the seconds are truncated to tenths of a second.

| The values are updated periodically during task dispatching. The accuracy of these values at any given moment depends on the task mix and the frequency of task switching operations.

The application programmer can ensure that both these time-of-day values are updated to a current setting by issuing the DFHIC TYPE=GETIME macro. This macro causes both forms of the time of day to be updated in the CSA and, optionally, places the requested form of the time of day in a four-byte field specified by the application programmer. When the programmer wants the time of day to be returned in a field other than those of the CSA, either the symbolic label of the four-byte field must be specified in the DFHIC TYPE=GETIME macro or the address of the field must be placed in TCAICDA prior to issuing the DFHIC TYPE=GETIME macro.

Note: For performance reasons, it should be recognized that lengthy conversion routines must be executed whenever updating of the packed decimal representation of time of day is requested.

The following example shows how to request that the time of day be placed at the storage locations represented by the symbolic label CLOCK.

```
DFHIC TYPE=GETIME,FORM=PACKED,
      TIMADR=CLOCK
```

The following examples show how to request that the time of day be placed in a field selected prior to (and independent of) execution of the DFHIC TYPE=GETIME macro.

ASM:

```
      MVC TCAICDA,=A(CLOCK)
      .
      .
      .
      DFHIC TYPE=GETIME,FORM=PACKED,
            TIMADR=YES
```

COBOL:

```
      MOVE CLOCKADR TO TCAICDA.
      .
      .
      .
      DFHIC TYPE=GETIME,FORM=PACKED,
            TIMADR=YES
```

PL/I:

```
      TCAICDA=ADDR(CLOCK);
      .
      .
      .
      DFHIC TYPE=GETIME,FORM=PACKED,
            TIMADR=YES
```

DELAY PROCESSING OF A TASK (TYPE=WAIT)

The format of the DFHIC macro to delay processing of a task until a specified time occurs is as follows:

```
DFHIC TYPE=WAIT
      [,INTRVAL={numeric value|YES}]|
         [,TIME={numeric value|YES}]
      [,REQID={name|YES|'prefix'}]
      [,NORESP=symb-addr]
      [,INVREQ=symb-addr]
      [,EXPIRD=symb-addr]
      [,ERROR=symb-addr]
```

The task synchronization feature of CICS time management provides the capability either of delaying the processing of a requesting task until a specified time occurs or of signaling the requesting task when a specified interval of time has elapsed. It also supports the cancellation of a pending time-ordered synchronization event by another task. See "Cancel a Request for Time Services (TYPE=CANCEL)" on page 215.

This macro causes the requesting task to temporarily suspend processing, and to resume control at a specified time of day or after a specified interval of time has elapsed. The INTRVAL and TIME operands are mutually exclusive. This macro supersedes and cancels any previously initiated DFHIC TYPE=POST macro for the task.

A numeric value specified in, or before issuing, the DFHIC TYPE=WAIT macro is used by CICS to calculate the time at which the requested time service is to be provided. See the section "Expiration Times" earlier in the chapter.

To identify the request and any data associated with it, a unique identification is assigned to each time-ordered request. The application programmer can specify a request identification to be assigned to his DFHIC TYPE=WAIT macro by the REQID operand. If none is assigned by the programmer, CICS assigns a unique request identification. A request identification should be specified by the application programmer if he wishes to provide another task with the capability of canceling the unexpired WAIT request. See "Cancel a Request for Time Services (TYPE=CANCEL)" on page 215.

The following example shows how to temporarily suspend the processing of a task for a specified period of time:

```
  DFHIC TYPE=WAIT,INTRVAL=500,
        REQID=GXLBZQMR
```

The following examples show how to request the suspension of a task until the time of day stored previously in TCAICRT is reached. A request identifier previously selected by the user is stored in TCAICQID as a unique identifier for this request for time service.

ASM:

```
      MVC TCAICRT,=PL4'124500'
      MVC TCAICQID,UNIQCODE
       .
       .
       .
      DFHIC TYPE=WAIT,TIME=YES,REQID=YES
```

COBOL:

```
      MOVE 124500 TO TCAICRT.
      MOVE UNIQCODE TO TCAICQID.
       .
       .
       .
      DFHIC TYPE=WAIT,TIME=YES,REQID=YES
```

PL/I:

```
      TCAICRT=124500;
      TCAICQID=UNIQCODE;
       .
       .
      DFHIC TYPE=WAIT,TIME=YES,REQID=YES
```

SIGNAL EXPIRATION OF A SPECIFIED TIME (TYPE=POST)

```
  DFHIC TYPE=POST
        [,INTRVAL={numeric value|YES}]|
            [,TIME={numeric value|YES}]
        [,REQID={name|YES|'prefix'}]
        [,NORESP=symb-addr]
        [,INVREQ=symb-addr]
        [,EXPIRD=symb-addr]
        [,ERROR=symb-addr]
```

In response to this macro, CICS makes a timer event control area available to the user for testing. This four-byte storage area is initialized to binary zeros and its address is returned to the requesting task in TCAICTEC.

When CICS determines that the time specified in a DFHIC TYPE=POST macro has expired, byte 0 of the timer event control area is set to X'40' and byte 2 is set to X'80'. This form of posting is compatible with the completion code postings performed by the operating systems. The timer event control area can be used as the event control area referred to in a DFHKC TYPE=WAIT macro. (See "Synchronize a Task (TYPE=WAIT)" on page 224.)

The timer event control area provided to the user is not released or altered (except as described above) until one of the following events occurs:

- The task issues a subsequent DFHIC TYPE=WAIT, DFHIC TYPE=POST, DFHIC TYPE=INITIATE, or DFHIC TYPE=PUT macro.
- The task issues a DFHIC TYPE=CANCEL macro request to nullify the DFHIC TYPE=POST macro (this releases the storage area occupied by the timer event control area).
- The task terminates, normally or abnormally.

A task can have only one DFHIC TYPE=POST request active at any given time. Any DFHIC TYPE=WAIT, DFHIC TYPE=POST, DFHIC TYPE=INITIATE, or DFHIC TYPE=PUT request supersedes and cancels a previously issued DFHIC TYPE=POST request by the task.

Note: The expiration of any CICS time-ordered event is determined by CICS when it is performing its task dispatching function. Therefore, for "posting" to occur, the application programmer must ensure that the task relinquishes control of CICS before each testing of the timer event control area. This can be done directly by issuing the DFHKC TYPE=WAIT or DFHKC TYPE=CHAP macro (see "Synchronize a Task (TYPE=WAIT)" on page 224) or indirectly by requesting a CICS service that in turn initiates a task service on behalf of the task.

A numeric value specified in, or before issuing, the DFHIC TYPE=POST macro is used by CICS to calculate the time at which the requested time service is to be provided. See the section "Expiration Times" earlier in the chapter.

The application programmer can specify a request identification to be assigned to a posting request by the REQID operand. If none is assigned by the programmer, CICS assigns a unique request identification, which is returned to the application program in TCAICQID. In either case, the request identification provides a means of symbolically identifying the request.

This macro indicates that CICS is to make a 4-byte timer event control area available to the application program for testing. The area is initialized to binary zeros, and its address is returned in TCAICTEC to the application program. This area is available to the application program for the duration of the task and is overridden if the application program issues another DFHIC request of the following types: POST, WAIT, PUT, or INITIATE.

The following example shows how to request that CICS provide a signal for the task when a specified interval of time has elapsed:

```
  DFHIC TYPE=POST,INTRVAL=30
```

The following examples show how to dynamically request that CICS provide a signal for the task when the time of day previously stored in TCAICRT is reached. Since no request identification is specified by the application programmer, CICS automatically assigns one and returns it to the application program at TCAICQID.

ASM:

```
       MVC TCAICRT,PACKTIME
       .
       .
       DFHIC TYPE=POST,TIME=YES
       MVC UNIQCODE,TCAICQID
```

COBOL:

```
       MOVE PACKTIME TO TCAICRT.
       .
       .
       DFHIC TYPE=POST,TIME=YES
       MOVE TCAICQID TO UNIQCODE.
```

PL/I:

```
       TCAICRT=PACKTIME;
       .
       .
       DFHIC TYPE=POST,TIME=YES
       UNIQCODE=TCAICQID;
```

INITIATE A TASK WITHOUT DATA (TYPE=INITIATE)

```
DFHIC TYPE=INITIATE
      [,INTRVAL={numeric value|YES}]|
        [,TIME={numeric value|YES}]
      [,REQID={name|YES|'prefix'}]
      [,TRANSID=name]
      [,TRMIDNT={name|YES}]
      [,NORESP=symb-addr]
      [,INVREQ=symb-addr]
      [,TRNIDER=symb-addr]
      [,TRMIDER=symb-addr]
      [,ERROR=symb-addr]
```

Through this macro, the application programmer provides the transaction identification of the task to be initiated at some future time and other information about the task. CICS queues the request until the specified time occurs. When the necessary resources are available (for example, a terminal), the task is initiated. Only one task is initiated if multiple DFHIC TYPE=INITIATE requests for the same transaction and terminal expire at the same time or prior to terminal availability. No data can be passed to the future task by means of the DFHIC TYPE=INITIATE macro. (To do so, see "Task Initiation with Data (PUT)," which follows.) This request supersedes and cancels any previously initiated DFHIC TYPE=POST request by the initiating task.

A numeric value specified in or before issuing the DFHIC TYPE=INITIATE macro is used by CICS to calculate the time of day at which the requested time service is to be provided. See the section "Expiration Times" earlier in the chapter.

As stated earlier, a unique request identifier is assigned to each time-ordered request as a means of symbolically identifying the request and any data associated with it. The application programmer can specify an identifier for his initiation request, or he can let CICS assign one, in which case it is returned to the application program in TCAICQID.

The application programmer must specify the transaction identification of the future task, either in the DFHIC TYPE=INITIATE macro or by placing it in TCAICTI before issuing the macro. CICS validates the transaction identification by scanning the program control table (PCT). If the specified identifier is not found in the table, CICS does not provide the requested service; a response code is placed at TCAICTR (for assembler language or PL/I) or at

TCAICRC (for COBOL) to indicate that the transaction identification is not valid.

If the future task must communicate with a terminal, the application programmer must also specify a terminal identifier, either in the macro or by placing it beforehand in TCAICTID. CICS validates the terminal identifier by scanning the terminal control table (TCT); if it fails to locate the terminal identifier in the TCT, CICS provides a response code at TCAICTR (for assembler language or PL/I) or at TCAICRC (for COBOL) without servicing the request.

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

```
  DFHIC TYPE=INITIATE,INTRVAL=10000,
        TRANSID=TRNL
```

The following examples show how to dynamically request automatic initiation of a task associated with a terminal. The task initiation time, transaction identification, and terminal identification are moved to fields of the TCA before the DFHIC TYPE=INITIATE macro is issued. Since no request identification is specified by the application programmer, CICS automatically assigns one and returns it to the application program at TCAICQID.

ASM:

```
        MVC TCAICRT,=PL4'10000',
        MVC TCAICTI,=CL4'TRN1'
        MVC TCAICTID,=CL4'STA5'
        .
        .
        .
        DFHIC TYPE=INITIATE,
              INTRVAL=YES,
              TRMIDNT=YES
        MVC UNIQCODE,TCAICQID
```

COBOL:

```
        MOVE 10000 TO TCAICRT.
        MOVE 'TRN1' TO TCAICTI.
        MOVE 'STA5' TO TCAICTID.
        .
        .
        .
        DFHIC TYPE=INITIATE,
              INTRVAL=YES,
              TRMIDNT=YES
        MOVE TCAICQID TO UNIQCODE.
```

PL/I:

```
        TCAICRT=10000;
        TCAICTI='TRN1';
        TCAICTID='STA5';
        .
        .
        .
        DFHIC TYPE=INITIATE,
              INTRVAL=YES,
              TRMIDNT=YES
        UNIQCODE=TCAICQID;
```

TASK INITIATION WITH DATA (TYPE=PUT)

```
DFHIC TYPE=PUT
      [,INTRVAL={numeric value|YES}]|
       [,TIME={numeric value|YES}]
      [,REQID={name|YES|'prefix'}]
      [,TRANSID=name]
      [,TRMIDNT={name|YES}]
      [,ICDADDR={symb-addr|YES}]
      [,NORESP=symb-addr]
      [,INVREQ=symb-addr]
      [,TRNIDER=symb-addr]
      [,TRMIDER=symb-addr]
      [,IOERROR=symb-addr]
      [,ERROR=symb-addr]
```

This macro indicates that CICS is to initiate a nonterminal-oriented task at some future time and makes one data record available to that task, or provides time-ordered data to be made available to a terminal-oriented task that is to be initiated at some future time.

This macro is used to provide the transaction identification, the location of the data to be stored, and other information applicable to the task to be initiated. CICS stores the data and queues the request until the specified time occurs. As soon as all necessary resources are available (for example, a terminal), the task is initiated. CICS temporary storage management facilities support this facility of time management.

The DFHIC TYPE=PUT macro is used only when data is to be passed to a task to be initiated at some future time. It supersedes and cancels any previously initiated DFHIC TYPE=POST request of the task. If only task initiation at a future time is needed, the DFHIC TYPE=INITIATE macro should be used.

If the task to be initiated is associated with a terminal, the initial DFHIC TYPE=PUT request causes the task to be initiated at the specified time. Subsequent PUT macros with the same terminal identification, transaction identification, and expiration time are used to store data for subsequent retrieval by the initiated task. If the task to be initiated is not associated with a terminal, each DFHIC TYPE=PUT request results in a task being initiated at the specified time. That is, only one physical data record can be passed to a task not associated with a terminal. (See the section "Retrieve Time-Ordered Data (GET)", which follows.)

Most operands of the DFHIC TYPE=PUT macro are analogous to similar operands of the DFHIC TYPE=INITIATE macro. The discussions of time calculation, request identification, transaction identification, and terminal identification given in the section "Task Initiation without Data (INITIATE)," which precedes this section, apply to DFHIC TYPE=PUT in the same manner as they apply to DFHIC TYPE=INITIATE. In addition, because the DFHIC TYPE=PUT macro permits data to be passed, the application programmer must specify the symbolic address of the field containing the data. The label may be provided as a parameter of the macro or move the address to TCAICDA prior to issuing the macro.

The data passed to an initiated task must have the standard variable-length format, with the first four bytes containing LLbb. LL is a two-byte binary length field (the value of which includes the length of the data plus the first four bytes), and bb is a two-byte field containing binary zeros.

Note: An IOERROR will occur if there is not enough auxiliary temporary storage available to hold the data being passed. See the appropriate <u>CICS Customization Guide</u> discussion of temporary storage for further details of auxiliary temporary storage requirements.

The following example shows how to request automatic task initiation and request that time-ordered data be made available to a task associated with a terminal:

```
  DFHIC TYPE=PUT,TIME=173000,
        TRANSID=TRN2,TRMIDNT=STA3,
        ICDADDR=DATAFLD
```

The following examples show how to dynamically request automatic task initiation and request that time-ordered data be made available to a task associated with a terminal. Values for time, request identification, transaction identification, and terminal identification, as well as the address of data to be passed, are moved to appropriate fields of the TCA before issuing the DFHIC TYPE=PUT macro.

ASM:

```
       MVC TCAICRT,PACKTIME
       MVC TCAICQID,UNIQCODE
       MVC TCAICTI,=CL4'TRN2'
       MVC TCAICTID,=CL4'STA3'
       MVC TCAICDA,=A(DATAFLD)
       .
       .
       .
       DFHIC TYPE=PUT,
             TIME=YES,
             TRMIDNT=YES,
             REQID=YES,
             ICDADDR=YES
```

COBOL:

```
       MOVE PACKTIME TO TCAICRT.
       MOVE UNIQCODE TO TCAICQID.
       MOVE 'TRN2' TO TCAICTI.
       MOVE 'STA3' TO TCAICTID.
       MOVE DATADDR TO TCAICDA.
       .
       .
       DFHIC TYPE=PUT,
             TIME=YES,
             TRMIDNT=YES,
             REQID=YES,
             ICDADDR=YES
```

PL/I:

```
       TCAICRT=PACKTIME;
       TCAICQID=UNIQCODE;
       TCAICTI='TRN2';
       TCAICTID='STA3';
       TCAICDA=ADDR(DATAFLD);
       .
       .
       .
       DFHIC TYPE=PUT,
             TIME=YES,
             TRMIDNT=YES,
             REQID=YES,
             ICDADDR=YES
```

RETRIEVE TIME-ORDERED DATA (TYPE=GET)

```
DFHIC TYPE=GET
      [,ICDADDR={symb-addr|YES}]
      [,RELEASE=NO]
      [,NORESP=symb-addr]
      [,INVREQ=symb-addr]
      [,NOTFND=symb-addr]
      [,ENDDATA=symb-addr]
      [,IOERROR=symb-addr]
      [,TSINVLD=symb-addr]
      [,ERROR=symb-addr]
```

Only data from an expired DFHIC TYPE=PUT request can be accessed using the DFHIC TYPE=GET macro. To retrieve data stored by use of a DFHIC TYPE=PUT request, the DFHIC TYPE=GET macro must be used.

When time-ordered data is to be retrieved by means of a DFHIC TYPE=GET macro, the application programmer may specify the address of a storage area into which the data is to be placed. The address is specified either by including the address in the macro or by storing it in TCAICDA prior to issuing the macro. In either case, the storage area must be large enough to contain the four-byte length field (LL//) at the beginning of the data record as well as the data portion of the record. If the application programmer does not select a storage area, CICS automatically acquires an area of sufficient size and returns the address of that area in TCAICDA.

Each originating DFHIC TYPE=PUT macro provides the transaction identification of the task to receive the data, and if applicable, symbolically identifies the terminal associated with the task's operation. When CICS services a DFHIC TYPE=PUT macro, it does so in two steps; it first queues the request for automatic task initiation at a specified time and then stores the data. When the specified time occurs, the task is ready to be initiated, and the stored data is then available for retrieval.

A task not associated with a terminal that is initiated as a result of an expired DFHIC TYPE=PUT request can access only the single physical data record associated with the original request. It does this by issuing one DFHIC TYPE=GET macro. The storage occupied by the data associated with the task is released upon execution of the DFHIC TYPE=GET request, or upon termination of the task (normally or abnormally) if no DFHIC TYPE=GET macro is executed prior to termination.

A task associated with a terminal that is initiated as the result of an expired DFHIC TYPE=PUT macro, or that is active at the time of expiration of a DFHIC TYPE=PUT macro, can access all data records associated with expired DFHIC TYPE=PUT macros having the same transaction identification and terminal identification. Therefore, a task associated with a terminal can retrieve all data made available to the terminal and the task up to the current time by issuing consecutive DFHIC TYPE=GET macros.

Expired data records are presented to the task upon request in expiration time sequence. (Note that the data record is obtained from temporary storage using the REQID of the original DFHIC TYPE=PUT request as the temporary storage DATAID.) The storage occupied by the single data record associated with a DFHIC TYPE=PUT request is released after the data has been retrieved by a DFHIC TYPE=GET request or upon termination of CICS. Data passed in subsequent expired DFHIC TYPE=PUT requests specifying the same terminal identification and transaction identification can be retrieved in response to DFHIC TYPE=GET requests by the same task if that task is still active at their expiration times. Otherwise, such a DFHIC TYPE=PUT request causes a new task to be initiated.

When all passed data for which specified times have expired has been retrieved, CICS provides an end-of-data response at TCAICTR (for assembler language or PL/I) or TCAICRC (for COBOL) in response to a DFHIC TYPE=GET macro.

The following example shows how to request retrieval of a time-ordered data record into a data area specified in the request:

```
  DFHIC TYPE=GET,ICDADDR=DATAFLD
```

The following examples show how to dynamically request retrieval of a time-ordered data record. The address of the storage area reserved for the data record is placed in TCAICDA prior to the issuance of the DFHIC TYPE=GET macro.

ASM:

```
       MVC TCAICDA,=A(DATAFLD)
       .
       .
       .
       DFHIC TYPE=GET,ICDADDR=YES
```

COBOL:

```
       MOVE DATADDR TO TCAICDA.
       .
       .
       .
       DFHIC TYPE=GET,ICDADDR=YES
```

PL/I:

```
       TCAICDA=ADDR(DATAFLD);
       .
       .
       .
       DFHIC TYPE=GET,ICDADDR=YES
```

CANCEL A REQUEST FOR TIME SERVICES (TYPE=CANCEL)

```
DFHIC TYPE=CANCEL
      [,REQID={name|YES}]
      [,NORESP=symb-addr]
      [,INVREQ=symb-addr]
      [,NOTFND=symb-addr]
      [,ERROR=symb-addr]
```

This macro specifies that a request of one of the following types is to be acted upon as follows:

1. DFHIC TYPE=WAIT issued by another task (now suspended) is to be treated as though expired.

2. DFHIC TYPE=POST issued by this task is to be removed from the system.

3. DFHIC TYPE=POST issued by another task is to be treated as though expired.

4. DFHIC TYPE=INITIATE is to be removed from the system.

5. DFHIC TYPE=PUT is to be removed from the system.

The effect of the cancellation is dependent on whether a request identification is specified for the DFHIC TYPE=CANCEL request and on the type of service request being canceled.

Cancel an Interval Control POST Request

A DFHIC TYPE=POST request can be canceled by the originating task or by another task through use of the DFHIC TYPE=CANCEL macro.

When the originating task cancels a DFHIC TYPE=POST request, no request identification should be specified for the cancellation request. This cancellation request can be made either before or after expiration of the original request. In either case, the storage reserved for the timer event control area is released, and all references to the original request are removed from the system.

When a task other than the originating task cancels a DFHIC TYPE=POST request, the request identification of that request must be specified. The effect of the cancellation is the same as an early expiration of the original DFHIC TYPE=POST request. That is, the timer event control area for the originating task is posted as though the original expiration time had been reached.

Cancel an Interval Control WAIT Request

A DFHIC TYPE=WAIT request can only be canceled prior to its expiration, and only by a task other than the task that issued the DFHIC TYPE=WAIT (the originating task is suspended for the duration of the request). The request identification of the suspended task must be specified.

The effect of the cancellation is the same as an early expiration of the original DFHIC TYPE=WAIT or DFHKC TYPE=CHAP request. That is, the originating task resumes control (based on its normal dispatching priority) as though the original expiration time had been reached.

Cancel an Interval Control INITIATE or PUT Request

A request identification must be specified when the DFHIC TYPE=CANCEL macro is used to cancel a DFHIC TYPE=INITIATE or DFHIC TYPE=PUT request.

The effect of the cancellation is to remove the original request from the system, treating the original request as though it had never been made. The cancellation request is effective only prior to expiration of the original request.

I/O ERROR RETRY (TYPE=RETRY)

```
DFHIC TYPE=RETRY
      [,RELEASE=NO]
      [,NORESP=symb-addr]
      [,INVREQ=symb-addr]
      [,NOTFND=symb-addr]
      [,IOERROR=symb-addr]
      [,ERROR=symb-addr]
```

CICS attempts to retrieve the data record whose symbolic eight-character identification is specified at TCAICQID, and place it into the data area specified at TCAICDA. These fields are preset by CICS at the time the I/O error response is returned to the application program.

TEST RESPONSE TO A REQUEST FOR TIME SERVICES (TYPE=CHECK)

```
DFHIC TYPE=CHECK
      [,NORESP=symb-addr]
      [,INVREQ=symb-addr]
      [,EXPIRD=symb-addr]
      [,TRNIDER=symb-addr]
      [,TRMIDER=symb-addr]
      [,NOTFND=symb-addr]
      [,ENDDATA=symb-addr]
      [,IOERROR=symb-addr]
      [,TSINVLD=symb-addr]
      [,ERROR=symb-addr]
```

INTERVAL CONTROL RESPONSE CODES

The assembler language or PL/I programmer can access interval control response codes at TCAICTR; the COBOL programmer can access interval control response codes at TCAICRC. The possible response codes and the conditions to which they correspond are identified in the right-hand columns of Figure 24 on page 217. DFHIC macros for which the conditions are applicable are shown at the left.

If the application programmer does not check for a particular response to his service request, and the exception condition corresponding to that response occurs, program flow proceeds to the next sequential instruction in the application program.

The following examples show how to examine the response code provided by CICS at TCAICTR (for assembler language or PL/I) or TCAICRC (for COBOL) and transfer control to the appropriate user-written exception-handling routine.

The alternative approach available to COBOL programmers is also shown.

ASM:

```
        DFHIC   TYPE=GET,ICDADDR=DATAFLD
        CLI     TCAICTR,X'00'
        BE      GOOD
        DFHPC   TYPE=ABEND,ABCODE=TIME
GOOD    DS       0H
        .
        .
```

COBOL:

```
        DFHIC TYPE=GET,ICDADDR=DATAFLD
        IF TCAICRC = LOW-VALUES
        THEN GO TO GOOD
        ELSE NEXT SENTENCE.
        DFHPC TYPE=ABEND
GOOD.
        .
        .
```

Alternatively, the COBOL programmer may make use of the CICS generated condition names to test responses. For example:

```
        IF ICNORESP THEN GO TO GOOD.
        .
        .
        .
```

PL/I:

```
        DFHIC TYPE=GET,ICDADDR=DATAFLD
        IF TCAICTR='0'B THEN GO TO GOOD;
        DFHPC TYPE=ABEND
GOOD:
        .
        .
```

Time Services Request by DFHIC Macro	Condition	Response Code		
		Assembler	COBOL	PL/I
ALL	NORESP (Normal response)	X'00'	LOW-VALUES (ICNORESP)	00000000
GET,CHECK	ENDDATA (End of data condition)	X'01'	12-1-9 (ICENDDATA)	00000001
PUT,GET,RETRY, CHECK	IOERROR (INPUT/Output error)	X'04'	12-4-9 (ICIOERROR)	00000100
INITIATE,PUT, CHECK	TRNIDER (Transaction identification error)	X'11'	11-1-9 (ICTRNIDER)	00010001
INITIATE,PUT, CHECK	TRMIDER (Terminal identification error)	X'12'	11-2-9 (ICTRMIDER)	00010010
GET,CHECK	TSINVLD (No temporary storage support)	X'14'	11-4-9 (ICTSINVLD)	00010100
WAIT,POST,CHECK	EXPIRD (Expired)	X'20'	11-0-1-8-9 (ICEXPIRD)	00100000
GET,CANCEL, RETRY,CHECK	NOTFND (Not found)	X'81'	12-0-1 (ICNOTFND)	10000001
ALL	INVREQ (Invalid request)	X'FF'	12-11-0-7-8-9 (ICINVREQ)	11111111
ALL	ERROR (Any response other than NORESP)	(Note 2)	(Note 2)	(Note 2)

Notes:

1. The names enclosed in parentheses in the COBOL column indicate the condition names generated by CICS. These names may be used in testing for the conditions in a COBOL program.

2. The test for the ERROR response is satisfied by a **not equal** condition; that is, not X'00', not LOW-VALUES, or not 00000000 for assembler language, COBOL, and PL/I, respectively.

Figure 24. Interval Control Response Codes

OPERANDS OF DFHIC MACRO

ENDDATA=symb-addr
specifies the entry label of the user-written routine to which control is to be passed if no more data is stored for the task issuing a DFHIC TYPE=GET request. It can be considered a normal end-of-file response when retrieving sequential time-ordered data records.

ERROR=symb-addr
specifies the entry label of the user-written routine to which control is to be passed if any of the response conditions other than NORESP occurs.

EXPIRD=symb-addr
specifies the entry label of the user-written routine to which control is to be passed if the time specified in a DFHIC TYPE=POST or DFHIC TYPE=WAIT request has expired at the time the request is issued.

FORM=
indicates which time-of-day representation is desired.

BINARY
specifies that a binary representation of time of day (a four-byte positive value in hundredths of a second) is to be updated and retained in CSACTODB.

PACKED
specifies that the binary representation of time of day (described above) and the packed decimal representation (a four-byte positive value of the form HHMMSSt+ where seconds are truncated to tenths of a second) are to be updated and retained in CSACTODB and CSATODP respectively.

Note: COBOL and PL/I programmers should be aware that the zone portion of the low-order byte of this positive number contains hexadecimal F rather than C or D.

ICDADDR=
specifies the location of the data to be stored for the task to be initiated at some future time.

symb-addr
is the symbolic address of the storage area containing the data to be made available to the task.

YES
indicates that the symbolic address of the storage area containing the data has been placed in TCAICDA.

If no data is to be passed, DFHIC TYPE=INITIATE rather than DFHIC TYPE=PUT should be used.

INTRVAL=
specifies the interval of time that is to elapse before CICS initiates a task, or before CICS posting is to occur, or for which a task is to be suspended.

numeric value
is of 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. This numeric value is added to the current clock time by CICS when the associated macro is executed to calculate the time of day (clock time) when the task is to be initiated or posted, or when processing of the task is to be resumed. When used with TYPE=INITIATE, if the specified interval is zero, or if both INTRVAL and TIME are omitted, the task is initiated immediately.

YES
indicates that the interval of time (in packed decimal form, HHMMSS+) has been placed in TCAICRT.

If this operand is specified, the TIME operand cannot be specified.

INVREQ=symb-addr
specifies the entry label of the user-written routine to which control is to be passed if an invalid type of request was received for processing by the interval control program.

IOERROR=symb-addr
specifies the entry label of the user-written routine to which control is to be passed if an input/output error occurs during a DFHIC TYPE=GET or DFHIC TYPE=PUT operation on auxiliary storage. The DFHIC TYPE=RETRY macro can be used in the routine for handling DFHIC TYPE=GET input/output errors.

One of the causes of this error is during a TYPE=PUT if there is insufficient auxiliary temporary storage available to hold any data which is to be passed. See the appropriate <u>CICS Installation and Operations Guide</u>. discussion of temporary storage for further details of auxiliary temporary storage requirements.

NORESP=symb-addr
specifies the entry label of the user-written routine to which control is to be passed if no error occurs. NORESP signifies "normal response."

NOTFND=symb-addr
specifies the entry label of the user-written routine to which control is to be passed if the request identification specified in a DFHIC TYPE=CANCEL macro fails to match an unexpired time-ordered request. It is also applicable to DFHIC TYPE=GET or DFHIC TYPE=RETRY requests and signifies that the time-ordered data stored for retrieval through the DFHIC TYPE=PUT macro cannot be located using the unique request identification contained in TCAICQID at the time of this request. This condition occurs on a retrieval operation if some prior task retrieved the data stored under the request identification directly through temporary storage facilities and then released the data area. It also occurs if the request identification associated with the original DFHIC TYPE=PUT request fails to remain a unique identification.

RELEASE=NO
indicates that CICS is not to release the record from temporary storage after obtaining the record for the application program.

Upon completion of a successful DFHIC TYPE=GET,RELEASE=NO request, CICS places the identification of the temporary-storage record in TCAICQID. Using this identification, the user can retrieve or release the record from temporary storage through the DFHTS macro; the record is not available to any subsequent DFHIC get requests.

This operand is valid only for a retry of a DFHIC TYPE=GET request.

REQID=
is an optional operand used to assign a unique request identification to this request, as a means of symbolically identifying the request. It should be used if the application programmer wishes to provide another task with the capability of canceling an unexpired WAIT request (see the discussion of DFHIC TYPE=CANCEL, earlier in the chapter). The data is put in temporary storage with this identification.

name
is a unique identifier, up to eight characters in length, selected for this request by the application programmer.

YES
indicates that an eight-character request identification has been placed in TCAICQID by the application program.

'prefix'
is a two-character (including blanks) prefix to be affixed to the request identification generated by CICS. If REQID='' is specified, the prefix is assumed to be in the two-byte field TCAICQPX.

If this operand is omitted, CICS generates a unique request identification in the form "DFNNNNNN"; the prefix is DF.

TIMADR=
is used when the time of day is to be returned in an application programmer-selected four-byte field. For FORM=BINARY, the binary representation is returned; for FORM=PACKED, the packed decimal representation is returned.

symb-addr
is the symbolic address of the field in which the time of day is to be made available to the application program.

YES
indicates that the symbolic address of the field for the time of day is in TCAICDA.

If this operand is omitted, the fields of the CSA are updated, but the time of day is not placed in another field for reference by the application program.

TIME=
specifies the time of day at which CICS is to initiate the requested service. See the section "Expiration Times" earlier in the chapter.

numeric value
is of 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.

YES
indicates that the time of day (in packed decimal form, HHMMSS+) has been placed in TCAICRT.

If this operand is specified, the INTRVAL operand cannot be specified.

TRANSID=name
is the symbolic transaction identification of the task to be initiated. If this operand is omitted, the transaction identification is assumed to be in TCAICTI.

TRMIDER=symb-addr
specifies the entry label of the user-written routine to which control is to be passed if the symbolic terminal identification specified in the DFHIC TYPE=INITIATE or DFHIC TYPE=PUT request cannot be found in the terminal control table (TCT).

TRMIDNT=
is the symbolic terminal identification of the terminal associated with the task to be initiated. This operand is required when the task to be initiated must communicate with a terminal; it should be omitted otherwise.

TRNIDER=symb-addr
specifies the entry label of the user-written routine to which control is to be passed if the symbolic transaction identification specified in a DFHIC TYPE=INITIATE or DFHIC TYPE=PUT request cannot be found in the program control table.

TSINVLD=symb-addr
specifies the entry label of the user-written routine to which control is to be passed if the CICS temporary storage program does not support a DFHTS TYPE=GET request issued by the CICS interval control program. This situation can occur when a dummy temporary storage program is included in the current CICS system in place of a functional temporary storage program.

CHAPTER 5.3. TASK CONTROL (DFHKC MACRO)

Task management provides the capability to process transactions (tasks) concurrently. Transactions are scheduled, through task control, and processed according to priorities assigned by the user. Control of the processor is given to the highest priority task that is ready to be processed. Control of the processor is returned to the operating system when no further work can be done by CICS or by user-written application programs.

When a transaction is initiated in CICS, task control dynamically allocates storage for the task control area (the TCA), places the task in the dispatching priority queue, obtains the identification of the program initially required to process the task from the program control table (the PCT), and transfers control to program control.

The task management macro (DFHKC) is used to:

- Initiate a task
- Change the priority of a task
- Synchronize a task
- Synchronize the use of a resource by a task
- Purge a task on system overload.

The application programmer must specify parameter values when using the DFHKC macro. The values can be specified in either of two ways:

1. By including the parameters in operands of the DFHKC macro by which task control services are requested, or
2. By coding instructions that place the parameter values in fields of the TCA prior to issuing the DFHKC macro.

The second method adds flexibility by letting the programmer vary the parameter values of a single DFHKC macro to meet the needs of a given program.

INITIATE A TASK (TYPE=ATTACH)

```
DFHKC TYPE=ATTACH
      [,FCADDR=symb-addr]
      [,TRANSID=name]
```

This macro causes task control to obtain the TCA for a task and insert the task in the dispatching priority queue according to the overall transaction processing priority of the task. This macro is intended to be used by other CICS control modules. However, it can also be used by the application programmer to initiate additional tasks which must terminate themselves by a DFHPC TYPE=RETURN macro.

Most tasks running under CICS are initiated at a terminal and are thus associated with a terminal. Tasks initiated by CICS management programs (for example, automatic task initiation by transient data control) may or may not be associated with a terminal. The contents of TCAFCAAA varies depending upon whether the attached task is associated with a terminal, as discussed in "Task Control Area (TCA)" on page 25.

The number of tasks that can be active within the system at a given time is limited by the availability of main storage and/or by the "maximum number of tasks" control established by the system programmer at system generation or initialization. A new task is initiated by CICS only when sufficient main storage is available to process it, otherwise the request is queued (stored) until sufficient main storage becomes available. Tasks initiated by CICS management modules (for example, terminal control) are subject to the maximum number of tasks limitation. Application program requests for attachment of tasks are not subject to this limitation and therefore are allowed to exceed the maximum.

If the DFHKC TYPE=ATTACH macro is used by the application programmer, he must provide the facility control area address and transaction identifier required by CICS to initiate a new task. The address and identifier can be specified in two ways:

1. By coding two instructions that assign a facility control area address to TCAKCFA and a transaction identification to TCAKCTI **prior** to issuing the DFHKC TYPE=ATTACH macro, or
2. By including the FCADDR and TRANSID operands in the DFHKC TYPE=ATTACH macro, which then stores the assigned values in TCAKCFA and TCAKCTI, respectively.

The facility control area address provides a pointer to information that the attaching task wishes to pass to the new task, for example, it can be the address of an entry in the DCT that is associated with a resource such as a data set.

The specified task is not attached if the transaction identifier is not in the PCT or the program name is not in the processing program table (the PPT). If this situation exists or the attached task abends, a message is sent to the terminal operator, but the attaching task is not notified of the condition. Therefore the DFHKC TYPE=ATTACH macro must be used with extreme caution by the application programmer.

Ownership of a task's terminal cannot be passed via the DFHKC TYPE=ATTACH macro to the new task. Instead, one of the following approaches should be used:

- Automatic task initiation through transient data management.
- Automatic task initiation through time management (interval control program); for example, a DFHIC TYPE=INITIATE macro with a zero interval.
- Identification of the transaction identifier to be used with the next input message from the terminal by means of the TRANSID operand of the DFHPC TYPE=RETURN macro.

The transaction identifier stored in TCAKCTI or specified in TRANSID is used only for the current ATTACH; it cannot be assumed that the value remains in TCAKCTI for the duration of the task. Subsequent ATTACHes must reestablish the contents of TCAKCTI.

The flowchart in Figure 25 on page 223 shows Task A attaching Task B and synchronizing the processing steps of both tasks through use of the facility control address passed to the newly created task at attach time. Since Task B is a nonterminal-oriented task, it is unable to use terminal control macros. FCADDR specifies the address of Task A's TCA; ECB1 and ECB2 are fields in the TWA for Task A.

Figure 25 on page 223 includes steps labeled "POST ECB". Posting an ECB entails setting on the appropriate bit in the ECB, which is a 4-byte field. In CICS/OS/VS, the bit to be set on (that is, set to "1") is bit 1 of byte 0; in CICS/DOS/VS, it is bit 0 of byte 2.

The following examples show how to set bits on for each programming language.

ASM (CICS/DOS/VS):

```
ECB1  DC   F'0'
      MVC  ECB1(3),=X'400080'
```

ASM (CICS/OS/VS):

```
ECB1  DC   F'0'
      MVI  ECB1,X'40'
```

COBOL (CICS/DOS/VS):

```
     77 ECB1 PIC S9(8) COMP VALUE ZERO.
        .
        .
  PROCEDURE DIVISION.
     COMPUTE ECB1 = 2**15+2**30.
```

COBOL (CICS/OS/VS):

```
     77 ECB1 PIC S9(8) COMP VALUE ZERO.
        .
        .
  PROCEDURE DIVISION.
     COMPUTE ECB1=2**30.
```

PL/I (CICS/DOS/VS):

```
DCL ECB1 BIT(32) ALIGNED INIT('0'B);
ECB1='0100000000000000l'B;
```

PL/I (CICS/OS/VS):

```
DCL ECB1 BIT(32) ALIGNED INIT('0'B);
ECB1='01'B;
```

The DFHPC TYPE=RETURN macro can be used to terminate any tasks initiated by the application programmer through use of the task control DFHKC TYPE=ATTACH macro.

The following example shows how to use the DFHKC macro to provide a facility control area address and transaction identifier:

```
DFHKC TYPE=ATTACH,FCADDR=FACCTL,
      TRANSID=TRN1
```

The following example shows the assembler language coding required to dynamically provide a facility control area address and transaction identification prior to issuing the DFHKC macro:

```
MVC TCAKCTI,=CL4'TRN1'
MVC TCAKCFA,=A(FACCTL)
      .
      .
DFHKC TYPE=ATTACH
```

The equivalent instructions in COBOL are:

```
MOVE 'TRN1' TO TCAKCTI.
MOVE FACADR TO TCAKCFA.
```

The equivalent instructions in PL/I are:

```
TCAKCTI='TRN1';
TCAKCFA=FACADR;
```

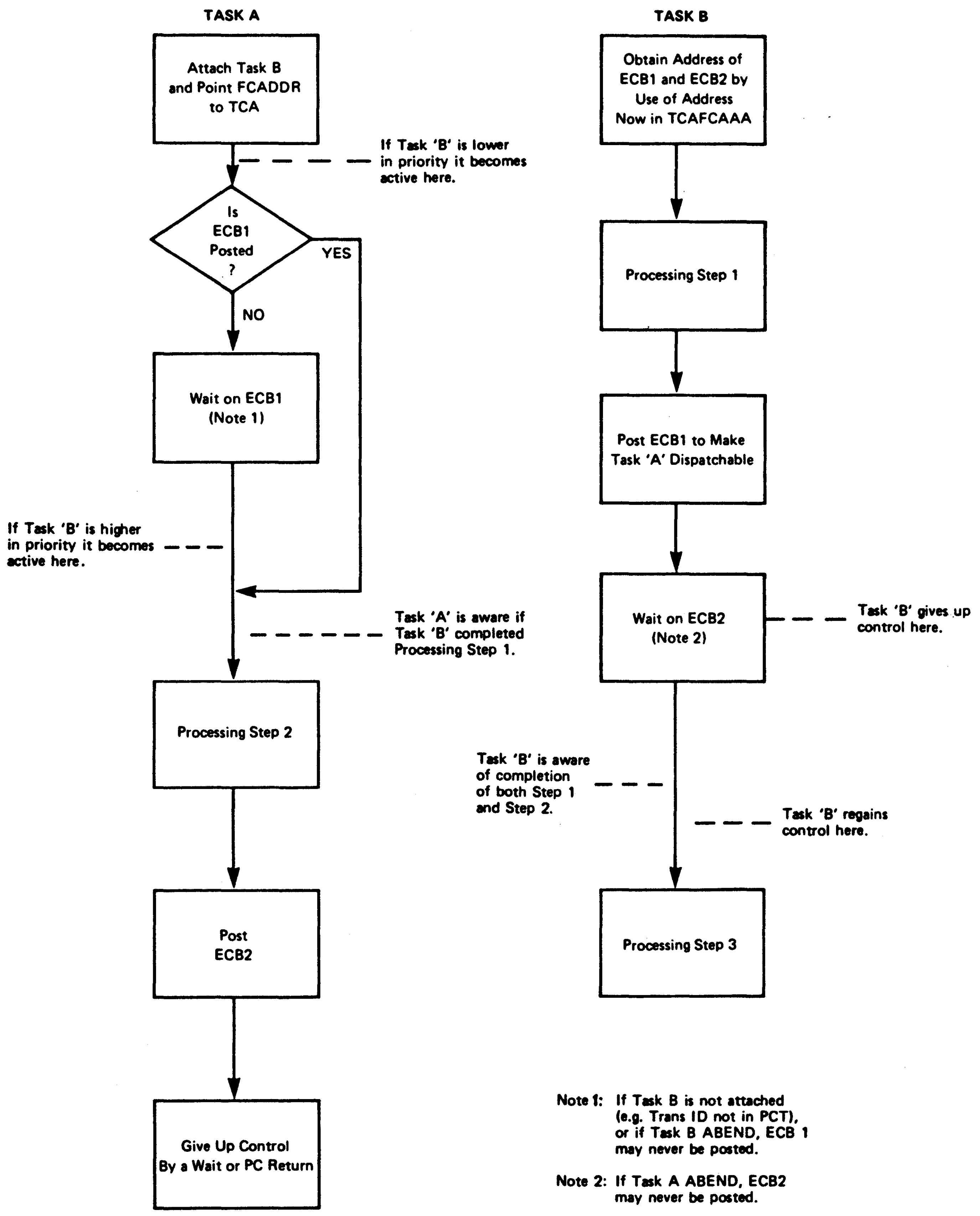

TASK A
TASK B
Attach Task B and Point FCADDR to TCA
If Task 'B' is lower in priority it becomes active here.
Is ECB1 Posted ?
YES
NO
Wait on ECB1 (Note 1)
If Task 'B' is higher in priority it becomes active here.
Task 'A' is aware if Task 'B' completed Processing Step 1.
Processing Step 2
Post ECB2
Give Up Control By a Wait or PC Return
Obtain Address of ECB1 and ECB2 by Use of Address Now in TCAFCAAA
Processing Step 1
Post ECB1 to Make Task 'A' Dispatchable
Wait on ECB2 (Note 2)
Task 'B' gives up control here.
Task 'B' is aware of completion of both Step 1 and Step 2.
Task 'B' regains control here.
Processing Step 3
Note 1: If Task B is not attached (e.g. Trans ID not in PCT), or if Task B ABEND, ECB 1 may never be posted.
Note 2: If Task A ABEND, ECB2 may never be posted.

Figure 25. Task Synchronization under CICS

CHANGE PRIORITY OF A TASK (TYPE=CHAP)

```
DFHKC TYPE=CHAP
      [,PRTY=priority value]
```

The overall transaction processing priority of a task is the sum of related transaction, terminal, and operator priorities as specified or established by default at system generation. This priority determines the position of the task in the dispatching priority queue and, therefore, its scheduling under CICS. The priority of an existing task can be changed by issuing the DFHKC TYPE=CHAP macro. The specified priority value must be in the range from 0 through 255, where 255 represents the highest priority. This task is placed below all other tasks of equal or higher priority in the dispatching priority queue.

The application programmer can include the PRTY=priority value operand in the DFHKC TYPE=CHAP macro to assign a new dispatching priority to a task. Alternatively, the programmer can assign a priority value to the dispatching priority field (TCATCDP) prior to issuing the DFHKC TYPE=CHAP macro.

A task can relinquish control to all tasks of equal or higher priority by issuing a DFHKC TYPE=CHAP macro. No priority value need be specified, and the current priority value of the task as stored in TCATCDP is not changed. However, the fact that the macro is issued permits control to be transferred from the task issuing the instruction to an equal or higher priority task within CICS. This capability is designed particularly for compute-bound tasks which, by continually demanding inordinate amounts of processor time, can significantly affect overall system performance.

The following example shows how to assign a new task dispatching priority value:

```
DFHKC TYPE=CHAP,PRTY=255
```

The following example shows the assembler language coding required to assign a dynamically selected priority value prior to issuing the DFHKC macro. This value can be specified as a binary, decimal, or hexadecimal number, depending on the programming language used.

```
MVI TCATCDP,X'FF'
     .
     .
DFHKC TYPE=CHAP
```

The equivalent instruction for COBOL is:

```
MOVE HIGH-VALUES TO TCATCDP.
```

The equivalent instruction for PL/I is:

```
TCATCDP='11111111'B;
```

SYNCHRONIZE A TASK (TYPE=WAIT)

The format of the DFHKC macro to synchronize the execution of a task with the completion of an event, or to relinquish control to a task of higher priority, is as follows:

```
DFHKC TYPE=WAIT
      [,DCI={SINGLE|LIST|DISP|CICS}]
      [,ECADDR=symb-addr]
```

The application programmer can synchronize a task with the completion of an event or one of a list of events initiated by the same task or by another task, or relinquish control to a task of higher dispatching priority, by issuing the DFHKC TYPE=WAIT macro. In the first case, this macro provides a method of directly relinquishing control to some other task until the event being waited on is completed. In the latter case, the task remains dispatchable. That is, execution of the task is resumed if no task of higher priority is ready to be processed.

If the task is to be synchronized with the completion of a single event or an event in a list of events, the application programmer must specify the address of either the single event control area or the list of event control areas. The address can be specified by including the ECADDR operand in the DFHKC TYPE=WAIT macro, or by coding a single instruction that places the event control address in TCATCEA prior to issuing the DFHKC TYPE=WAIT macro. In either case, the referenced event control area(s) must conform to the format and standard posting conventions of ECBs.

Examples showing how to post ECBs are given in the section "Initiate a Task (ATTACH)," earlier in the chapter. An event control area can also be the timer event control area referred to in a DFHIC TYPE=POST macro. (See the discussion of task synchronization in "Signal Expiration of a Specified Time (TYPE=POST)" on page 211.) In a CICS/OS/VS system, if two tasks are allowed to wait on the same event control area, CICS may terminate abnormally.

Synchronize a Task with a Single Event

The DFHKC TYPE=WAIT macro is also used by the application programmer to synchronize a task with the completion of a single event initiated by the same task or by another task.

The following example shows how to synchronize a task with a single event, providing the address of the appropriate event control area:

```
DFHKC TYPE=WAIT,DCI=SINGLE,
      ECADDR=EVENTCTL
```

The following example shows, in assembler language, how to synchronize a task with a single event, dynamically providing the address of the appropriate event control area prior to issuing the DFHKC macro.

```
ST    SINGADDR,TCATCEA

DFHKC TYPE=WAIT,DCI=SINGLE
```

The equivalent instruction for COBOL is:

```
MOVE SINGADDR TO TCATCEA.
```

The equivalent instruction for PL/I is:

```
TCATCEA=SINGADDR;
```

Synchronize a Task with One of a List of Events

The DFHKC TYPE=WAIT macro is also used by the application programmer to synchronize a task with the completion of one event of a list of events. This list consists of a series of contiguous four-byte fields, each field containing the address of a single event control area. The last four-byte field of the list contains binary ones, hexadecimal 'FF's, or the card code (multipunch) 12-11-0-7-8-9.

As there is a limited amount of ECB storage for DCI=LIST, care should be taken not to specify too high a number of ECBs. The amount of storage is described in the appropriate _CICS Installation and Operations Guide_.

The following example shows how to synchronize a task with one of a list of events, providing the address of the appropriate list of events:

```
DFHKC TYPE=WAIT,DCI=LIST,
      ECADDR=TOPOLIST
```

The following example shows, in assembler language, how to synchronize a task with one of a list of events, dynamically providing the address of the appropriate list of events prior to issuing the DFHKC macro.

```
ST    LISTADDR,TCATCEA

DFHKC TYPE=WAIT,DCI=LIST
```

The equivalent instruction for COBOL is:

```
MOVE LISTADDR TO TCATCEA.
```

The equivalent instruction for PL/I is:

```
TCATCEA=LISTADDR;
```

Relinquish Control to a Task of Higher Priority

The DFHKC TYPE=WAIT macro is also used by the application programmer to voluntarily relinquish control to a task of higher dispatching priority. Control is returned to the task issuing the macro if no other task of a higher priority is ready to be processed.

When binary synchronous communication lines are part of the user's configuration, these lines may time out if excessive processor time is required by an application program. One way to avoid this condition is to include one or more DFHKC TYPE=WAIT,DCI=DISP macros in the application program to voluntarily relinquish control before the line time-out can occur.

The following example shows how to voluntarily relinquish control to a task of higher dispatching priority:

```
DFHKC TYPE=WAIT,DCI=DISP
```

The DFHKC TYPE=WAIT macro differs from a TYPE=CHAP macro that does not indicate a priority in that the former relinquishes control only to a task of higher priority, while the latter may relinquish control to a task of either equal or higher priority.

ENQUEUE UPON A RESOURCE (TYPE=ENQ)

The format of the DFHKC macro to enqueue upon a resource, causing execution of a task to be synchronized with the availability of that resource, is as follows:

```
DFHKC TYPE=ENQ[,COND=YES|NO]
      [,QARGADR=symb-addr]
      [,QARGLNG=number]
```

In the CICS environment, where tasks are processed concurrently, it is sometimes desirable to protect a given resource from concurrent use by more than one task. In effect, the resource can be treated as serially reusable. To provide this resource protection, an installation convention must be established for all application programmers to follow.

The convention is based on use of the DFHKC TYPE=ENQ macro, identifying the resource by an address or a character-string argument. When executed, this macro causes further execution of the task issuing the instruction to be synchronized with the availability of the specified resource; control is returned to the task when the resource is available. When all tasks accessing a resource adhere to the convention of enqueuing upon the resource, the resource is afforded "single-server" protection.

When a single-server resource is being used by a task and other tasks concurrently enqueue upon the same resource, the first task to issue the DFHKC TYPE=ENQ macro receives the resource when it becomes available. The other tasks obtain the resource, in turn, in the order in which they enqueue upon it.

For assembler language application programs only, when COND=YES is specified, control is returned to the requesting transaction whether or not the requested resource is available; task control places a return code in TCATCTR indicating the result of the enqueue request. The return codes and their meanings are:

TCATCOK	The requested resource has been given to the requestor
TCATCONQ	The requested resource is not available
TCADUPQ	The requestor already has the requested resource

COND=NO is the default, and when this is specified, either explicitly or by default, the normal enqueue mechanism operates (that is, the requestor is enqueued upon the resource if it is not immediately available).

When issuing the DFHKC TYPE=ENQ macro, the application programmer must identify the single-server resource being enqueued upon by one of the following methods:

1. Specify a symbolic main storage address that represents the single-server resource. The application programmer must provide the symbolic main storage address in the DFHKC TYPE=ENQ macro or by coding instructions (prior to issuing the DFHKC TYPE=ENQ macro) that place the address in the low-order three bytes of TCATCQA, a four-byte field. Binary zeros must be placed in the high-order byte.

2. Specify a symbolic main storage address that contains a unique character-string argument (for example, an employee name) that represents the single-server resource. The unique argument may be up to 255 bytes in length, beginning at the location pointed to by the contents of the specified address. The application programmer must provide the symbolic main storage address and the length in the DFHKC TYPE=ENQ macro or by coding instructions (prior to issuing the DFHKC TYPE=ENQ macro) that place the address in the low-order three bytes of TCATCQA, a four-byte field, and the length (in bytes) in the high-order byte. CICS task control makes a copy of this pointer in its storage for use in controlling the resource.

DEQUEUE UPON A RESOURCE (TYPE=DEQ)

The format of the DFHKC macro to dequeue upon a resource (effectively, to revoke a preceding enqueue request upon that resource) is as follows:

```
DFHKC TYPE=DEQ
      [,QARGADR=symb-addr]
      [,QARGLNG=number]
```

When issuing the DFHKC TYPE=DEQ macro, the application programmer must identify the resource he is dequeuing by the method that was used in enqueuing. The COBOL programmer may find it convenient to use the program control DFHPC TYPE=COBADDR macro (see the example below) if preloading of the address is desired.

If a task enqueues upon a resource but does not dequeue it, task control automatically dequeues the single-server protection request upon termination of the task. The single server protection request is also dequeued automatically if necessary during sync point processing.

The following examples show how to enqueue upon a single-server resource using method 1, above. Substituting "DEQ" for "ENQ" in these examples illustrates the ways in which the application programmer can release single-server protection from a resource prior to termination of the associated task.

ASM:

```
         COPY DFHCSADS
CSAWABA  DS   F
         .
         .
         DFHKC TYPE=ENQ,
               QARGADR=CSAWABA

                       OR

       LA   WORKREG,CSAWABA
       ST   WORKREG,TCATCQA
       .
       .
       DFHKC TYPE=ENQ
```

COBOL:

```
       01  DFHCSADS COPY DFHCSADS.
           02  CSAWABA PIC X(50).
       .
       .
       MOVE ZEROS TO TCATCQA.
       DFHKC TYPE=ENQ,
             QARGADR=CSAWABA

                       OR

       DFHPC TYPE=COBADDR,
             LABEL=CSAWABA
       MOVE TCAPCLA TO TCATCQA.
       .
       .
       DFHKC TYPE=ENQ
```

PL/I:

```
       %INCLUDE DFHCSADS;
       DCL 1 DFHEXCSA BASED(CSACBAR),
           2 FILLER CHAR (512),
           2 CSAWABA CHAR (50);
       .
       .
       DFHKC TYPE=ENQ,
             QARGADR=CSAWABA

                       OR

       TCATCQA=ADDR(CSAWABA);
       .
       .
       DFHKC TYPE=ENQ
```

The following examples show how to enqueue upon a single-server resource using method 2. The resource to be enqueued upon is identified by the nine-character social security number in a field labeled SOCSECNO. Task control makes a copy of this field for its use in controlling the resource.

Substituting "DEQ" for "ENQ" in these examples illustrates the ways in which the application programmer can release single-server protection from a resource prior to termination of the associated task.

ASM:

```
       DFHKC TYPE=ENQ,
             QARGADR=SOCSECNO,
             QARGLNG=9

                   OR

       LA   WORKREG,SOCSECNO
       ST   WORKREG,TCATCQA
       MVI  TCATCQA,X'09'
       .
       .
       DFHKC TYPE=ENQ
```

COBOL:

```
       DFHKC TYPE=ENQ,
             QARGADR=SOCSECNO,
             QARGLNG=9
```

PL/I:

```
       DFHKC TYPE=ENQ,
             QARGADR=SOCSECNO,
             QARGLNG=9

                   OR

       %INCLUDE DFHTCADS;
       DCL 1 DFHEXTCA BASED(TCACBAR),
           2 FILLER CHAR (20),
           2 TCATCQAL BIT(8);
       .
       .
       TCATCQA=ADDR(SOCSECNO);
       TCATCQAL='00001001'B;
       .
       .
       DFHKC TYPE=ENQ
```

DECLARE A TASK TO BE PURGEABLE (TYPE=PURGE)

The format of the DFHKC macro to declare that a task may be purged if a system stall condition occurs is as follows:

```
DFHKC TYPE=PURGE
```

Certain overload conditions, where all of a given system resource (for example, main storage) has been allocated and where each task requires still more of that resource, can occur in CICS. The result is a situation in which no task is able to continue processing and no new task can be initiated; the system stalls.

CICS has the capability of detecting certain system stall conditions and taking corrective action, which consists, in part, of purging (deleting) the lowest priority task in the system that is designated as stall purgeable.

A task is initially defined as purgeable or not purgeable in the program control table (PCT) entry associated with the transaction identification for that task. This entry is established by the system programmer at system generation. The application programmer can dynamically change the purgeability status of a task by issuing the

```
  DFHKC TYPE=PURGE
```

macro to indicate that the task is purgeable, or the

```
  DFHKC TYPE=NOPURGE
```

macro to indicate that the task is not purgeable. The designated status remains in effect for that task until another change is initiated or until the task is terminated. For example, a long-running task may issue a DFHKC TYPE=NOPURGE macro prior to critical processing, then issue a DFHKC TYPE=PURGE macro after that processing is completed. This ensures that the task is not stall-purged during the critical processing.

DECLARE A TASK TO BE NONPURGEABLE (TYPE=NOPURGE)

The format of the DFHKC macro to declare that a task cannot be purged if a system stall condition occurs is as follows:

```
  DFHKC TYPE=NOPURGE
```

The PURGE and NOPURGE options of the DFHKC macro are intended to be used as temporary overrides to the SPURGE specification in the DFHPCT TYPE=ENTRY macro for a task. For example, if a DFHKC TYPE=NOPURGE macro is issued in a program for a task, the task cannot be purged even though SPURGE=YES is specified in the DFHPCT TYPE=ENTRY system macro for the task at system generation. See the appropriate CICS Resource Definition manual.

OPERANDS OF DFHKC MACRO

COND=
specifies whether or not an enqueue is to be conditional (assembler language only).

YES
the enqueue request is conditional. Control is returned to the requestor whether or not the requested resource is available. A return code at TCATCTR indicates the result of the request:

TCATCOK	The resource has been given to the requestor
TCATCONQ	The resource is not available
TCADUPQ	The requestor already has the resource

NO
the enqueue is not conditional. If the requested resource is not immediately available, the requesting transaction will be enqueued upon it. COND=NO is the default.

DCI=
specifies when synchronization is to occur.

SINGLE
specifies that the task is to be synchronized with the completion of a single event.

LIST
specifies that the task is to be synchronized with the completion of one event in a list of events.

DISP
specifies that the task is to give up control to any higher priority task that is ready to be processed; if none exists, control is to be returned to this task.

CICS (OS only and ASM only)
specifies that the ECB will be posted by another transaction rather than by the operating system. This option means that the ECB will not be added to the operating system WAIT macro issued by CICS. ECBs to be posted by other transactions should reside in permanent storage.

Tasks that are to synchronize with each other as illustrated in Figure 25 on page 223, may do so by using either DCI=CICS or DCI=SINGLE (CICS/DOS/VS must use DCI=SINGLE only). DCI=CICS must be used if more than one synchronizing task is going to wait on the same ECB. In all other cases it is preferable to use DCI=SINGLE.

ECADDR=symb-addr
is used with DCI=SINGLE or DCI=LIST to specify the symbolic address of the single event control area or

list of event control areas identifying the event with which this task is to be synchronized; if omitted when SINGLE or LIST is specified, the address is assumed to be in TCATCEA.

FCADDR=symb-addr
is the symbolic address of the facility control area (FCA) associated with this task; if omitted, the address is assumed to be in TCAKCFA.

PRTY=priority value
is a decimal numeral in the range from 0 through 255 to be taken as the priority value for this task; if omitted, the priority value is assumed to be in TCATCDP.

QARGADR=symb-addr
is either the symbolic address of the resource to be enqueued or dequeued, or the symbolic address of a location that contains a unique argument (for example, an employee name) that represents the resource. If this operand is omitted, the address is assumed to be in the three low-order bytes of TCATCQA, a four-byte field.

QARGLNG=number
is the length, in bytes, of the resource to be enqueued upon or to be dequeued. This operand is needed only if the QARGADR operand is a unique argument that represents the resource to be enqueued. If omitted in such a case, the contents of the high-order byte of TCATCQA are assumed to be the length of the argument. COBOL programs must not use this operand unless the QARGADR operand is used.

TRANSID=name
is the transaction identification for the task; if omitted, the transaction identification is assumed to be in TCAKCTI.

CHAPTER 5.4. PROGRAM CONTROL (DFHPC MACRO)

All program communication within CICS is accomplished by program management. The program management macro (DFHPC) is used to request any of the following services:

- Link one user-written application program to another, anticipating subsequent return to the requesting program (TYPE=LINK).
- Transfer control from one user-written application program to another, anticipating no return to the requesting program (TYPE=XCTL).
- Load a designated application program, table, or map (generally, for use with basic mapping support) into main storage and return control to the requesting program (TYPE=LOAD).
- Return control from one user-written application program to another or to CICS (TYPE=RETURN).
- Delete a previously loaded application program from main storage (TYPE=DELETE).
- Abnormally terminate a transaction and its related task (TYPE=ABEND).
- Activate, cancel, or reactivate an exit that permits user-written abnormal termination processing (TYPE=SETXIT or TYPE=RESETXIT).
- Convert a symbolic label in a COBOL program into an address which is returned in TCAPCLA (TYPE=COBADDR).

Application programs running under CICS are executed at various logical levels. For example, where one user-written application program is linked to another, the linked-to program is considered to reside at the next lower logical level. Where control is simply transferred from one application program to another, the two programs are considered to reside at the same logical level. A DFHPC TYPE=LINK macro is used for the former; a DFHPC TYPE=XCTL macro (where XCTL means transfer control) is used for the latter. Figure 26 on page 232 illustrates this difference between program linkage and transfer of program control. Each of the programs shown in this figure may have been written in any of the CICS-supported languages (assembler language, COBOL, or PL/I). Use of LINK, XCTL, RETURN, and ABEND is explained in greater detail below.

Tasks can share the use of common work areas. However, each task requires the use of a unique intermediate storage area, such as the transaction work area (TWA), to retain information needed upon subsequent return to that task. The application programmer must provide addressability to that intermediate storage area by symbolically defining it in his program.

Parameters can be passed from one program to another in the same task through user-defined storage areas, for example, the transaction work area (TWA), the terminal input/output area (TIOA), the terminal control table terminal entry (TCTTE), or the file work area (FWA).

CICS automatically saves program control information and general-purpose registers, when applicable, in the task control area (TCA). CICS automatically restores general-purpose registers, as necessary, to return control to a program. The name of any program referred to in a request for program services must have been placed in the processing program table (PPT) prior to execution of CICS. If the program has been defined with RELOAD=YES in the DFHPPT TYPE=ENTRY system macro, it is the user's responsibility to delete the program by means of a storage control FREEMAIN rather than by a program control DELETE. Eight bytes must be subtracted from the address at which the program is loaded before issuing the FREEMAIN.

PASS PROGRAM CONTROL ANTICIPATING RETURN (TYPE=LINK)

The format of the DFHPC macro to pass control to an application program at the next lower logical level is as follows:

```
DFHPC TYPE=LINK
      [,PROGRAM=name]
      [,COND=YES]
      [,NORESP=symb-addr]
      [,PGMIDER=symb-addr]
```

When a DFHPC TYPE=RETURN macro is executed in the linked-to program, control is returned to the first program at the next sequential (executable) instruction.

The application programmer must specify the name of the program to which control

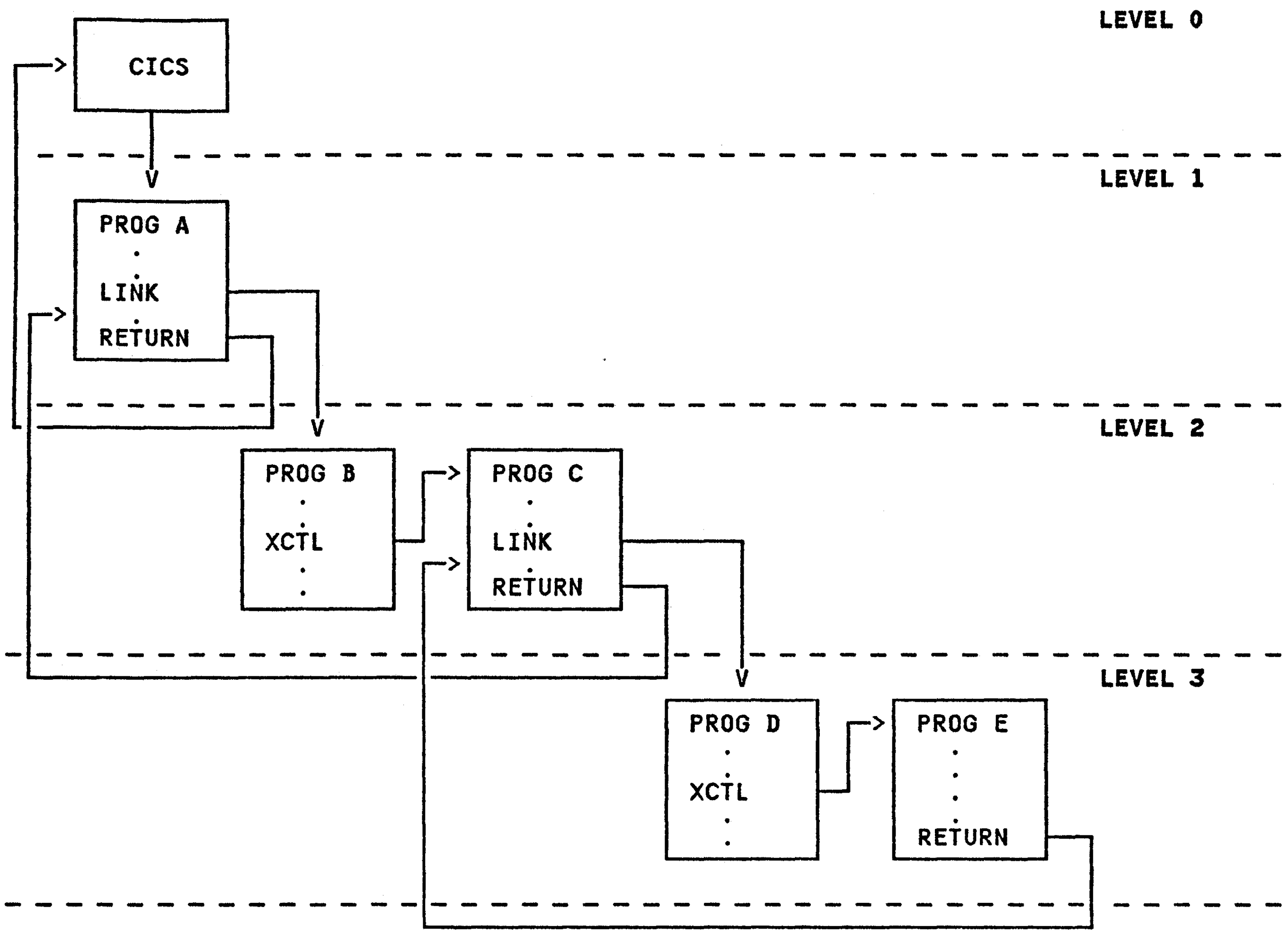


Figure 26. Logical Relationship of Application Programs

is to be passed in the PROGRAM operand or in a single instruction that places the program name in TCAPCPI prior to issuing this macro. The COND operand specifies that control will be returned to the first program if the specified program is disabled or its name cannot be found in the PPT.

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

```
DFHPC TYPE=LINK,PROGRAM=PROG1
```

The following example shows, for assembler language, how to link to an application program by means of an instruction (MVC), executed prior to the DFHPC TYPE=LINK macro, that places the linked-to program name in the TCA.

```
MVC   TCAPCPI,=CL8'PROG1'
         .
         .
DFHPC TYPE=LINK
```

The equivalent instruction for COBOL is:

```
MOVE 'PROG1' TO TCAPCPI.
```

The equivalent instruction for PL/I is:

```
TCAPCPI='PROG1';
```

__TRANSFER PROGRAM CONTROL (TYPE=XCTL)__

The format of the DFHPC macro to pass (transfer) control to an application program at the same logical level is as follows:

```
DFHPC TYPE=XCTL
      [,PROGRAM=name]
```

This macro specifies that program control is transferred from one user-written application program to another at the same logical level. The program from which control is transferred is released. Any return from the transferred-to program is to a program from which there was an exit at the next higher logical level. If there is no user-written application program at the next higher logical level, control is returned to CICS.

The application programmer must specify the name of the program to which control is to be transferred in the PROGRAM operand or in a single instruction that places the program name in TCAPCPI prior to issuing this macro. The field TCAPCPI is eight bytes in length. If the program name is less than eight bytes, the field must be padded on the right with blanks.

The following example shows how to request a transfer of control to a particular application program:

```
  DFHPC TYPE=XCTL,PROGRAM=PROG2
```

The following example shows, for assembler language, how to transfer control to an application program by means of an instruction (MVC), executed prior to the DFHPC TYPE=XCTL macro, that places the transferred-to program name in the TCA.

```
  MVC   TCAPCPI,=CL8'PROG2'

  DFHPC TYPE=XCTL
```

The equivalent instruction for COBOL is:

```
  MOVE 'PROG2' TO TCAPCPI.
```

The equivalent instruction for PL/I is:

```
  TCAPCPI='PROG2';
```

LOAD A PROGRAM (TYPE=LOAD)

The format of the DFHPC macro to load a program, table, or map from its location in a CICS program library is as follows:

```
DFHPC TYPE=LOAD
      [,PROGRAM=name]
      [,LOADLST=NO]
      [,COND=YES]
      [,NORESP=symb-addr]
      [,PGMIDER=symb-addr]
```

This macro specifies that programs, tables, or maps are to be fetched from the library where they reside and loaded into main storage. This facility is used to (1) load a program that will be used repeatedly, thereby reducing system overhead through a one-time load, (2) load a table to which control is not to be passed, or (3) load a map to be used in a mapping operation (see "Chapter 4.3. Basic Mapping Support" on page 143). CICS returns the address of the loaded program in TCAPCLA.

The loaded program remains in main storage until the DFHPC TYPE=DELETE macro is issued or until the task that issued the DFHPC TYPE=LOAD is terminated, either normally or abnormally (unless LOADLST=NO is specified). If LOADLST=NO is specified, the loaded program remains resident until it is deleted by this, or another, task.

The application programmer must provide the name (identification) of the program to be loaded in the DFHPC TYPE=LOAD macro or in a single instruction that places the program name in TCAPCPI prior to issuing the DFHPC TYPE=LOAD macro.

The following example shows how to load a user-written application program:

```
  DFHPC TYPE=LOAD,PROGRAM=PROG3
```

The following example shows, for assembler language, how to load an application program by means of an instruction (MVC), executed prior to the DFHPC TYPE=LOAD macro, that places the program name in the TCA.

```
  MVC   TCAPCPI,=CL8'PROG3'

  DFHPC TYPE=LOAD
```

The equivalent instruction for COBOL is:

```
  MOVE 'PROG3' TO TCAPCPI.
```

The equivalent instruction for PL/I is:

```
  TCAPCPI='PROG3';
```

RETURN PROGRAM CONTROL (TYPE=RETURN)

The format of the DFHPC macro to return control from an application program to the program at the next higher logical level is as follows:

```
DFHPC TYPE=RETURN
      [,TRANSID=transaction code]
```

When this macro is executed in a lower level (linked-to) program, it restores the registers of the higher level (linked-from) program to their contents at the time the DFHPC TYPE=LINK was issued and releases save areas for the lower-level program. In general, the program to which control is returned must have relinquished control by execution of a DFHPC TYPE=LINK macro and must reside one logical level higher than the program returning control. Upon normal termination of transaction processing, control is returned to CICS.

If no default transaction code has been assembled into the terminal control table terminal entry (TCTTE) for a particular terminal, the application programmer can specify the transaction identification for the next program to be associated with that terminal in either of two ways: (1) by including the desired transaction identification in the DFHPC TYPE=RETURN macro, or (2) by coding a single instruction that places the desired transaction identification in TCANXTID prior to issuing the DFHPC TYPE=RETURN macro. By doing so, the programmer ensures that subsequent unsolicited input can be entered from the terminal without the specification of a transaction identification. A flexible means of starting the next task is thus provided.

Note, however, that the methods of specifying the transaction described above may be overridden by issuing BMS paging commands. (See "Terminal-Oriented Task Identification" on page 115, for a precise description.)

Note also that if the terminal is in TRANSCEIVE status, a task started by ATI (automatic task initiation) may run before a task started by the next input from the terminal. In this case, CICS compares the transaction identifier of the task started by ATI with that specified in the TRANSID operand. If they are the same, CICS erases its record of the TRANSID transaction identifier because it assumes that the task started by ATI will perform the same function.

DELETE A LOADED PROGRAM (TYPE=DELETE)

The format of the DFHPC macro to delete a previously loaded program is as follows:

```
DFHPC TYPE=DELETE
      [,PROGRAM=name]
```

This macro specifies that a program previously loaded through use of the DFHPC TYPE=LOAD macro with or without the LOADLST=NO operand is to be deleted. If the DFHPC TYPE=LOAD macro includes LOADLST=NO, the loaded program is deleted only in response to a DFHPC TYPE=DELETE macro. If LOADLST=NO is not specified, the loaded program can be deleted by a DFHPC TYPE=DELETE request, or it will be automatically deleted when the task that issued the load request is terminated.

The application programmer must specify the name (identification) of the program to be deleted in the DFHPC TYPE=DELETE macro or in an instruction that places the program name in TCAPCPI prior to issuing the DFHPC TYPE=DELETE macro.

The following example shows how to delete a user-written application program loaded in response to a DFHPC TYPE=LOAD macro.

```
DFHPC TYPE=DELETE,PROGRAM=PROG4
```

The following example shows, for assembler language, how to delete an application program by means of an instruction (MVC), executed prior to the DFHPC TYPE=DELETE macro, that places the program name in the TCA.

```
MVC   TCAPCPI,=CL8'PROG4'

DFHPC TYPE=DELETE
```

The equivalent instruction for COBOL is:

```
MOVE 'PROG4' TO TCAPCPI.
```

The equivalent instruction for PL/I is:

```
TCAPCPI='PROG4';
```

ABNORMALLY TERMINATE A TRANSACTION (TYPE=ABEND)

The format of the DFHPC macro to abnormally terminate a transaction (task) is as follows:

```
DFHPC TYPE=ABEND
      [,ABCODE={value|YES}]
      [,CANCEL=YES]
```

This macro specifies that a transaction and its related task is to be terminated abnormally. If a task is attached by another task, only the task that issues the ABEND is terminated. The main storage associated with the terminated transaction is released. If CANCEL=YES is specified, all exits established by DFHPC TYPE=SETXIT macros at any level in the task are canceled.

The application programmer can request a dump of main storage related to the terminated transaction. The request must specify a four-character abnormal termination code that dump control will place in the formatted storage dump to identify the ABEND condition. This code can be specified in either of two ways:

1. It can be specified in the TYPE=ABEND macro, as follows:

   ```
   DFHPC TYPE=ABEND,ABCODE=1234
   ```

2. It can be placed in TCAPCAC before issuing the macro, as shown in the following assembler language example:

   ```
   MVC   TCAPCAC,=CL4'1234'

   DFHPC TYPE=ABEND,ABCODE=YES
   ```

   The equivalent instruction in COBOL is:

   ```
   MOVE '1234' TO TCAPCAC.
   ```

   The equivalent instruction in PL/I is:

   ```
   TCAPCAC='1234';
   ```

Note: The DFHPC macro will preserve the original contents of the two bytes starting at TCAPCTR by moving them to TCACCSV1. Thus a dump will contain the response codes from the last CICS service call. If ABCODE (but not ABCODE=YES) is specified, the original contents of TCAPCAC will also be preserved in TCACCSV2. If ABCODE=YES is specified, and you wish the original contents of TCAPCAC to appear in the dump, they must be stored elsewhere before you store the ABEND code there. It is therefore preferable to use method 1 above when specifying a dump code.

ACTIVATE OR CANCEL AN EXIT FOR ABNORMAL TERMINATION PROCESSING (TYPE=SETXIT)

The format of the DFHPC macro to activate or cancel an exit to a user-written routine or program to be executed upon abnormal termination of a task is as follows:

```
DFHPC TYPE=SETXIT
      [,PROGRAM={name|YES}]|
      [,ROUTINE={symb-addr|YES}]
      [,NORESP=symb-addr]
      [,PGMIDER=symb-addr]
```

This macro specifies that a user exit is to be:

1. Activated, if the PROGRAM or ROUTINE operand is specified
2. Canceled, if no additional operands are specified.

During abnormal termination of a task, a program-level ABEND exit facility is provided in CICS program control so that a user-written exit routine can be executed if desired. One example of a function performed by such a routine is the "cleanup" of a program that has started but not completed normally. An ABEND exit within an application program is activated in response to the DFHPC TYPE=SETXIT macro. The application programmer must specify the name of a program, or (for assembler language and COBOL programs) the address of a routine, to be given control when an abnormal termination condition occurs. The program name or routine address can be specified in the DFHPC TYPE=SETXIT macro, or placed in the appropriate field in the TCA before the macro is issued. A program name is placed in TCAPCPI; a routine address is placed in TCAPCERA. The PROGRAM and ROUTINE operands are mutually exclusive.

A DFHPC TYPE=SETXIT macro in which a program or routine name is specified overrides (effectively, replaces) any preceding DFHPC TYPE=SETXIT macro in any application program at the same logical level. (Logical levels are illustrated in Figure 26 on page 232.) Thus, each application program of a transaction can have its own exit, but only one exit at each logical level can be active. To cancel a previously established exit at the logical level of the application program in control, the application programmer can issue a DFHPC TYPE=SETXIT macro in which neither the program name nor the routine name operand is specified.

When a task ABEND occurs, CICS 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 27 on page 236, which also shows how subsequent ABEND exit processing is determined by the user's exit routine or program.

Note: When a DFHPC TYPE=XCTL macro is to be used to transfer control from an application program, and an exit routine (rather than a program) is specified, the exit will be reset. This situation will not occur if an exit program is specified, instead of a routine. Routines can be used without risk in application programs that do not use a DFHPC TYPE=XCTL macro.

To prevent recursive ABENDs in an exit routine, CICS deactivates an exit upon entry to the exit routine. If attempting a retry of the operation, the programmer can branch to a point in the program that was in control at the time of the ABEND and issue the DFHPC TYPE=RESETXIT macro to reactivate the exit. The user can also use this macro to reactivate an exit that was canceled

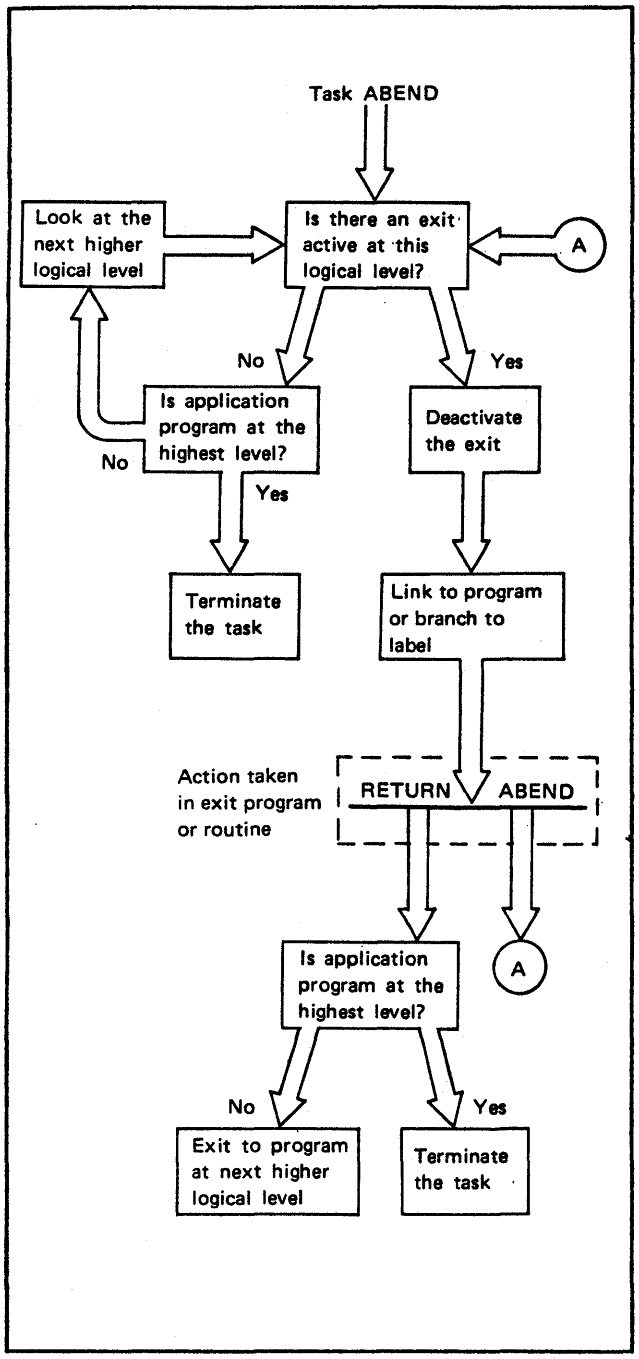

Task ABEND
Look at the next higher logical level
Is there an exit active at this logical level?
A
No
Yes
Is application program at the highest level?
No
Yes
Deactivate the exit
Terminate the task
Link to program or branch to label
Action taken in exit program or routine
RETURN
ABEND
Is application program at the highest level?
A
No
Yes
Exit to program at next higher logical level
Terminate the task

Figure 27. ABEND Exit Processing

previously as described above. No additional parameters are required.

Upon entry to an exit program, no addressability can be assumed other than that normally assumed for an application program coded in the language. If the exit logic is in the form of a routine, the amount of addressability varies with the source language, as detailed under "Creating a Program ABEND Exit" in the appropriate CICS Customization Guide.

For additional information concerning preparation of the exit routine, see that guide.

The following example shows how to establish a program as an exit:

```
  DFHPC TYPE=SETXIT,PROGRAM=EXITPGM
```

The following examples show how to establish a program as an exit by dynamically storing the program name prior to executing the DFHPC TYPE=SETXIT macro.

ASM:

```
       MVC    TCAPCPI,=CL8'EXITPGM'
        .
        .
       DFHPC TYPE=SETXIT,PROGRAM=YES
```

COBOL:

```
       MOVE   'EXITPGM' TO TCAPCPI.
        .
        .
       DFHPC TYPE=SETXIT,PROGRAM=YES
```

PL/I:

```
       TCAPCPI='EXITPGM';
        .
        .
       DFHPC TYPE=SETXIT,PROGRAM=YES
```

The following examples show how to establish a routine as an exit by dynamically storing the address of the routine prior to executing the DFHPC TYPE=SETXIT macro. (Note that routines cannot be established as exits in PL/I application programs.)

ASM:

```
       LA     14,EXITRTN
       ST     14,TCAPCERA
        .
        .
       DFHPC TYPE=SETXIT,ROUTINE=YES
```

COBOL:

```
       DFHPC TYPE=COBADDR,LABEL=EXITRTN
       MOVE  TCAPCLA TO TCAPCERA.
        .
        .
       DFHPC TYPE=SETXIT,ROUTINE=YES
```

REACTIVATE AN EXIT FOR ABEND PROCESSING (TYPE=RESETXIT)

The format of the DFHPC macro to reactivate an exit to a user-written routine or program to be executed upon abnormal termination of a transaction (task) is as follows:

```
DFHPC TYPE=RESETXIT
```

This macro specifies that an exit to user-written abnormal termination processing is to be reactivated after a preceding application program cancellation or CICS cancellation upon execution of the exit routine.

CONVERT SYMBOLIC LABEL TO ADDRESS (TYPE=COBADDR)

The format of the DFHPC macro to convert a symbolic label appearing in a COBOL program to an address is as follows:

```
DFHPC TYPE=COBADDR
      ,LABEL=symbolic label
```

This macro specifies that the address of the location represented by a symbolic label is to be returned in TCAPCLA to the application program. The first byte of TCAPCLA can be nonzero, and should therefore be initialized if necessary.

A comparable facility is available within both PL/I and assembler language; this macro is designed to provide the capability for COBOL programmers. COBOL support must have been generated within CICS to support COBOL programs.

TEST RESPONSE TO A REQUEST FOR PROGRAM SERVICES (TYPE=CHECK)

The format of the DFHPC macro to test the CICS response to a request for program management services is as follows:

```
DFHPC TYPE=CHECK
      [,NORESP=symb-addr]
      [,PGMIDER=symb-addr]
```

PROGRAM CONTROL RESPONSE CODES

To test the response code the application programmer must know (1) the CICS response codes and their meanings, and (2) the symbolic label by which he can refer to the response code; these are shown below. In an assembler language or PL/I program the response code will be found in TCAPCTR. In a COBOL program, the response code will be found in TCAPCRC.

Condition	**ASM**	**COBOL**	**PL/I**
NORESP	X'00'	LOW-VALUES (PCARCNR)	00000000
PGMIDER	X'01'	12-1-9 (PCPGMIDER)	00000001

The names enclosed in parentheses in the COBOL column indicate the condition names generated by CICS. These names may be used in testing for the conditions in a COBOL program.

Note: Because the multipunch codes to be checked in a COBOL program commonly correspond to unprintable characters, an alternative facility is provided in CICS for use by the COBOL programmer. In COBOL the response code can be referred to by a condition name, formed as a two-character identification of the CICS management module providing the requested service, followed by the keyword for the condition being checked (for example, PCNORESP). Use of this approach is illustrated in the examples at the end of this discussion.

To provide for the possibility of failure to find a requested program in the processing program table (PPT), or finding a disabled program in response to DFHPC TYPE=LINK or TYPE=LOAD, the COND operand must be included in these macros. This operand causes control to be passed to the user-specified exception-handling routine specified in the PGMIDER operand if the error occurs. If the COND operand is not specified and the error occurs, the requesting program is abnormally terminated with an APCT ABEND code.

The following examples show how to examine the response code provided by CICS at TCAPCTR (for assembler language or PL/I) or TCAPCRC (for COBOL) and transfer control to an appropriate user-written error-handling routine. The alternative approach available to COBOL programmers is also shown.

ASM:

```
       DFHPC TYPE=SETXIT,PROGRAM=MYPROG
       CLI     TCAPCTR,X'00'
       BE      GOOD
       DFHPC   TYPE=ABEND
GOOD   DS      0H
       .
       .
```

COBOL:

```
       DFHPC TYPE=SETXIT,PROGRAM=MYPROG
       IF TCAPCRC = ' ' THEN GO TO GOOD.
       DFHPC TYPE=ABEND
GOOD.
       .
       .
```

Alternatively, the COBOL programmer may test responses by using the CICS generated condition names:

```
       IF PCNORESP THEN GO TO GOOD.
```

PL/I:

```
       DFHPC TYPE=SETXIT,PROGRAM=MYPROG
       IF TCAPCTR='0'B THEN GO TO GOOD;
       DFHPC TYPE=ABEND
GOOD:  . . .
```

OPERANDS OF DFHPC MACRO

ABCODE=
indicates that main storage related to the transaction is to be dumped and provides a four-character abnormal termination code to identify the output dump.

value
is a combination of four alphabetic, numeric, and/or special characters to be printed as the abnormal termination code.

YES
indicates that the abnormal termination code has been placed in TCAPCAC.

Note: If a dump is requested, any information in the common control area of the application program communication section of the TCA is likely to be different in the dump. The DFHPC TYPE=ABEND macro preserves the original contents of the overwritten fields in the TCA by moving the two bytes starting at TCAPCTR to TCACCSV1. If an explicit abnormal termination code is specified, the macro will also move the original contents of TCAPCAC to TCACCSV2. If ABCODE=YES is specified, and the original contents of TCAPCAC are required in the dump, the information must be stored elsewhere before storing an abnormal termination code there. If the ABCODE operand is not specified, the macro does not use the TCAPCAC field.

CANCEL=YES
indicates that all exits established by DFHPC TYPE=SETXIT macros at any level in the task are to be canceled; in effect, they are ignored.

COND=YES
indicates that control is to be returned to the program issuing the macro if the program specified in the PROGRAM operand cannot be found in the PPT or is disabled. If this operand is omitted and the requested program cannot be found or is disabled, the task is abnormally terminated with the ABEND code "APCT".

LABEL=symbolic label
is the symbolic label that represents the location in the COBOL program for which the address is required.

LOADLST=NO
indicates that the loaded module is not to be deleted when the task issuing the load request is terminated; that is, the loaded module remains resident until deleted at the request of this task or of another task.

NORESP=symb-addr
specifies the entry label of the user-written routine to which control is to be passed if no errors occur during program control processing. NORESP signifies "normal response."

PGMIDER=symb-addr
specifies the entry label of the user-written routine to which control is to be passed if the requested program cannot be found in the PPT or is disabled. Control will not be passed unless the COND operand is specified also in the TYPE=LINK or TYPE=LOAD macros, or unless the PROGRAM operand is specified also in the TYPE=SETXIT macro.

PROGRAM=name
is the name of the program to which control is to be passed or the name of the program, table, or map to be loaded; if omitted, the name is assumed to be in TCAPCPI. TCAPCPI is an eight-character field; names less than eight characters must be padded right with blanks. If the requested program cannot be found or is disabled, the task is abnormally terminated with the ABEND code "APCT".

For the TYPE=SETXIT macro only, PROGRAM=name specifies the name, in the PPT, of the program to receive control if abnormal termination occurs. PROGRAM=YES specifies that the name of the program to receive control has been placed in TCAPCPI.

ROUTINE=
identifies the routine to receive control if abnormal termination occurs. (This operand applies only to assembler language and COBOL programs.)

There is a risk involved in the use of this operand if the application program transfers control using DFHPC TYPE=XCTL. The occurrence of a short-on-storage condition could lead to the storage used by this application program being re-used, and any reference to the re-used storage would have unpredictable results.

symb-addr
is the symbolic address of the routine to receive control.

YES
indicates that the address of the routine to receive control has been placed in TCAPCERA.

TRANSID=transaction code
is the transaction identification to be used with the next input message entered from the terminal with which this requesting task has been associated prior to this request for return of control.

CHAPTER 5.5. STORAGE CONTROL (DFHSC MACRO)

Storage management controls all main storage for CICS and for user-written application programs. Requests to acquire or release main storage are communicated to CICS storage control by means of the storage management macro (DFHSC).

CICS management programs automatically issue requests for main storage to provide input/output areas, program load areas, and user-defined work areas needed to process a task. An application program can also issue requests for main storage to provide intermediate work areas and any other main storage area not automatically provided by CICS but needed to process a task. Main storage acquired by an application program can be initialized to any bit configuration, for example, binary zeros or EBCDIC blanks.

Main storage associated with a task is controlled and accounted for by CICS. This allows CICS to release all main storage associated with a task upon request or when the task is normally or abnormally terminated. Main storage is accounted for as follows:

- Task control areas (TCAs) are accounted for through pointers in the dispatch control areas (DCAs). The DCAs are chained from the common system area (CSA).
- Task storage is chained off the task control area (TCA).
- Terminal storage is chained off the TCTTE (the TCTTESC field is the origin of the terminal input/output area (TIOA) chain; the TCTTEDA field contains the address of the current TIOA regardless of the position of that TIOA on the chain).
- Program storage is accounted for in the processing program table (PPT).
- Suspended tasks are accounted for by the suspending CICS management program (task control, storage control, or temporary storage control).

If there is insufficient main storage to satisfy a storage acquisition request, TCASCSA is filled with binary zeros. All activity within the task is suspended until sufficient dynamic storage becomes available and its address is placed in TCASCSA, unless the application programmer has specified in his request that control is to be returned to the application program. Lack of storage will cause a short-on-storage condition. The initiation of new tasks is restricted by CICS until the short-on-storage condition is alleviated. Normally, this occurs as a result of some other task releasing storage currently reserved for it. (See "Declare a Task to be Purgeable (TYPE=PURGE)" on page 227, for corrective action that can be taken if the short-on-storage condition continues.)

OBTAIN AND INITIALIZE MAIN STORAGE (TYPE=GETMAIN)

The format of the DFHSC macro to get main storage and initialize the area obtained, if required, is as follows:

```
DFHSC TYPE=GETMAIN
      [,INITIMG={number|YES}]
      [,NUMBYTE=number]
      [,COND={YES|(YES,symb-addr)
         |(NO,symb-addr)}]
      [,CLASS={TERMINAL|USER|
         TRANSDATA|TEMPSTRG}]
```

This macro is used to get main storage of a specified size and class and, optionally, to initialize that storage to a specified bit configuration. The address of the storage area obtained is placed in TCASCSA on a doubleword boundary by CICS. TERMINAL, TRANSDATA, and TEMPSTRG can be abbreviated to TERM, TD, and TS respectively.

When using this macro, the application programmer should:

- Check whether any existing storage that is no longer required by the task should be released, to avoid causing a short-on-storage condition to occur, or if it may be left for CICS to release when the task is terminated.
- Specify the class of storage required using the CLASS operand.
- Calculate the number of bytes required and either specify that amount in the NUMBYTE operand, or place it in TCASCNB, in binary form, before issuing the DFHSC macro. A zero data length is not allowed for a DFHSC TYPE=GETMAIN macro.
- Specify the COND operand if control is to be returned to the application program, irrespective of whether the

```
ASM:

       MVI    TCASCIB,B'0'        INITIALIZE WITH BINARY ZEROS
       MVC    TCASCNB,=H'1024'    SIZE OF STORAGE REQUESTED
              .
              .
       DFHSC  TYPE=GETMAIN,       OBTAIN NEW STORAGE AREA              *
              INITIMG=YES,        INITIALIZE WITH BINARY ZEROS         *
              COND=YES,           RETURN CONTROL                       *
              CLASS=TERMINAL      CLASS OF STORAGE REQUESTED
       CLC    TCASCSA,=F'0'       WAS STORAGE AVAILABLE?
       BE     NOSTRG              BRANCH IF NOT
       L      TIOABAR,TCASCSA     LOAD REGISTER IF STORAGE FOUND

COBOL:

       MOVE ' ' TO TCASCIB.       NOTE INITIALIZE WITH BLANKS
       MOVE 1024 TO TCASCNB.      NOTE SIZE OF STORAGE REQUESTED.
              .
              .
       DFHSC  TYPE=GETMAIN,       OBTAIN NEW STORAGE AREA              *
              INITIMG=YES,        INITIALIZE WITH BLANKS               *
              COND=YES,           RETURN CONTROL                       *
              CLASS=TERMINAL      CLASS OF STORAGE REQUESTED
       IF TCASCSA EQUAL 0 GO TO NOSTRG.
       MOVE TCASCSA TO TIOABAR.

PL/I:

       TCASCIB=0;                 /*INITIALIZE WITH BINARY ZEROS*/
       TCASCNB=1024;              /*SIZE OF STORAGE REQUESTED*/
            .
            .
       DFHSC  TYPE=GETMAIN,       OBTAIN NEW STORAGE AREA              *
              INITIMG=YES,        INITIALIZE WITH BINARY ZEROS         *
              COND=YES,           RETURN CONTROL                       *
              CLASS=TERMINAL      CLASS OF STORAGE REQUESTED
       IF UNSPEC(TCASCSA) = 0 THEN GO TO NOSTRG;
       TIOABAR=TCASCSA;           /*LOAD REGISTER IF STORAGE FOUND*/
```

requested storage has been acquired or not.

- Specify a symbolic base address for the storage area.
- Move the storage address located at TCASCSA to the symbolic base address. (This address points to the storage accounting area of the storage area.)
- Copy the symbolic storage definition for the appropriate input/output area or storage accounting area **prior** to the symbolic definition of the user's program storage area.

The following example shows how to request a 1024-byte area of main storage and initialize it with zeros:

```
DFHSC TYPE=GETMAIN,INITIMG=00,
      NUMBYTE=1024,CLASS=TERMINAL
```

The above examples show how to specify the size of a required storage area and the value to which it is to be initialized and then request that the storage be acquired.

RELEASE MAIN STORAGE (TYPE=FREEMAIN)

The format of the DFHSC macro to release main storage is as follows:

```
DFHSC TYPE=FREEMAIN
      [,RELEASE=ALL]
```

If the task itself does not release acquired storage, the storage is released by CICS upon termination of the task.

When this macro is used to release a single storage area, the address of that area must be placed in TCASCSA **prior** to execution of the macro. If all terminal storage acquired by means of DFHSC TYPE=GETMAIN,CLASS=TERMINAL macros in the application program or by CICS on behalf of the task is to be released, the RELEASE=ALL operand will achieve that result; in this case, it is not necessary to place an address in TCASCSA.

The following example shows how to release all main storage currently allocated to a terminal:

```
  DFHSC TYPE=FREEMAIN,RELEASE=ALL
```

The use of the RELEASE=ALL operand is restricted during basic mapping support (BMS) output operations having "OUT" disposition, to preserve the terminal storage used by BMS. Once a DFHBMS macro with "OUT" disposition has been issued, the application program must not issue a DFHSC TYPE=FREEMAIN,RELEASE=ALL macro until either a DFHBMS TYPE=PAGEOUT or DFHBMS TYPE=PURGE macro has been issued.

The use of the RELEASE=ALL operand is also restricted during data interchange output operations (ADD, ERASE, REPLACE, NOTE, QUERY, END, and ABORT) to preserve the terminal storage used by the data interchange program (DFHDIP). Once a destination has been selected, RELEASE=ALL must not be specified until TYPE=END, TYPE=QUERY, or TYPE=ABORT has been specified in the DFHDI macro for that destination.

The following example shows, in assembler language, how to release a single main storage area, placing the address of the area to be released in TCASCSA before issuing the release request.

```
  ST    TIOABAR,TCASCSA
              .
              .
  DFHSC TYPE=FREEMAIN
```

The equivalent instruction in COBOL is:

```
  MOVE TIOABAR TO TCASCSA.
```

The equivalent instruction in PL/I is:

```
  TCASCSA=TIOABAR;
```

OPERANDS OF DFHSC MACRO

CLASS=
specifies the class of the storage to be acquired, as follows:

TERMINAL or TERM
specifies that the storage area is to be used as a terminal input/output area (TIOA), which is chained to the terminal control table terminal entry (TCTTE). All requests for storage related to terminal input/output must specify this class.

If storage other than TERMINAL class is used as a TIOA for subsequent terminal control input/output operations, storage violations may occur.

USER
indicates that the storage area is to be associated with the application program and used by that program. This area chained to the TCA associated with the requesting task.

TRANSDATA or TD
specifies that the storage area is to be used for transient data record storage (a TDIA or TDOA). This area is chained to the TCA associated with the requesting task and is used by transient data control.

TEMPSTRG or TS
specifies that the storage area is to be used as a temporary storage input/output area (TSIOA). This area is chained to the TCA associated with the requesting task and is used by temporary storage control.

Note: USER, TRANSDATA, and TEMPSTRG specifications have essentially the same effect. The advantage of using CLASS=TRANSDATA or CLASS=TEMPSTRG when either is appropriate is that the specification serves as documentation both in the program and in the class code of the storage accounting field for the area.

COND=
specifies that control is to be returned to the application program, whether or not the requested storage area is acquired.

YES
specifies that control is to be given to the instruction immediately following the expansion for the DFHSC TYPE=GETMAIN macro in the application program. To determine whether the requested storage area is acquired, the application program must examine TCASCSA, which is set to binary zeros if the request cannot be satisfied.

(YES,symb-addr)
causes a branch to the location specified by the symbolic address if the requested storage is acquired; otherwise, control is returned

to the instruction immediately following the expansion for the DFHSC TYPE=GETMAIN macro in the application program.

(NO,symb-addr)
causes a branch to the location specified by the symbolic address if the requested storage is not acquired; otherwise, control is returned to the instruction immediately following the expansion for this macro in the application program.

INITIMG=
specifies that the acquired storage area is to be initialized to the desired bit configuration.

number
is a two-digit hexadecimal numeral indicating the bit configuration desired.

YES
specifies that the desired bit configuration is in TCASCIB.

NUMBYTE=number
is a decimal numeral up to 65520 specifying the size, in bytes, of the storage area being requested; if omitted, the number of bytes is assumed to be stored in binary form in TCASCNB. A zero data length is not allowed for a DFHSC TYPE=GETMAIN macro. In BMS mapping operations, the number of bytes can be specified as an assembler language expression, for example:

```
NUMBYTE=mapname.E-TIOADBA
```

Note: Depending upon the class of storage specified (see the CLASS operand), CICS storage management automatically increments the amount of storage requested to allow for the storage accounting field and other control information. For CLASS=USER and CLASS=TERMINAL (TIOA) storage, the exact number of bytes required should be specified. For CLASS=TRANSDATA (TDIA and TDOA) and CLASS=TEMPSTRG (TSIOA) storage, the amount requested must include four additional bytes to allow for a portion of CICS control information, namely, the length (LLbb) field at the beginning of the area. (See also "Storage Defined During Initialization" on page 35 that apply when programming in COBOL.)

RELEASE=ALL
specifies that all main storage acquired by means of DFHSC TYPE=GETMAIN,CLASS=TERMINAL macros is to be released.

The use of the RELEASE=ALL operand is restricted during basic mapping support (BMS) output operations that have an OUT disposition; this restriction preserves the terminal storage used by BMS. Once a DFHBMS macro with an OUT disposition has been issued, the application program must not issue a DFHSC TYPE=FREEMAIN,RELEASE=ALL macro until either a DFHBMS TYPE=PAGEOUT or DFHBMS TYPE=PURGE macro has been issued.

If this operand is not specified, only one storage area can be released by a DFHSC TYPE=FREEMAIN macro; the address of that area must be in TCASCSA and must be the main storage address returned as a result of a previously issued DFHSC TYPE=GETMAIN macro.

CHAPTER 5.6. TRANSIENT DATA CONTROL (DFHTD MACRO)

Transient data management provides, through transient data control, a generalized queuing facility. Data can be queued (stored) for subsequent internal or external processing. Selected units of information, as specified by the application programmer, can be routed to or from predefined symbolic destinations, either intrapartition or extrapartition. The definitions for the destinations must be contained in a destination control table (DCT) established by the system programmer at system generation. See the appropriate CICS Resource Definition manual.

Intrapartition destinations are queues of data on direct access storage devices developed for input to one or more programs running asynchronously (concurrently) as separate tasks; they are internal to the CICS partition/region. Data directed to or from these internal destinations is called intrapartition data and must consist of variable-length records. Intrapartition destinations can be associated with either a terminal or an output data set. Intrapartition data may be ultimately transmitted upon request to a destination terminal or retrieved sequentially from the output data set. Typical uses of this facility involve message switching, broadcasting, data base access, routing of output to multiple 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).

An intrapartition queue is reusable. The system programmer can indicate, by symbolic destination, whether (1) transient data space management is to control the reuse of tracks associated with a particular destination identification (DESTID), or (2) the releasing of track space is to be controlled through use of the transient data PURGE macro. If transient data space management is not used, an intrapartition queue continues to grow, irrespective of whether the data has been read, until the application programmer purges it.

Extrapartition destinations are queues (data sets) external to the CICS partition/region, residing on any sequential device (DASD, tape, printer, and so on). In general, sequential extrapartition destinations are used for storing data external to the CICS partition/region or for retrieving data from outside the partition/region. 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/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 is intended for subsequent batched input to non-CICS 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- or variable-length, blocked or unblocked. The record format for a particular extrapartition destination must be described by the system programmer when setting up the DCT.

Intrapartition and extrapartition destinations can be used as indirect destinations, which are symbolic references to still 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 need be 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.

For intrapartition destinations, CICS provides the option of automatic task initiation (ATI). A basis for ATI is established by the system programmer by specifying a nonzero trigger level for a particular intrapartition destination in the DCT. When the number of entries (PUTs from one or more programs) in the queue (destination) reaches the specified level, a transaction 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 GETs to deplete the queue.

Once the queue has been depleted, a new ATI 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 QUEZERO response is returned in response to a DFHTD TYPE=GET macro). 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 QUEZERO condition rather than for some application-dependent factor such as an anticipated number of records. It is the QUEZERO condition only that indicates a depleted queue.

Requests for transient data services are communicated to transient data control through CICS macros. Transient data control then executes as a service program under control of the TCA of the requesting program.

It runs at the priority of the requesting program and saves and restores registers from its TCA. After the requested transient data service has been provided (or attempted), control is returned to the next executable instruction in the requesting program.

The transient data management macro (DFHTD) is used to request any of the following services:

1. Direct data to a predefined symbolic destination which references a data set or a terminal

2. Acquire data from a predefined symbolic source which references a data set or a terminal

3. Control the processing of an extrapartition data set

4. Purge data associated with an intrapartition data set

5. Check the response to a request for transient data services.

The application programmer must specify the parameters required when requesting transient data services. Parameters can be specified in two ways: (1) by including the parameters in operands of the DFHTD macro by which the service is requested, or (2) by coding instructions that move the required parameters to fields of the TCA prior to issuing the DFHTD macro. The latter approach provides some degree of flexibility in that a single DFHTD macro can be tailored according to current logic needs within the application program.

The application programmer can check the CICS response as described in "Test Response to a Request for TD Services (TYPE=CHECK)" on page 249. The operands that can be specified in DFHTD macros are explained in detail at the end of the chapter.

CICS routes a variety of messages generated by CICS programs or tasks to transient data control. For example, terminal control detects a line or terminal problem (not related to a user-provided task) and routes control to the CICS terminal abnormal condition program (DFHTACP). DFHTACP then generates a message to the control system terminal log (CSTL) and/or to the control system master terminal.

Destination definitions for all user and CICS destinations must be included in the DCT. Lack of a destination definition leads to an IDERROR (identification error) response to a | DFHTD macro.

DISPOSE OF DATA (TYPE=PUT)

The format of the DFHTD macro to direct transient data to a predefined symbolic destination is as follows:

```
DFHTD TYPE=PUT
      [,DESTID=symb-name]
      [,TDADDR=symb-addr]
      [,NORESP=symb-addr]
      [,IDERROR=symb-addr]
      [,IOERROR=symb-addr]
      [,NOTOPEN=symb-addr]
      [,NOSPACE=symb-addr]
```

Destinations are intrapartition if associated with a facility allocated to the CICS partition/region and extrapartition if the data is directed to some destination that is external to the CICS partition/region. If intrapartition data is to be placed in the transient data output area, the symbolic storage definition for this area (DFHTDOA) should be copied in the application program. The first four bytes of this definition are a length field. All references to the output area should be made through the use of a register (TDOABAR) which points to the beginning of the area.

The address of the output area containing the data to be written, must either be specified in the TDADDR operand or placed in TCATDAA prior to issuing the macro. For variable-length records or intrapartition data, the first four bytes of the output area must contain the length of the record. For fixed length records, the start of the output area must be the start of the data. The format of the length field is LLbb, where LL is a two-byte binary length (the value of which includes the length of the data plus the four bytes for the length field) and bb should be two bytes containing binary zeros. Transient data control does not release this area after the data is written as output.

If the destination is extrapartition, TYPEFLE=OUTPUT must be specified in the appropriate DFHDCT TYPE=SDSCI system macro, otherwise unpredictable results or an abnormal termination will occur.

The following examples show how to write data to a predefined symbolic destination, in this case, the control system message log (CSML). The address of TDOAVRL, the 2-byte length field at the beginning of the transient data output area (TDOA), is a pointer to the start of the variable-length data to be written.

ASM:

```
TDOABAR   EQU     7
          COPY    DFHTDOA
DATA      DS      CL10
          .
          .
          DFHSC   TYPE=GETMAIN,
                  CLASS=TRANSDATA,
                  INITIMG=00,
                  NUMBYTE=14
          L       TDOABAR,TCASCSA
          MVC     TDOAVRL,LENGTH
          MVC     DATA,MESSAGE
          MVC     TCATDDI,=C'CSML'
          DFHTD TYPE=PUT,
                TDADDR=TDOAVRL
          .
          .
```

COBOL:

```
       02 TDOABAR PIC S9(8) COMP.
          .
          .
    01 DFHTDOA COPY DFHTDOA.
       02 SDATA PIC X(10).
          .
          .
          DFHSC TYPE=GETMAIN,
                CLASS=TRANSDATA,
                INITIMG=00,
                NUMBYTE=14
          MOVE TCASCSA TO TDOABAR.
          MOVE SLENGTH TO TDOAVRL.
          MOVE SMESSAGE TO SDATA.
          MOVE 'CSML' TO TCATDDI.
          DFHTD TYPE=PUT,
                TDADDR=TDOAVRL
          .
          .
```

PL/I:

```
%INCLUDE DFHTDOA;
          2  DATA CHAR(10);
                .
                .
             DFHSC TYPE=GETMAIN,
                   CLASS=TRANSDATA,
                   INITIMG=00,
                   NUMBYTE=14
             TDOABAR=TCASCSA;
             TDOAVRL=LENGTH;
             DATA=MESSAGE;
             TCATDDI='CSML';
             DFHTD TYPE=PUT,
                   TDADDR=TDOAVRL
```

ACQUIRE QUEUED DATA (TYPE=GET)

The format of the DFHTD macro to retrieve queued data from an extrapartition or intrapartition destination is shown below. The address of the retrieved data is returned at TCATDAA.

```
DFHTD TYPE=GET
      [,DESTID=symb-name]
      [,QUEBUSY=symb-addr]
      [,NORESP=symb-addr]
      [,QUEZERO=symb-addr]
      [,IDERROR=symb-addr]
      [,IOERROR=symb-addr]
      [,NOTOPEN=symb-addr]
```

If the data is extrapartition, TCATDAA points to the first word of the data area. For variable-length records, the first four bytes of this area contain the length (LLbb) as specified for variable-length data sets. TYPEFLE=INPUT or TYPEFLE=RDBACK must be specified in the appropriate DFHDCT TYPE=SDSCI system macro, and DESTID must not indicate a system spool file, otherwise unpredictable results or an abnormal termination will occur.

If the data is intrapartition, the symbolic storage definition for the transient data input area (DFHTDIA) must have been copied in the application program. TCATDAA points to a CICS input area defined by DFHTDIA. TDIAIRL contains the length (data length plus the length of the length field) of the area.

Transient data (either intrapartition or extrapartition) must be moved from the input area before it can be used in any other input/output operation.

If the application programmer issues a DFHTD TYPE=GET macro, the input area acquired for the previous GET is reused if it is long enough to contain the input record. If it is not, CICS acquires a new input area of sufficient length and releases the input area previously used. If the application programmer issues a DFHTD TYPE=PUT macro, the input area acquired for a previous GET may also be changed or released. The application programmer should always move data to be saved from the input area to a user area to ensure that it is not overlaid with new data. Addressability to the area should also be reestablished following each GET.

The application programmer should not attempt to free storage acquired by the transient data control program in response to a DFHTD TYPE=GET macro. This storage is freed by CICS in the case of intrapartition data, or by the operating system in the case of extrapartition data. An attempt to free storage acquired for extrapartition data may result in an abnormal termination of CICS, since the storage area address returned by transient data control points to storage that is not part of the CICS dynamic storage subpool.

The following examples show how to read a variable-length record from an intrapartition data set specified prior to issuing the DFHTD TYPE=GET macro. In these examples, the data set is the control system message log (CSML).

ASM:
```
TDIABAR   EQU     7
          COPY    DFHTDIA
          .
          .
          .
          MVC     TCATDDI,=C'CSML'
          DFHTD   TYPE=GET
          L       TDIABAR,TCATDAA
```

COBOL:
```
          02  TDIABAR PIC S9(8) COMP.
          .
          .
       01  DFHTDIA COPY DFHTDIA.
          .
          .
       MOVE 'CSML' TO TCATDDI.
       DFHTD TYPE=GET
       MOVE TCATDAA TO TDIABAR.
          .
          .
```

PL/I:
```
       %INCLUDE DFHTDIA;
          2   DUMMY CHAR(1);
          .
          .
          .
       TCATDDI='CSML';
       DFHTD TYPE=GET
       TDIABAR=TCATDAA;
          .
          .
```

Assume that, in the above examples, the variable-length record is read from an extrapartition data set. The address placed at TCATDAA by CICS is the address of the length (LLbb) field that precedes the actual data. Since the DFHTDIA symbolic storage definition is being used, the address must be adjusted to point to the CICS system section preceding the actual data. Therefore, an instruction to adjust the address should be inserted immediately following the instruction that moves the contents of TCATDAA to TDIABAR. The following examples apply to CICS/OS/VS but are applicable to CICS/DOS/VS if '36' is replaced by '8'.

```
ASM: SH TDIABAR,=H'36'
          .
          .

COBOL: SUBTRACT 36 FROM TDIABAR.
          .
          .

PL/I:  DCL TDIABAA FIXED BIN(31)
       BASED(TDIABAB);
       TDIABAB=ADDR(TDIABAR);
       /* OVERLAY POINTER */
       TDIABAA=TDIABAA-36
       /* DO POINTER ARITHMETIC */
          .
          .
```

Since these examples deal with variable-length records, the first byte of the data is assumed to be the length field (LLbb). If the examples dealt with fixed-length records, appropriate values would be 40 and 12 for CICS/OS/VS and CICS/DOS/VS, respectively.

Note: These values are subject to change in future versions of CICS, because this DSECT is intended only for intrapartition data sets. No DSECT is provided for extrapartition data. Each user should define the extrapartition DSECT so as not to use the absolute values in the above example.

FORCE END OF VOLUME ON AN EXTRAPARTITION DATA SET (TYPE=FEOV)

The format of the DFHTD macro to create a "forced end of volume" situation on an extrapartition magnetic tape data set is as follows:

```
DFHTD TYPE=FEOV
      [,DESTID=symb name]
      [,NORESP=symb-addr]
      [,IDERROR=symb-addr]
      [,NOTOPEN=symb-addr]
```

This macro specifies that a magnetic tape reel is to be rewound and unloaded; output labels are to be created as required and new input labels verified according to host operating system forced-end-of-volume processing. CICS operation is halted, and the next tape reel must be loaded before CICS operation is resumed.

Note: This facility should be used with caution, since CICS operation is halted until the new tape reel has been loaded.

The following examples show how to create a "forced end of volume" situation on an extrapartition magnetic tape data set.

ASM:
```
MVC   TCATDDI,=C'CSML'
DFHTD TYPE=FEOV
  .
  .
```

COBOL:
```
MOVE 'CSML' TO TCATDDI.
DFHTD TYPE=FEOV
  .
  .
```

PL/I:
```
TCATDDI='CSML';
DFHTD TYPE=FEOV
  .
  .
```

PURGE INTRAPARTITION DATA (TYPE=PURGE)

The format of the DFHTD macro to purge all data associated with a particular intrapartition destination (queue) is as follows:

```
DFHTD TYPE=PURGE
      [,DESTID=symb-name]
      [,NORESP=symb-addr]
      [,IDERROR=symb-addr]
```

When transient data associated with a particular intrapartition destination (queue) is no longer needed, the application programmer can purge the data associated with that destination by issuing this macro, which causes all storage associated with the destination to be freed (deallocated).

This macro must be used to free storage associated with a destination designated as nonreusable in the destination control table. Otherwise, the storage remains allocated to the destination; the data and amount of storage associated with the destination continue to grow whenever a DFHTD TYPE=PUT macro refers to the destination.

TEST RESPONSE TO A REQUEST FOR TD SERVICES (TYPE=CHECK)

The format of the DFHTD macro to test the CICS response to a request for transient data services is as follows:

```
DFHTD TYPE=CHECK
      [,NORESP=symb-addr]
      [,QUEZERO=symb-addr]
      [,IDERROR=symb-addr]
      [,IOERROR=symb-addr]
      [,NOTOPEN=symb-addr]
      [,NOSPACE=symb-addr]
```

TRANSIENT DATA RESPONSE CODES

The assembler language or PL/I programmer accesses transient data response codes at TCATDTR; the COBOL programmer accesses these response codes at TCATDRC. In addition, the COBOL programmer can refer to the response codes by means of condition names (for example, TDNORESP or TDQUEZERO). The possible response codes and their meanings are shown below.

If the application programmer does not check for a particular response to a service request, and the exception condition corresponding to that response occurs, program flow proceeds to the next sequential instruction in the application program.

Condition	ASM	COBOL	PL/I
NORESP	X'00'	LOW-VALUES (TDNORESP)	00000000
QUEZERO	X'01'	12-1-9 (TDQUEZERO)	00000001
IDERROR	X'02'	12-2-9 (TDIDERROR)	00000010
IOERROR	X'04'	12-4-9 (TDIOERROR)	00000100
NOTOPEN	X'08'	12-8-9 (TDNOTOPEN)	00001000
NOSPACE	X'10'	12-11-1-8-9 (TDNOSPACE)	00010000

The names enclosed in parentheses in the COBOL column indicate the condition names generated by CICS. These names may be used in testing for the respective conditions in a COBOL program.

The following examples show how to examine the response code provided by CICS and transfer control to the appropriate user-written exception-handling routine.

ASM:
```
DFHTD TYPE=GET,DESTID=CSML
CLI   TCATDTR,X'00'
BE    GOOD
```

```
       DFHPC TYPE=ABEND,ABCODE=GETE
GOOD   DS    0H
       .
       .
```

COBOL:

```
       DFHTD TYPE=GET,DESTID=CSML
       IF TCATDRC = ' ' THEN GO TO GOOD.
       DFHPC TYPE=ABEND,ABCODE=GETE
GOOD.
       .
       .
```

Alternatively, the COBOL programmer may test responses by using the CICS generated condition names.

```
       IF TDNORESP THEN GO TO GOOD.
       .
       .
```

PL/I:

```
       DFHTD TYPE=GET,DESTID=CSML
       IF TCATDTR='0'B THEN GO TO GOOD;
       DFHPC TYPE=ABEND,ABCODE=GETE
GOOD:
       .
       .
```

OPERANDS OF DFHTD MACRO

DESTID=symb-name
specifies the symbolic name of the destination to which the data is to be routed and queued, or from which queued data is to be read. This name must appear in the destination control table (the DCT). If this operand is omitted, the symbolic name of the destination is assumed to be in TCATDDI. For a TYPE=GET macro, DESTID must not indicate a system spool file.

IDERROR=symb-addr
specifies the entry label of the user-written routine to which control is to be passed if the symbolic destination referred to by a DFHTD macro cannot be found.

IOERROR=symb-addr
specifies the entry label of the user-written routine to which control is to be passed if an input/output error occurs on a data record and the data record in error is skipped. Transient data returns IOERROR as long as the queue can be read; a QUEZERO response is returned when the queue cannot be read, in which case, the user may attempt a restart. This condition can also be raised if an attempt is made to write a zero length record to an intrapartition data set. This condition can also be raised under VSAM if the record is too large to fit in a control interval.

NORESP=symb-addr
specifies the entry label of the user-written routine to which control is to be passed if no error occurs during a data set (file) operation. NORESP signifies "normal response."

NOSPACE=symb-addr
specifies the entry label of the user-written routine to which control is to be passed if no more space exists on a queue or if the write request cannot be serviced. If the NOSPACE response is received, no more data should be written to the queue, because it may be lost.

NOTOPEN=symb-addr
specifies the entry label of the user-written routine to which control is to be passed if a destination is closed.

QUEBUSY=symb-addr
specifies the symbolic address of the routine to receive control if the input request attempts to access a record on an input intrapartition queue that has been enqueued upon for output by a PUT or PURGE request. If this operand is omitted, the task issuing the request waits until the queue is no longer being used for output.

QUEZERO=symb-addr
specifies the entry label of the user-written routine to which control is to be passed when the destination (queue) accessed by a DFHTD TYPE=GET macro is empty.

TDADDR=symb-addr
specifies the symbolic address of the output area containing data to be written (for intrapartition data and variable-length extrapartition data, the first four bytes of this area must contain the length of the record). If this operand is omitted, the address of the output area is assumed to be in TCATDAA.

CHAPTER 5.7. TEMPORARY STORAGE CONTROL (DFHTS MACRO)

Temporary storage control enables user-written application programs to store temporary data in main storage or in auxiliary storage on a direct access storage device.

Temporary data is stored, retrieved, and released using a symbolic name (up to eight characters) assigned to the data by the originating task. (The symbolic name must not consist solely of binary zeros.)

The data may be a single record or records retrieved from or added to a temporary storage message set. The former provides a typical "scratch pad" capability. The latter is designed primarily for terminal paging. It is used in conjunction with basic mapping support (see "Chapter 4.3. Basic Mapping Support" on page 143) and page supervision programs to achieve random access to general-purpose storage files. In general, the paging facility of temporary storage should be used **only** when multiple records are involved and direct access to those records is necessary. This queuing of message sets should not be used for sequential data. Transient data management provides facilities for efficient handling of sequential data sets. If data contained in a message set is to be updated and retrieved by multiple tasks, it may be necessary to protect it by means of the task control enqueuing facility.

Data placed in temporary storage can remain intact beyond the time that the originating task is active in the system. That is even after the originating task is terminated and its transaction storage released, data placed in temporary storage can be accessed by other tasks through references to the symbolic name under which it is stored. Temporary data remains intact until released by the originating task or by any other task. Prior to release, it can be accessed any number of times.

When temporary data is released, the space that it occupied is reusable. If the data is in main storage, the storage area becomes part of available dynamic storage. If the data is in auxiliary storage, the physical space that the data occupied becomes available and can be reused for other data.

Temporary data can be retrieved by the originating task or by any other task using the symbolic name assigned to it. The name assigned to a single record should be unique. If more than one record has the same name, the record is queued in temporary storage. If an attempt is made to retrieve a record from the queue, the records will be presented on a first in first out basis. All information moved to or from a temporary storage message set is referred to by a unique name assigned to the message set. Specific entries (logical records) within a message set are referred to by relative position numbers. To avoid conflicts caused by duplicate names, a naming convention should be established and followed by all programmers. For example, the operator identification, terminal identification, or transaction identification could be appended as a prefix or suffix to each programmer-supplied symbolic name.

Temporary data can be stored in either main or auxiliary storage. Generally, main storage should be used if the data is needed for only short periods of time; auxiliary storage should be used if the data must be kept for extended periods of time. Another consideration is that data stored on auxiliary storage is maintained after CICS termination and can be recovered in a subsequent restart. No attempt is made to recover data in main storage. 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 that they be quasi-reenterable.

Some uses of the page queuing facility follow:

1. Terminal paging. A task could retrieve a large master record from a direct access data set, format it into several screen images, 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 coding (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 the like. This facility is provided by CICS Basic Mapping Support as described in "Chapter 4.3. Basic Mapping Support" on page 143.

2. A suspend data set. Assume a data collection task is in progress on a certain terminal. The task reads in one or more units of input and then allows the terminal operator to interrupt the process. If no interruption occurs (some kind of coded input), the task repeats the data collection process. If the

operator interrupts the data collection stream with coded input, the data collection task writes its "incomplete" data to temporary storage and terminates the task. The terminal is now free for entry of a different transaction (perhaps a high-priority inquiry). When the terminal is available to continue the data collection operation, 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.

3. An application that accepts input data to be written as output on a preprinted form.

The DFHTS macro is used to:

- Acquire data from main or auxiliary storage
- Send data to main or auxiliary storage
- Update data in main or auxiliary storage
- Release temporary data in main or auxiliary storage
- Check the response to a request for temporary storage services.

Parameters can be specified in either of two ways:

- By including the parameters in operands of the DFHTS macro by which temporary storage services are requested, or
- By coding instructions that place the parameter values in fields of the TCA prior to issuing the DFHTS macro.

The second of these approaches provides flexibility in that the parameters of a single DFHTS macro can vary to meet the logic needs of the application program.

The CICS response to a request for temporary storage services can be checked, as explained under "Test Response to a Request for Temporary Storage Services," later in this chapter. If the programmer does not check for a particular response, and the condition corresponding to that response occurs, program flow proceeds to the next sequential instruction in the application program. All operands that can be included in the DFHTS macro are discussed at the end of the chapter.

STORE TEMPORARY DATA AS A SINGLE UNIT OF INFORMATION (TYPE=PUT)

The format of the DFHTS macro to store a single unit of information as temporary data in main or auxiliary storage (that is, as though using a "scratch pad") is as follows:

```
DFHTS TYPE=PUT
      [,TYPOPER=REPLACE]
      [,DATAID=name]
      [,TSDADDR={symb-addr|YES}]
      [,STORFAC={AUXILIARY|MAIN}]
      [,COND=YES]
      [,NOSPACE=symb-addr]
      [,NORESP=symb-addr]
      [,IOERROR=symb-addr]
      [,INVREQ=symb-addr]
      [,ERROR=symb-addr]
```

This macro causes data to be written to temporary storage as a single unit of information (logical record).

Temporary data may be written from a temporary storage input/output area (TSIOA) or from a main storage area identified by the application programmer. It must have the standard variable-length format, with the data length specified in the first four bytes. These bytes should contain LLbb, where LL is a two-byte binary length field (the value of which includes the length of the data plus the four bytes for the length field) and bb is a two-byte field of binary zeros. The maximum temporary storage record size is based on user-specified data set characteristics. (See temporary storage in the appropriate <u>CICS Installation and Operations Guide</u>.)

Existing temporary storage data can be updated by adding the TYPOPER=REPLACE operand. This causes the current data identified by the DATAID operand to be released and replaced with the data provided. If the data cannot be found, the TYPOPER=REPLACE operand is ignored.

The following examples show how to write a single record of information to temporary storage.

ASM:

```
TSIOABAR   EQU    7
           COPY   DFHTSIOA
DATA       DS     CL11
           .
           .
           DFHSC  TYPE=GETMAIN,
                  CLASS=TEMPSTRG,
                  NUMBYTE=15
           L      TSIOABAR,TCASCSA
           MVC    TSIOAVRL,LENGTH
           MVC    DATA,MESSAGE
```

```
          DFHTS TYPE=PUT,
                DATAID=UNIQNME,
                TSDADDR=TSIOAVRL
LENGTH    DC    AL2(L'MESSAGE+4)
MESSAGE   DC    C'HELLO THERE'
```

COBOL:

```
  WORKING-STORAGE SECTION.
  77 SMESSAGE PIC X(11) VALUE
       'HELLO THERE'.
  77 SLENGTH PIC 9(8) COMP VALUE 15
  LINKAGE SECTION.
      .
  02 TSIOABAR PIC S9(8) COMP.
      .
      .
  01  DFHTSIOA COPY DFHTSIOA.
      02 SDATA PIC X(11).
      .
      .
  DFHSC TYPE=GETMAIN,CLASS=TEMPSTRG,
        NUMBYTE=15
      MOVE TCASCSA TO TSIOABAR.
      MOVE SLENGTH TO TSIOAVRL.
      MOVE SMESSAGE TO SDATA.
  DFHTS TYPE=PUT,DATAID=UNIQNME,
        TSDADDR=TSIOAVRL
```

PL/I:

```
  %INCLUDE DFHTSIOA;
      2 DATA CHAR(11);
      .
      .
  DFHSC TYPE=GETMAIN,CLASS=TEMPSTRG,
        NUMBYTE=15
  TSIOABAR=TCASCSA;
  TSIOAVRL=LENGTH;
  DATA=MESSAGE;
  DFHTS TYPE=PUT,DATAID=UNIQNME,
        TSDADDR=TSIOAVRL
  DCL MESSAGE CHAR(11) INIT
        ('HELLO THERE');
  DCL LENGTH FIXED BIN(15) INIT(15);
```

STORE DATA TO A TEMPORARY STORAGE MESSAGE SET (TYPE=PUTQ)

The format of the DFHTS macro to cause an entry to be written to a temporary storage message set is as follows:

```
DFHTS TYPE=PUTQ
      [,TYPOPER=REPLACE]
      [,DATAID=name]
      [,TSDADDR={symb-addr|YES}]
      [,STORFAC={AUXILIARY|MAIN}]
      [,ENTRY={n|YES}]
      [,COND=YES]
      [,NOSPACE=symb-addr]
      [,NORESP=symb-addr]
      [,IOERROR=symb-addr]
      [,INVREQ=symb-addr]
      [,ENERROR=symb-addr]
      [,ERROR=symb-addr]
```

This macro causes a unit of information to be written to a message set, or queue, in temporary storage. The unit is written in a relative position that is one beyond the last entry written to the message set. Following a PUTQ request, the relative record number is returned to the user in TCATSRN, a two-byte field.

Temporary data may be written from a temporary storage input/output area (TSIOA) or from a main storage area identified by the application programmer. It must have the standard variable-length format, with the data length specified in the first four bytes. These bytes should contain LLbb, where LL is a two-byte binary length field (the value of which includes the length of the data plus the four bytes for the length field) and bb is a two-byte field of binary zeros. The maximum temporary storage record size is based on user-specified data set characteristics. (See temporary storage in the appropriate CICS Installation and Operations Guide.)

Existing temporary storage data can be updated by specifying the TYPOPER=REPLACE and ENTRY operands. The specified record within the message set is released and replaced with the data provided. If the queue does not exist, TYPOPER=REPLACE and ENTRY are ignored. If the queue exists, but the specified entry does not, ENERROR is returned.

RETRIEVE A SINGLE UNIT OF TEMPORARY DATA (TYPE=GET)

The format of the DFHTS macro to retrieve a single unit of temporary data is as follows:

```
DFHTS TYPE=GET
      [,STORCLS={TERMINAL|TERM|
       TEMPSTRG|TS}]
      [,DATAID=name]
      [,TSDADDR={symb-addr|YES}]
      [,RELEASE={YES|NO}]
      [,NORESP=symb-addr]
      [,IDERROR=symb-addr]
      [,IOERROR=symb-addr]
      [,INVREQ=symb-addr]
      [,ERROR=symb-addr]
```

This macro causes a single record to be retrieved from temporary storage. A record stored in temporary storage by a DFHTS TYPE=PUT macro can be retrieved only by this macro. A record, once retrieved, can be released by the RELEASE=YES operand. If RELEASE=NO is specified, or is assumed by default, the record is retained until released by another task or when CICS is terminated.

The STORCLS and TSDADDR operands are mutually exclusive.

If TSDADDR is specified, the record, including its length field (LLbb), is placed either in storage at the symbolic address specified, or at the address in TCATSDA if YES is specified.

If STORCLS=TEMPSTRG is specified, the record, including its length field (LLbb), is placed in a temporary storage class storage area whose address is returned in TCATSDA. Before this area can be used as a TSIOA, the application program must reduce the address in TCATSDA by 8 bytes to include the storage accounting area. This makes it addressable by TSIOABAR.

If STORCLS=TERMINAL is specified, the record, including its length field (LLbb), is placed in a terminal class storage area. This area is prefixed by CICS with an 8 byte storage area. The address of the prefixed area is returned in TCATSDA.

If neither STORCLS nor TSADDR is specified, STORCLS=TEMPSTRG is assumed by default and processing is as described above.

The following examples show how to read a single record from temporary storage with the required addressability and adjustments. The examples show the use of the DFHTS TYPE=GET macro with STORCLS=TEMPSTRG assumed by default.

ASM:
```
TSIOABAR    EQU  7
            COPY DFHTSIOA
            .
            .
        DFHTS TYPE=GET,DATAID=UNIQNME
        L TSIOABAR,TCATSDA
        SH TSIOABAR,=H'8'
```

COBOL:
```
            02 TSIOABAR PIC S9(8) COMP.
            .
            .
        01 DFHTSIOA COPY DFHTSIOA.
            .
            .
        DFHTS TYPE=GET,DATAID=UNIQNME
        MOVE TCATSDA TO TSIOABAR.
        SUBTRACT 8 FROM TSIOABAR.
```

PL/I:
```
        %INCLUDE DFHTSIOA;
            2 DATA CHAR(10);
            .
            .
        DFHTS TYPE=GET,DATAID=UNIQNME
        DCL TSIOBAA FIXED BIN(30)
                    BASED(TSIOBAB);
        TSIOABAR=TCATSDA;
        TSIOBAB=ADDR(TSIOABAR);
        TSIOBAA=TSIOBAA-8;
```

The following examples show the use of the DFHTS TYPE=GET macro with STORCLAS=TERMINAL specified explicitly.

ASM:
```
TIOABAR     EQU  7
            COPY DFHTIOA
            .
            .
       DFHTS TYPE=GET,STORCLS=TERM,
             DATAID=UNIQNME
       L TIOABAR,TCATSDA
```

COBOL:
```
            02 TIOABAR PIC S9(8) COMP.
            .
            .
       01 DFHTIOA COPY DFHTIOA
            .
            .
       DFHTS TYPE=GET,STORCLS=TERM,
             DATAID=UNIQNME
       MOVE TCATSDA TO TIOABAR.
```

PL/I:
```
       %INCLUDE DFHTIOA;
           2 DATA CHAR(10);
           .
           .
       DFHTS TYPE=GET,STORCLS=TERM,
             DATAID=UNIQNME
             TIOABAR=TCATSDA;
```

<u>RETRIEVE DATA FROM A TEMPORARY STORAGE MESSAGE SET (TYPE=GETQ)</u>

The format of the DFHTS macro to retrieve a logical record from a temporary storage message set is as follows:

```
DFHTS TYPE=GETQ
      [,STORCLS={TERMINAL|TEMPSTRG}]
      [,DATAID=name]
      [,TSDADDR={symb-addr|YES}]
      [,ENTRY={n|YES}]
      [,NORESP=symb-addr]
      [,IDERROR=symb-addr]
      [,IOERROR=symb-addr]
      [,INVREQ=symb-addr]
      [,ENERROR=symb-addr]
      [,ERROR=symb-addr]
```

This macro causes an entry previously written to a temporary storage message set, or queue, to be retrieved. A record stored in temporary storage by a DFHTS TYPE=PUTQ macro can only be retrieved by a TYPE=GETQ macro. The record to be retrieved from a queue is identified by the ENTRY operand which indicates its relative position within the queue. The position of an entry is determined by its order of creation.

RELEASE A SINGLE UNIT OF TEMPORARY DATA (TYPE=RELEASE)

The format of the DFHTS macro to release a single unit of data placed in temporary storage by means of a DFHTS TYPE=PUT macro is as follows:

```
DFHTS TYPE=RELEASE
      [,DATAID=name]
      [,NORESP=symb-addr]
      [,IDERROR=symb-addr]
      [,INVREQ=symb-addr]
      [,ERROR=symb-addr]
```

This macro causes the main or auxiliary storage area used for a single record of temporary data (created by means of a DFHTS TYPE=PUT macro) to be released.

If temporary data named in a DFHTS TYPE=RELEASE macro is in main storage, the area that it occupies is released and returned to the available dynamic storage area. If the data is in auxiliary storage, the space is made available for reuse.

A single unit of data should be released at the earliest possible time to avoid using excessive amounts of storage for this purpose.

The following examples show how to release a single record from temporary storage.

ASM:

```
MVC   TCATSDI,=C'UNIQNME'
DFHTS TYPE=RELEASE
```

COBOL:

```
MOVE 'UNIQNME' TO TCATSDI.
DFHTS TYPE=RELEASE
```

PL/I:

```
TCATSDI='UNIQNME';
DFHTS TYPE=RELEASE
```

PURGE A TEMPORARY STORAGE MESSAGE SET (TYPE=PURGE)

The format of the DFHTS macro to purge, or free, data saved as a temporary storage message set (that is, in response to DFHTS TYPE=PUTQ macros) is as follows:

```
DFHTS TYPE=PURGE
      [,DATAID=name]
      [,NORESP=symb-addr]
      [,IDERROR=symb-addr]
      [,INVREQ=symb-addr]
      [,ERROR=symb-addr]
```

This macro causes all existing entries in a temporary storage queue (created by means of DFHTS TYPE=PUTQ macros) to be freed. There is no way to free selected records from a temporary storage message set; in particular, a DFHTS TYPE=RELEASE macro cannot be used to free a record that is part of a message set created by means of DFHTS TYPE=PUTQ.

If the temporary data is in main storage, the area that it occupies is freed and returned to the available dynamic storage area. If the data is in auxiliary storage, the space is made available for reuse.

A temporary storage message set should be purged at the earliest possible time to avoid using excessive amounts of storage for this purpose.

TEST RESPONSE TO A REQUEST FOR TEMPORARY STORAGE SERVICES (TYPE=CHECK)

The format of the DFHTS macro to test the CICS response to a request for temporary storage services is as follows:

```
DFHTS TYPE=CHECK
      [,NOSPACE=symb-addr]
      [,NORESP=symb-addr]
      [,IDERROR=symb-addr]
      [,IOERROR=symb-addr]
      [,INVREQ=symb-addr]
      [,ENERROR=symb-addr]
      [,ERROR=symb-addr]
```

TEMPORARY STORAGE RESPONSE CODES

The assembler language or PL/I programmer can access temporary storage response codes at TCATSTR; the COBOL programmer at TCATSRC. In addition the COBOL programmer can refer to the response codes by means of condition names (TSNORESP, TSIDERROR, and so on). The possible response codes and their meanings are shown below. (See also the examples at the end of this discussion.)

Condition	ASM	COBOL	PL/I
NORESP	X'00'	LOW-VALUES (TSNORESP)	00000000
ENERROR	X'01'	12-1-9 (TSENERROR)	00000001
IDERROR	X'02'	12-2-9 (TSIDERROR)	00000010
IOERROR	X'04'	12-4-9 (TSIOERROR)	00000100
NOSPACE	X'08'	12-8-9 (TSNOSPACE)	00001000
INVREQ	X'20'	11-0-1-8-9 (TSINVREQ)	00100000

The names enclosed in parentheses in the COBOL column indicate the condition names generated by CICS. These names may be used in testing for the respective conditions in a COBOL program.

The test for the ERROR response is satisfied by a **not equal** condition; that is, not X'00', not LOW-VALUES, or not 00000000 for ASM, COBOL, and PL/I, respectively.

If the application programmer does not check for a particular response to a service request, and the condition corresponding to that response occurs, program flow proceeds to the next sequential instruction in the application program.

The following examples show how to examine the response code provided by CICS and transfer control to the appropriate user-written exception-handling routine.

ASM:

```
       DFHTS TYPE=GET,DATAID=UNIQNME,
             TSDADDR=YES
       CLI   TCATSTR,X'00'
       BE    GOOD
       DFHPC TYPE=ABEND
GOOD   DS    0H
```

COBOL:

```
       DFHTS TYPE=GET,DATAID=UNIQNME,
             TSDADDR=YES
       IF TCATSRC = ' ' THEN GO TO GOOD.
       DFHPC TYPE=ABEND
GOOD.
```

Alternatively, the COBOL programmer may test responses by using the CICS generated condition names, as follows:

```
          IF TSNORESP THEN GO TO GOOD.
```

PL/I:

```
       DFHTS TYPE=GET,DATAID=UNIQNME,
             TSDADDR=YES
       IF TCATSTR='0'B THEN GO TO GOOD;
       DFHPC TYPE=ABEND
GOOD:
```

OPERANDS OF DFHTS MACRO

COND=YES
specifies that control is to be returned to the application program when the request cannot be satisfied immediately because there is not enough space on the temporary storage data set. If this operand is omitted, the requesting task is suspended when no space is available and is resumed when the space becomes available. Space becomes available as it is released by other tasks in the system.

DATAID=name
specifies the unique alphanumeric name, up to eight characters in length, to be assigned to the temporary data. If this operand is omitted, the name is assumed to be in TCATSDI.

Note: The application program should not construct a DATAID beginning with any of the characters X'FA' through X'FF'. Use of these characters for this purpose is reserved for CICS.

ENERROR=symb-addr
specifies the entry label of the user-written routine to which control is to be passed if the entry number specified or implied is invalid (that is, not within the limits of the message set).

ENTRY=
specifies the number, relative to one, of the logical record to be retrieved from the message set.

n
is a decimal numeral to be taken as the relative number of the logical record to be retrieved. This number may be the number of any entry that has been written to the temporary storage message set.

YES
indicates that the number (in binary) of the logical record to be retrieved is in TCATSRN, a two-byte field.

If this operand is omitted, CICS retrieves (1) the first logical record from the message set, for the first retrieval request, or (2) the next sequential logical record following the last-retrieved record (by any task), for other than the first request. In the latter case, the relative record number is returned in TCATSRN.

ERROR=symb-addr
specifies the entry label of the user-written routine to which control is to be passed if an error occurs and the corresponding specific error routine operand (for example, IDERROR) has not been specified.

IDERROR=symb-addr
specifies the entry label of the user-written routine to which control is to be passed if the symbolic destination identification referred to by a GET, GETQ, RELEASE, or PURGE macro cannot be found in either main storage or auxiliary storage.

INVREQ=symb-addr
specifies the entry label of the user-written routine to which control is to be passed if (1) a PUT or PUTQ request refers to data whose length is equal to zero or greater than the control interval size of the auxiliary data set minus 84 bytes for control information, or (2) the request is otherwise determined to be invalid.

IOERROR=symb-addr
specifies the entry label of the user-written routine to which control is to be passed if an unrecoverable input/output error occurs.

NORESP=symb-addr
specifies the entry label of the user-written routine to which control is to be passed if no errors occur during temporary storage processing. NORESP signifies "normal response."

NOSPACE=symb-addr
specifies the entry label of the user-written routine to which control is to be passed when insufficient space is available on the temporary storage data set to contain the data in a PUT or PUTQ request. The user-written NOSPACE routine is passed control only if COND=YES is also specified in the PUT or PUTQ request.

RELEASE=
specifies the disposition of the temporary data following the move operation.

YES
the data and storage area used for the data are to be released after this operation.

NO
the data is to be retained, available for subsequent similar reference.

STORCLS=
specifies the class of storage to be obtained for the temporary data. This operand is ignored if TSADDR is specified.

TERMINAL or TERM
specifies that the data is to be placed in a TERMINAL class storage area.

TEMPSTRG or TS
specifies that the data is to be placed in a temporary storage area.

If this operand is omitted, TEMPSTRG is assumed.

STORFAC=
specifies the type of storage to be used for the temporary data.

AUXILIARY
indicates that the data is to be placed in auxiliary storage on a direct access storage device.

MAIN
indicates that the data is to be placed in main storage.

If this operand is omitted, AUXILIARY is assumed.

TSDADDR=
specifies the symbolic address of the data portion (including the LLbb field) of the area in which the temporary data is stored.

symb-addr
is the symbolic address of the data portion of the storage area that contains the temporary data.

YES
indicates that the symbolic address of the data portion of the storage area has been placed in TCATSDA by the application programmer.

If this operand is omitted, the appropriate symbolic address is assumed to be in TCATSDA.

TYPOPER=REPLACE
indicates that the specified record within the data set or message set is to be released and replaced with the record or data provided. If the message set does not exist (DATAID cannot be found), the data provided is placed in temporary storage as in a normal PUTQ without TYPOPER=REPLACE specified.

For TYPE=PUTQ, whenever REPLACE is specified the ENTRY operand must also be coded.

PART 6. CICS BUILT-IN FUNCTIONS

CHAPTER 6.1. INTRODUCTION TO CICS BUILT-IN FUNCTIONS

Several commonly used functions are available to the application programmer through CICS macros. These are functions which would otherwise have to be coded as separate subroutines by the programmer. These functions, referred to as built-in functions, provide the following services:

- Table search
- Phonetic conversion
- Verification of a data field
- Editing of a data field
- Bit manipulation
- Input formatting
- Weighted retrieval.

Requests for these services are communicated to the CICS built-in functions program (DFHBFP) through the DFHBIF macro. DFHBFP is then executed, at the priority of the requesting task, under control of the common control communication area (TCACCCA) of the TCA of the requesting task. Normally, control is returned to the next sequential instruction following the macro expansion in the requesting program; however, conditional branch options can be specified in the macro request if desired.

Since DFHBFP uses TCACCCA, the application program must issue the DFHBFTCA macro to copy the symbolic storage definition for this area and store any required information therein before issuing the DFHBIF macro. "Chapter 6.2. Storage Definition for Built-In Functions (DFHBFTCA Macro)" on page 263 explains how to do this.

The formats and operands of the DFHBIF macro are described in "Chapter 6.3. CICS Built-In Functions (DFHBIF Macro)" on page 265.

CHAPTER 6.2. STORAGE DEFINITION FOR BUILT-IN FUNCTIONS (DFHBFTCA MACRO)

When CICS built-in functions (BIFs) are used in an application program, the symbolic storage definition for the TCACCCA used by these built-in functions must be copied into the application program. This copying is achieved by means of the DFHBFTCA macro, which must immediately follow the statement that copies the TCA and the user's definition of a TWA, if any.

The format of the DFHBFTCA macro is as follows:

```
DFHBFTCA [OPTION={BASIC|WTRET}]
         [,COBLANG={LVL1|LVL2}]
```

where:

OPTION indicates which built-in functions are to be used.

BASIC is required if any of the following functions are used: table search, phonetic conversion, field verification, field editing, bit manipulation, or input formatting.

WTRET is required if weighted retrieval is used.

If OPTION is omitted, both BASIC and WTRET are assumed.

COBLANG indicates the language level of a COBOL application program, as follows:

LVL1 specifies COBOL language 1

LVL2 specifies COBOL language 2

If COBLANG is omitted, LVL1 is assumed.

The following examples show the statements needed to copy the symbolic storage definitions referred to by the built-in functions, positioned as required.

ASM:

```
        COPY  DFHTCADS
NAME    DS    CL20              TWA
STREET  DS    CL20              TWA
CITY    DS    CL10              TWA
STATE   DS    CL3               TWA
        DFHBFTCA
```

COBOL:

```
01 DFHTCADS COPY DFHTCADS.
   02 NAME PIC X(20).         NOTE TWA
   02 STREET PIC X(20).       NOTE TWA
   02 CITY PIC X(10).         NOTE TWA
   02 STATE PIC X(3).         NOTE TWA
   DFHBFTCA
```

PL/I:

```
%INCLUDE (DFHTCADS);
  2 NAME CHAR(20),            /*TWA*/
  2 STREET CHAR(20),          /*TWA*/
  2 CITY CHAR(10),            /*TWA*/
  2 STATE CHAR(3);            /*TWA*/
  DFHBFTCA
```

TABLE SEARCH BUILT-IN FUNCTION (TYPE=TSEARCH)

```
DFHBIF TYPE=TSEARCH
       [,ARG=symb-addr]
       [,TARGET=symb-addr]
       [,ATABLE=([sa1][,sa2|YES}]
         [,n1][,{n2|YES}][,n3])]
       [,FTABLE=([{sa1|YES}]
         [,{sa2|YES}][,{n1|YES}]
               [,{n2|YES}])]
       [,ORDER={ASCENDING|
               DESCENDING}]
       [,SUBST={sa|'lit value'}]
       [,NOMATCH=symb-addr]
       [,INDEX=symb-addr]
       [,RANGE=YES]
       [,ERROR=symb-addr]
```

This macro specifies that a table is to be searched for a given entry, causing a corresponding value within that table or a second table, the address of the corresponding value, and the index of the selected entry (relative to one) to be returned.

The search argument is compared on a byte-for-byte basis with a specified field of entries in the table being searched. Optionally, a default value can be returned instead of a corresponding value if the searched-for entry is not found. If an index is requested, but the entry is not found, an index value of zero is returned.

Note: In the syntax display, "symb-addr" and "numeric value" have, in some cases, been shortened to "sa" and "n" respectively, and "literal" has been shortened to "lit".

Returned Values

An entry in the argument table that matches the search argument satisfies the table search built-in function. If such an entry is found, the address of the corresponding entry in the function table is returned in TCATSA5, a fullword field.

If TARGET is specified, the function value is returned in the location identified by that operand. If the function table contains more than one matching entry, the address (and the function value, if requested) of the first matching entry encountered during the search is returned.

If ORDER is specified, a binary search is performed, and the address returned is that of the first matching entry found.

If ORDER is omitted, a sequential search is performed, starting at the last entry in the table, and the address returned is that of the last matching entry in the table. The index of the matching entry is returned in TCATSH4 and in the field identified by the INDEX operand if specified.

If RANGE=YES is specified, a matching entry satisfies the search as described above. If no matching entry is found, the search is satisfied in an alternative manner:

- If ORDER=ASCENDING is specified, the argument table entry having the largest argument value less than the search argument satisfies the search.
- If ORDER=DESCENDING is specified, the argument table entry having the smallest argument value greater than the search argument satisfies the search.

The results of the table search built-in function can be tested by examining the response codes, as follows:

Condition	ASM	COBOL	PL/I
Match found	X'00'	LOW-VALUES (TCATSMH)	00000000
ATABLE fa<ea (sa2<sa1)	X'04'	12-4-9 (TCATSER2)	00000100
FTABLE fa<ea (sa2<sa1)	X'08'	12-8-9 (TCATSER1)	00001000
No match	X'F0'	'0'	'0'

Note: Here, "fa" means "field address", and "ea" means "entry address". The names in parentheses in the COBOL column indicate the condition names generated by CICS. These names may be used in testing for the respective conditions in a COBOL program.

Example - Separate Tables

The following example shows how the macro is used in an assembler language program. A four-character argument is matched against fields in a seven-entry argument table. If the search is satisfied, the address of a two-character corresponding field in the function table is placed in TCATSA5 and the index value of the matching entry is placed in TCATSH4. If no matching entry is found, a branch to BR1 occurs.

```
         DFHBIF TYPE=TSEARCH,ARG=ARG1,
                ATABLE=(ATBL,AFLD,9,4,7),
                FTABLE=(FTBL,FFLD,5,2),
                ERROR=ERROR1,NOMATCH=BR1

ERROR1

BR1

ATBL     DS   0XL9 FIRST ENTRY OF ARG TABLE
         DS   XL5
AFLD     DS   XL4  FIRST ARGUMENT FIELD
         DS   6XL9 SPACE FOR 6 MORE ENTRIES
FTBL     DS   0XL5 FIRST ENTRY OF FUN TABLE
         DS   XL3
FFLD     DS   XL2  FIRST FUNCTION FIELD
         DS   6XL5 SPACE FOR 6 MORE ENTRIES
ARG1     DS   XL4  SEARCH ARGUMENT
```

Example - Complex Table

The following example shows how to search a complex table, that is, a table which contains both argument and function values. The search is similar to that above, except that only one table is described.

```
         DFHBIF TYPE=TSEARCH,ARG=ARG1,
                ATABLE=(TBL1,FLDA,5,2,3),
                FTABLE=(TBL1,FLDF,5,3),
                ERROR=ERROR1,NOMATCH=BR1

ERROR1

BR1

TBL1     DS  0CL5 ENTRY OF ARG/FUN TABLE
FLDA     DS  CL2  FIRST ARGUMENT FIELD
FLDF     DS  CL3  FIRST FUNCTION FIELD
         DS  2CL5 SPACE FOR 2 MORE ENTRIES
ARG1     DS  CL2  SEARCH ARGUMENT
```

PHONETIC CONVERSION BUILT-IN FUNCTION (TYPE=PHONETIC)

```
DFHBIF TYPE=PHONETIC
       [,FIELD=symb-addr]
       [,ERROR=symb-addr]
```

This macro is used to encode a 16-byte field of data phonetically. The macro converts a name into a key, which can be used to access data in a data base data set. The key that is generated is based upon the sound of the name; names that sound alike, even though spelled differently, produce identical keys. For example, the names SMITH, SMYTH, and SMYTHE produce the same key.

You should be aware that a change to the phonetic conversion built-in function has been incorporated in CICS Version 1 Release 7. The encoding routine ignores embedded blanks in a name. Records with phonetic keys encoded before this change may require reconversion before they are used with CICS Version 1 Release 7.

In addition to the phonetic conversion built-in function, a CICS subroutine that performs similar conversion of keys is available for use by offline user-written programs. Together, these facilities allow the CICS user to organize his file according to name (or any similar alphabetic key) and access the file using search arguments that may be misspelled or misunderstood to retrieve the required data.

Returned Value

The returned value is placed at TCAPHON. This value is the four-byte phonetic equivalent of the data passed to the built-in function. It consists of the first character of the data and three EBCDIC digits representing the characters in the remainder of the data.

If the first character is not alphabetic, an error code is placed in TCAPHNR. This code is X'50' for assembler language or '&' for both COBOL and PL/I. For COBOL, the condition name (TCAPINN) generated by CICS can be used instead.

Phonetic Conversion Subroutine

A CICS subroutine that performs phonetic conversion of 16-character names in the same manner as the phonetic conversion built-in function is available for use by offline user-written programs. The name to be converted is provided as input to the subroutine; the four-byte phonetic equivalent of that name is returned as a result.

The general form of the macro to invoke the subroutine (in ASM, COBOL, and PL/I respectively) is as follows:

```
CALL DFHPHN,(lang,name,phon)

CALL 'DFHPHN' USING lang name phon.

CALL DFHPHN (lang,name,phon);
```

lang is the symbolic address of a one-byte code indicating the programming language being used: X'F0' for ASM or COBOL, X'F1' for PL/I. If an error occurs during processing of this request, X'50' is returned in this location. If no error occurs, X'00' is returned, and the location must be reset to indicate the programming language before the location can be reused.

name is the symbolic address of the field that contains the 16-character name to be converted.

phon is the symbolic address of the field in which the four-byte phonetic equivalent of the name passed to the subroutine is returned to the calling program.

FIELD VERIFY BUILT-IN FUNCTION (TYPE=FVERIFY)

```
DFHBIF TYPE=FVERIFY
       [,FIELD=symb-addr]
       [,LENGTH={symb-addr|num-val}]
       [,ALPHA=symb-addr]
       [,NUMERIC=symb-addr]
       [,PACKED=symb-addr]
```

This macro verifies that the contents of a data field are:

- Entirely alphabetic: blanks or A-Z
- Entirely EBCDIC digits, with or without trailing sign: 0-9 (X'F0'-X'F9')
- Entirely packed decimal (COMP-3 in COBOL or FIXED DEC in PL/I).

A branch is made to an appropriate user-written routine accordingly.

The ALPHA, NUMERIC, and PACKED operands may be specified in any combination or order, but at least one of them must be specified. The conditions specified are tested upon request in the order ALPHA, NUMERIC, PACKED, irrespective of the order of the operands. If none of the test conditions is met, control goes to the instruction following the DFHBIF TYPE=FVERIFY macro in the application program.

Returned Values

The results of the verify built-in function can be tested by examining the response code at TCACHKR, as follows:

Condition	ASM	COBOL	PL/I
Packed field	X'20'	11-4-9 (TCACKPK)	00100000
Numeric field	X'40'	No punches (TCACKNM)	01000000
Alpha field	X'80'	12-0-1-8 (TCACKAL)	10000000
Mixed field	X'E0'	0-2-8 (TCACKMX)	11100000

Note: The names in parentheses in the COBOL column indicate the condition names generated by CICS. These names may be used in testing for the respective conditions in a COBOL program.

Example

```
DFHBIF TYPE=FVERIFY,FIELD=CONT,
       LENGTH=16,ALPHA=MYROUT
```

The contents of CONT, a 16-byte field, will be checked to see if it contains only alphabetic characters and/or blanks. If it does, control is transferred to MYROUT. Otherwise, control returns to the instruction following this DFHBIF macro in the application program.

FIELD EDIT BUILT-IN FUNCTION (TYPE=DEEDIT)

```
DFHBIF TYPE=DEEDIT
       [,FIELD=symb-addr]
       [,LENGTH={symb-addr|num-val}]
```

This macro specifies that all bytes (except the rightmost byte) containing other than EBCDIC numeric characters are to be deleted from the data field. The remaining characters are right justified in the field with zero padding at the left as necessary. 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 any hexadecimal character from X'A'-X'F'. The digit portion of this byte may contain one of the hexadecimal digits from X'0'-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. In any case, the returned value is in the field that initially contained the unedited data.

Note: A field of one byte will be returned unaltered, no matter what the field contains.

Example:

```
DFHBIF TYPE=DEEDIT,FIELD=CONTG,LENGTH=9
```

All characters other than EBCDIC numbers are removed from CONTG, a nine-byte field. The edited result is returned in that field to the application program. Say, for example, CONTG contains

```
14-6704/B
```

after editing, it will contain:

```
00146704B
```

BIT MANIPULATION BUILT-IN FUNCTIONS

The bit manipulation built-in functions are designed to change or test the state of specified bits in a given area of main storage. They are particularly useful for COBOL application programs, which are otherwise unable to manipulate bits.

TYPE=BITSETON

```
DFHBIF TYPE=BITSETON
       [,FIELD=symb-addr]
       [,BIT={symb-addr|value}]
       [,BITON=symb-addr]
       [,BITOFF=symb-addr]
```

This macro specifies that all bits selected by a specified bit pattern are on after execution of the macro.

The application programmer specifies the eight-bit mask (bit pattern) to be applied against the byte containing bits to be operated on. The mask can be described by a single EBCDIC character within single quotation marks: for example, '5' or 'M'.

Alternatively, the symbolic address of a one-byte field containing the mask can be specified. If desired, control can be transferred to a specified location if all affected bits (or all bits in the byte) are on after completion of the bit manipulation.

The returned value is the contents of the byte specified in the FIELD operand, with selected bits modified as specified; for example:

```
      DFHBIF  TYPE=BITSETON,
              FIELD=DATAF,
              BIT=PATERN,
              BITON=BLABEL
        .
        .
PATERN DC X'FF'
```

The macro ensures that all bits of the one-byte field DATAF are set on and causes a branch to BLABEL.

TYPE=BITSETOFF

```
DFHBIF TYPE=BITSETOFF
       [,FIELD=symb-addr]
       [,BIT={symb-addr|value}]
       [,BITON=symb-addr]
       [,BITOFF=symb-addr]
```

This macro specifies that all bits selected by a specified bit pattern are off after execution of this macro.

The application programmer specifies the eight-bit mask (bit pattern) to be applied against the byte containing bits to be operated on. The mask can be described by a single EBCDIC character within single quotation marks: for example, '5' or 'M'.

Alternatively, the symbolic address of a one-byte field containing the mask can be specified. If desired, control can be transferred to a specified location if all affected bits (or all bits in the byte) are off after completion of the bit manipulation.

The returned value is the contents of the byte specified in the FIELD operand, with selected bits modified as specified.

TYPE=BITFLIP

```
DFHBIF TYPE=BITFLIP
       [,FIELD=symb-addr]
       [,BIT={symb-addr|value}]
       [,BITON=symb-addr]
       [,BITOFF=symb-addr]
```

This macro specifies that the state of each bit selected by a specified bit pattern is changed.

The application programmer specifies the eight-bit mask (bit pattern) to be applied against the byte containing bits to be operated on. The mask can be described by a single EBCDIC character within single quotation marks: for example, '5' or 'M'.

Alternatively, the symbolic address of a one-byte field containing the mask can be specified. If desired, control can be transferred to a specified location if all affected bits (or all bits in the byte) are on, or if all affected bits (or all bits in the byte) are off, after completion of the bit manipulation.

The returned value is the contents of the byte specified in the FIELD operand, with selected bits modified as specified.

TYPE=BITEST

```
DFHBIF TYPE=BITEST
       [,FIELD=symb-addr]
       [,BIT={symb-addr|value}]
       [,BITON=symb-addr]
       [,BITOFF=symb-addr]
```

This macro specifies that the state of each bit in a specified bit pattern is to be tested and an indicator is to be set accordingly.

The BIT operand specifies the eight-bit mask (bit pattern) that is to be applied against the byte containing bits to be operated on. The mask can be described by a single EBCDIC character within single quotation marks: for example, '5' or 'M'.

Alternatively, the symbolic address of a one-byte field containing the mask can be specified. If desired, control can be transferred to a specified location if all affected bits (or all bits in the byte) are on, or if all affected bits (or all bits in the byte) are off, after completion of the bit manipulation.

Returned Values

For BITEST, the result of the test is returned in TCABITR as follows:

Condition	ASM	COBOL	PL/I
Bits off	X'00'	LOW-VALUES (TCABIFOF)	00000000
Bits on	X'F0'	'0' (TCABIFON)	'0'

Note: The names enclosed in parentheses in the COBOL column indicate the condition names generated by CICS. These names may be used in testing for the conditions in a COBOL program.

If BITON, BITOFF, or both BITON and BITOFF are specified, and if certain conditions are met as described in the explanations of these operands, control is transferred. For example, the macro:

```
DFHBIF TYPE=BITEST,BIT='A'
                  ,BITOFF=CLABEL
```

causes a bit pattern of 11000001 to be applied to the one-byte field whose address is stored in TCABITF. If all tested bits are off, control is transferred to CLABEL.

INPUT FORMATTING BUILT-IN FUNCTIONS

There are two versions of the DFHBIF built-in function to handle input formatting, as follows:

TYPE=DEFLDNM defines keywords in free-format input

TYPE=INFORMAT specifies formatting of terminal input

Both these versions are described later in the chapter.

The input formatting built-in function allows free-format terminal input to be converted into a predefined fixed format that can be processed by the application program. The application program can accept any of three forms of input (fixed, positional, or keyword) from the terminal. These forms are discussed in order of increased flexibility below.

FIXED FORMAT

This is the simplest case, but it requires a rigid adherence to form. The input transaction must be identical in format to the fixed internal format established by statements in the application program. For example, assume the fixed internal format for data consisting of a transaction identification; last name, first name, and middle initial; and identification number is as follows:

Cols	Description
1-4	Transaction id
5,6	blank
7-18	Last name
19,20	blank
21-30	First name
31,32	blank
33	Middle initial
34,35	blank
36-41	Identification no
42	EOB (end-of-block)

Each input field must be entered by the terminal operator in the positions established for it in the fixed internal format.

POSITIONAL FORMAT

This option allows the terminal operator to enter a system-programmer selected field-separator character to indicate the end of a field or the absence of a field. The order of the fields on input must be the same as the order established for the fixed internal format; the field lengths need not be the same. If other fields follow, the end of a field or the absence of a field within the input must be indicated by a field-separator character.

Assume that input consisting of a transaction identification; last name, first name, and middle initial; and identification number is to be entered from a terminal. Further assume that the input formatting built-in function is invoked by the application program to process this input, recognizing the slash (/) as a field-separator character. The following examples show permissible free-format positional input. Each input transaction is terminated by an EOB character.

- Complete input

  ```
  INQR/HUGHES/JOHN/Q/096556 EOB
  ```

- Middle initial unknown

  ```
  INQR/HUGHES/JOHN//096556 EOB
  ```

- Middle initial and identification number unknown

  ```
  INQR/HUGHES/JOHN EOB
  ```

KEYWORD FORMAT

The keyword format provides an even greater degree of flexibility in that terminal input can be entered in any order. The terminal operator need only be familiar with the keyword identifiers that have been established for the input fields. Each keyword identifier consists of up to four characters selected by the application programmer. The keyword identifier must be preceded by a field-name start character and followed by a field-separator character, both of which must be specified at system initialization by the system programmer. If either of these characters is not specified, the default assumed is a blank; however, since the field-name start character must be different from the field-separator character, it is invalid to take both defaults.

As an example, assume that keyword identifiers are established by use of the TYPE=DEFLDNM macro (described later in the chapter) as follows:

LN	-	last name field
FN	-	first name field
MI	-	middle initial field
ID	-	identification number

Further assume that the period has been specified as the field-name start character and the equal sign has been specified as the field-separator character. The following examples show permissible free-format keyword input.

- Complete input

  ```
  INQR.FN=JOHN.MI=Q.ID=096556.
  LN=HUGHES EOB
  ```

- First name unknown

  ```
  NQR.LN=HUGHES.MI=Q.ID=096556 EOB
  ```

- First name and identification number unknown

  ```
  INQR.LN=HUGHES.MI=Q EOB
  ```

The first entry in each of these examples is the transaction identification. Since CICS expects this identification, it must be first and no keyword identifier is required for it. Succeeding data fields are entered in any order. The input is terminated by an EOB character.

COMBINATION INPUT

DFHBIF macros can be included in an application program to permit a combination of fixed, positional, and keyword input. For example, the following variations may be allowed.

- Complete input

  ```
  INQR.LN=HUGHES.FN=JOHN/Q/096556 EOB
  INQR.FN=JOHN.LN=HUGHES.MI=Q/096556 EOB
  INQR/HUGHES/JOHN/Q  096556 EOB
  ```

- First name unknown

  ```
  INQR.LN=HUGHES//Q/096556 EOB
  INQR/HUGHES//Q.ID=096556 EOB
  INQR  HUGHES//Q  096556 EOB
  ```

- First name and identification number unknown

  ```
  INQR.LN=HUGHES//Q EOB
  INQR/HUGHES.MI=Q EOB
  INQR  HUGHES.MI=Q EOB
  ```

The application programmer can write a program to handle free-format input entered from a terminal. The free-format input may be either positional or keyword-oriented, or both, and may be entered in combination with fixed-format input. An example of positional is:

```
INQR/HUGHES/JOHN/Q/096556 EOB
```

An example of keyword-oriented is:

```
INQR.FN=JOHN.MI=Q.ID=096556.LN=HUGHES EOB
```

A task that issues DFHBIF macros to provide input formatting must be attached to a terminal.

Storage Definition

As a first step in defining storage, the programmer must copy the CICS control section of the terminal input/output area (TIOA) into his program. Definitions of the fields for which input data may be entered should follow the definition of the CICS control section. For example, the assembler language programmer may write the following code:

```
COPY DFHTIOA      START OF TIOA
TIOATI DS CL4     TRANS ID
TIOALN DS CL15    LAST NAME
TIOAFN DS CL9     FIRST NAME
TIOAMI DS CL1     MIDDLE INITIAL
TIOAID DS CL6     IDENTIFICATION
```

TIOADBA is the CICS-established name representing the first byte of the user's section of the TIOA for assembler language only. Succeeding names are application-programmer-selected identifiers of the input fields. (The copying of symbolic storage definitions is described in Part 2.)

TYPE=DEFLDNM

The format of the DFHBIF macro that defines keywords in free-format input is as follows:

```
DFHBIF TYPE=DEFLDNM
       ,NAMES=(keyword[,keyword,...])
       ,LABEL=symb-addr
```

If this macro is used in a COBOL program, it must appear in the working storage section of the program. It must appear with other data definitions in an assembler language or PL/I program. This macro is not needed if only free-format positional input is to be handled by a program.

For example, a DFHBIF TYPE=DEFLDNM macro defining keywords that the user can enter to refer to fields of the TIOA in the previous section, "Storage Definition," is:

```
DFHBIF TYPE=DEFLDNM,
       NAMES=(TI,LN,FN,MI,ID),
       LABEL=DEFI
```

In this example, the keywords are formed by taking the last two characters of the TIOA field names. Use of similar names within the DFHBIF macro and the TIOA definition is wise programming practice, but not a requirement. The following macro is also acceptable.

```
DFHBIF TYPE=DEFLDNM,
       NAMES=(TRAN,LAST,FIR,MID,IDEN),
       LABEL=MYIN
```

In both of these examples, the first keyword is a dummy name, because the first field will contain the transaction identification. The keyword for this field is provided to obtain the correlation between the TIOA definition and the macro, but would not appear in the free-format input from the terminal.

Required Delimiters

When providing free-format keyword-oriented input capabilities to terminal users, the application programmer, working with system programmers, must define a field-name start character and a field-separator character for the system before initialization. (See the appropriate CICS Resource Definition manual for details.)

TYPE=INFORMAT

The format of the DFHBIF macro that specifies formatting of terminal input is as follows:

```
DFHBIF TYPE=INFORMAT
       ,FIELDS=(symb-addr [,
          symb-addr,...])
       [,NAMES={symb-addr|YES}]
       [,LENGTH={symb-addr|
          numeric value}]
       [,ERROR=symb-addr]
```

Data entered as free-format input (keyword or positional) is read into a TIOA in the same manner as other data entered from a terminal. CICS places the address of the TIOA into TCTTEDA (as it must be for a formatting operation). To provide for the formatting of this free-format input, a DFHBIF TYPE=INFORMAT macro should be issued immediately following the terminal control (DFHTC) macro that causes data to be moved into the TIOA to make sure that the address of the TIOA containing data to be formatted is in TCTTEDA.

This built-in function reformats data from the input TIOA into a CICS-acquired TIOA. Data is moved from the input TIOA into locations in the output TIOA named in the FIELDS operand. The length of the data moved is the difference between displacements of the field being processed and the next field named in the FIELDS operand. All data is treated as alphanumeric, is left justified in each output field, and is padded on the right with blanks, or is truncated, as necessary.

If, however, the form of input is positional and an input data item is longer than the internal field defined for it, the data item is not truncated unless it is for the last field. Instead, a response code of 4 is set, as described below, and the data is treated as fixed-format input. (The reason for this treatment is that mixed formats are allowed, so it is impossible for the built-in functions program to distinguish between an intentional fixed-format input for more than one field and a positional input of excessive length for a single field.)

The input TIOA supplied by the user is released by the built-in function program (DFHBFP). The address of the fixed-format TIOA is returned in TCTTEDA to the application program. The application programmer should establish addressability to this TIOA immediately, just as for any TIOA used in the program. (See the instructions for copying symbolic storage definitions in Part 2.)

If the DFHBIF TYPE=INFORMAT macro is issued immediately following the read instruction, the address of the TIOA containing the data to be formatted will be stored in TCTTEDA. If any intervening macros are issued, the application programmer is responsible for saving and restoring the contents of TCTTEDA. For COBOL, TIOABAR must be loaded before a DFHBIF because the expansion will contain "CALL DFHCBLI USING fields."

Returned Values

The address of the fixed-format TIOA containing the reformatted data is available in TCTTEDA. This address must be loaded into TIOABAR, the base register for the area.

Certain error conditions may be detected during execution of the DFHBIF TYPE=INFORMAT macro. In such cases, an error indication (response code) is returned to the application program in TCAINRC, a one-byte field. The error conditions that may occur and the response code for each are as follows:

Condition	**ASM**	**COBOL**	**PL/I**
No error	X'00'	LOW-VALUES (TCAINNOE)	00000000
No FNS or FS[1]	X'20'	11-0-1-8-9 (TCAINALS)	00100000
Two FS chars[2]	X'F1'	'1' (TCAINER1)	'1'
Invalid name[3]	X'F2'	'2' (TCAINER2)	'2'
No macro[4]	X'F3'	'3' (TCAINER3)	'3'
Length[5]	X'F4'	'4' (TCAINER4)	'4'
Order wrong[6]	X'F5'	'5' (TCAINER5)	'5'

[1] The input data does not contain field name start (FNS) or field separator (FS) characters. (Such data may not be erroneous, if deliberately entered in this manner.)

[2] The input data contains two field name start characters with no field separator character between them.

[3] The input data contains an invalid name.

[4] A field name is specified in the input data, but no DFHBIF TYPE=DEFLDNM macro is contained in the application program.

[5] The length of an input data field exceeds the defined internal field size.

[6] The subparameters of the FIELDS operand are not specified in order of ascending displacement within the TIOA.

The names enclosed in parentheses in the COBOL column indicate the condition names generated by CICS. These names may be used in testing for the conditions in a COBOL program.

For error conditions other than X'F4', no reformatted data is returned; that is, TCTTEDA does not contain the address of a fixed-format TIOA containing the reformatted data.

Application programmers and terminal operators should be aware that if fixed-format input is provided to an application program designed to accept free-format input, field overrun (X'F4') errors are apt to occur.

Examples

Assume the TIOA definition and the first DFHBIF TYPE=DEFLDNM macro above. Further assume that the period has been established as the field-name start character and the equal sign and the slash as field-separator characters.

The free-format positional input

```
  INQR/HUGHES/JOHN/Q/096556 EOB
```

can be processed by issuing the following macro:

```
  DFHBIF TYPE=INFORMAT,
  FIELDS=(TIOAIN,TIOALN,TIOAFN,TIOAMI,
          TIOAID)
```

The free-format keyword input

```
  INQR.FN=JOHN.MI=Q.ID=096556.LN=HUGHES
   EOB
```

can be processed by issuing the following macro:

```
  DFHBIF TYPE=INFORMAT,
         FIELDS=(TIOAIN,TIOALN,TIOAFN,
         TIOAMI,TIOAID),
         NAMES=DEFI
```

A mixture of free-format positional and keyword input can be handled by this latter form of DFHBIF TYPE=INFORMAT macro. For example,

```
  INQR.LN=HUGHES//Q/096556 EOB
```

will be handled correctly.

WEIGHTED RETRIEVAL BUILT-IN FUNCTION

The weighted retrieval built-in function allows the application programmer to search a group of records in a VSAM key sequenced data set, selecting only those records that satisfy or are closest to the selection criteria provided.

In general, a series of DFHBIF macros is involved.

1. A DFHBIF TYPE=WTRETST macro indicates the start of a weighted retrieval operation.

2. One or more DFHBIF TYPE=WTRTPARM macros provide the specifications to be used by CICS in the weighted retrieval process.

3. One or more DFHBIF TYPE=WTRETGET macros retrieve one or more selected records.

4. A DFHBIF TYPE=WTRETREL macro releases the VSAM work area (VSWA) and other main storage used for the weighted retrieval process.

5. A DFHBIF TYPE=WTRETCHK macro performs a check on the success of a phase of the weighted retrieval process.

Each of these macros is discussed more fully below.

Each record with a specified partial key (beginning with the first one, or with the one having a specified relative number) is examined to see whether entries in certain other fields of the record match the values specified for those fields as selection criteria. Matching may be on exact comparison or within a given range.

Differentiating among individuals is one example of an area in which weighted retrieval processing is advantageous. In federal and state governments, the banking industry, and many other application areas dealing with large populations, name alone does not provide unique identification. A number of people may have the same name, so additional identifying characteristics are required. Such attributes as sex, race, birth date, address, relatives, and employment tend to permit unique identification. A basic example showing weighted retrieval on the basis of last name, first initial, and mother's maiden name is given later in this chapter.

Each comparison performed during weighted retrieval causes a match value to be added to a current total counter maintained automatically by CICS. If the comparison yields a match, the match value is also added to a weighted counter. If the compared fields do not match, the no-match value is subtracted from the weighted counter. Fields in the search criteria or in a record being examined that contain a predefined null character may be ignored (not included in the search) if desired.

When all fields to be considered in the selection process have been examined, a weighted qualification percentage (WQP) is calculated for the record. If this percentage is within the limits of acceptability established in the application program, the percentage and complete key of the record are saved in a key-save storage block.

After all records with the specified partial key have been examined, the saved keys are sorted into descending percentage-value order. Under normal processing, the records whose keys have been saved are retrieved one at a time and made available to the application program in order of decreasing acceptability. A further judgment as to acceptability or verification of identifiers is then made by the application program, which may involve the terminal operator in the final selection.

If the number of saved keys exceeds a maximum established in the application program (say, n), all keys having a weighted qualification percentage (WQP) equal to or lower than that of the "n+1"th key are dropped. If this dropping causes less than the application program-specified maximum number of keys to be saved but some keys are saved (as in Figure 28), no indication is given to the application program. However, if all percentages are the same so that all keys are dropped thereby, control is passed to an overflow routine (if one is specified in the application program). If the amount of storage required for saved keys exceeds the amount of storage available for keys, an overflow also occurs, and the application program is notified. An alternative, lower maximum can be established by the application program. The maximum number of records that can be retrieved is restricted by the maximum size of a key-save block (64K). This maximum is calculated as storage size divided by saved key length plus one.

Notes:

1. Because of the potential effect of weighted retrieval operations on system performance, this function should not be used indiscriminately. The amount of file accessing and the use of main storage should be taken into account.

2. The computations applied by CICS in weighted retrieval processing can be expressed as follows:

   Let MV=match value
   NMV=no-match value

   a. The weighted counter (WC), which holds the sum of all match values that had a match minus the sum of all no-match values that had no match, is given by:

      $$WC=(MV+MV+...+MV)-(NMV+NMV+...+NMV)$$

   b. The sum of all match values specified in WTRTPARM macros for the weighted retrieval operation; the potential count (PC) is given by:

      $$PC=MV+MV+...+MV$$

   c. The sum of all match values generated by the record comparisons (excludes those comparisons bypassed because the null character is present); the current total counter (CTC) is given by:

      $$CTC=MV+MV+...+MV \qquad n\leq k$$

   d. The weighted potential (WP) is given by:

      $$WP=\frac{PC+CTC}{2}$$

   e. The weighted qualification percentage (WQP) is given by:

      $$WQP=\frac{WC}{WP}$$

An overall effect of this method of computation is to provide a minimum weighting penalty for records having absent fields but yet prevent them from being chosen in preference to records that have all identifiers present.

INITIATE WEIGHTED RETRIEVAL (TYPE=WTRETST)

The format of the DFHBIF macro to indicate the start of a weighted retrieval operation is as follows:

```
DFHBIF TYPE=WTRETST
       [,DATASET=symb-name]
       [,RDIDADR=symb-addr]
       [,INPUTNO={symb-addr|
         numeric value|YES}]
       [,INPUTST={symb-addr|
         numeric value|YES}]
       [,INPUTPC=([suboperand1]
                 [,suboperand2])]
       [,NRECDS={symb-addr|
         numeric value|YES}]
       [,NORESP=symb-addr]
       [,DSIDER=symb-addr]
       [,NOTOPEN=symb-addr]
       [,NOTFND=symb-addr]
       [,INVREQ=symb-addr]
       [,IOERROR=symb-addr]
       [,OFLOW=symb-addr]
       [,ILLOGIC=symb-addr]
```

Returned Values

The address of a VSAM work area (VSWA) to be used by weighted retrieval throughout this series of weighted retrieval operations is returned in TCAWRAA, a four-byte field. Since any CICS macro issued within the application program may cause the contents of TCAWRAA to be changed, the application programmer should save this address. It must be restored in TCAWRAA prior to any subsequent DFHBIF macro included in this series of weighted retrieval operations. A response code indicating how CICS has responded to this request is returned in TCAWTRC, a one-byte field (see "Test Response to a Request for Weighted Retrieval," later in this chapter).

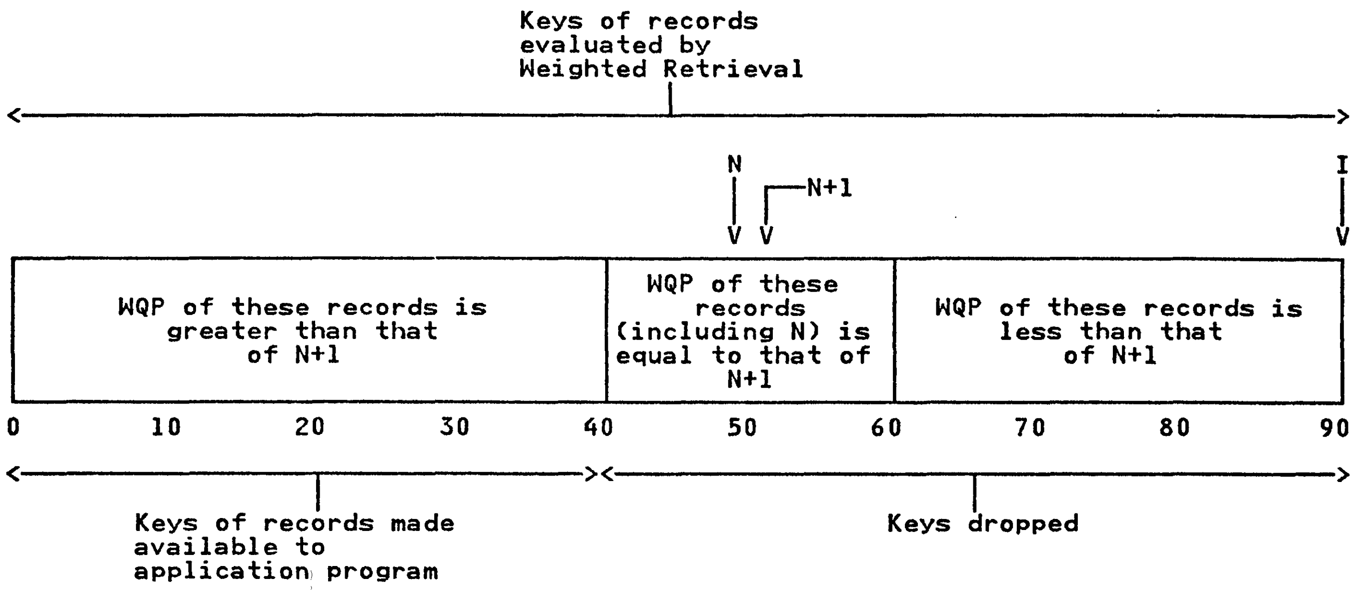

Keys of records evaluated by Weighted Retrieval
N
N+1
I
WQP of these records is greater than that of N+1
WQP of these records (including N) is equal to that of N+1
WQP of these records is less than that of N+1
0
10
20
30
40
50
60
70
80
90
Keys of records made available to application program
Keys dropped

Figure 28. Selection of Records by Weighted Retrieval

ESTABLISH SELECTION CRITERIA (TYPE=WTRTPARM)

The format of the DFHBIF macro to pass selection criteria to CICS is as follows:

```
DFHBIF TYPE=WTRTPARM
       [,FIELD1=([symb-addr][,
          numeric value][,char])]
       [,FIELD2=([symb-addr1][,
          symb-addr2])]
       [,NULL={symb-addr|character
          value|YES}]
       [,MATCH={symb-addr|
          numeric value}]
       [,NOMATCH={symb-addr|
          numeric value}]
       [,RANGE=(suboperand1,
          suboperand2[,suboperand3])]
```

One of these macros must be coded for each field that is to be examined during the selection process. Match and no-match values are established separately for each field. Then, during weighted retrieval processing, the applicable match and no-match values for examined fields of a record are used to determine a weighted qualification percentage for the record.

RETRIEVE SELECTED RECORDS (TYPE=WTRETGET)

The format of the DFHBIF macro to retrieve a record that has been saved by the weighted retrieval built-in function is as follows:

```
DFHBIF TYPE=WTRETGET
       [,NORESP=symb-addr]
       [,ENDFILE=symb-addr]
       [,NOTOPEN=symb-addr]
       [,NOTFND=symb-addr]
       [,INVREQ=symb-addr]
       [,IOERROR=symb-addr]
       [,OFLOW=symb-addr]
       [,ILLOGIC=symb-addr]
```

This macro specifies that next record saved by weighted retrieval (as ordered according to decreasing weighted qualification percentage) is to be retrieved.

Before this macro is executed, TCAWRAA must contain the address of the VSAM work area (VSWA) used in this series of weighted retrieval operations.

Returned Values

A record saved as the result of weighted retrieval is returned to the application program. The address of this record is contained in VSWAREA within the VSWA provided by the WTRETST macro. The length of the record is returned in VSWALEN.

In addition, the contents of several halfword fields are significant. TCAWGH1 contains the highest percentage of acceptability for this weighted retrieval operation, TCAWGH2 contains the lowest percentage of acceptability for this weighted retrieval operation, TCAWGH3 contains the percentage of acceptability of this record, and TCAWGH4 contains the number of records left to be presented to the user. After the first DFHBIF TYPE=WTRETGET macro, TCAWGH5 contains a count of any records dropped to remain within the maximum specified in the NRECDS operand of the WTRETST macro. On succeeding WTRETGETs, TCAWGH5 contains zero. The full key of the returned record is returned at the location specified in the RDIDADR operand of the DFHBIF TYPE=WTRETST macro initiating this weighted retrieval operation.

TCABFTR, a one-byte field, contains the response code that describes the CICS response to this DFHBIF TYPE=WTRETGET macro. This response code can be interrogated as described under "Test Response to a Request for Weighted Retrieval," below.

RELEASE WEIGHTED RETRIEVAL STORAGE AREAS (TYPE=WTRETREL)

The format of the DFHBIF macro to specify that the VSWA established when the DFHBIF TYPE=WTRETST macro is issued and the main storage used for saving the records is released is as follows:

```
DFHBIF TYPE=WTRETREL
       [,NORESP=symb-addr]
       [,INVREQ=symb-addr]
       [,ILLOGIC=symb-addr]
```

TEST RESPONSE TO A REQUEST FOR WEIGHTED RETRIEVAL (TYPE=WTRETCHK)

The general format of the DFHBIF TYPE=WTRETCHK macro is as follows:

```
DFHBIF TYPE=WTRETCHK
       [,NORESP=symb-addr]
       [,DSIDER=symb-addr]
       [,NOTOPEN=symb-addr]
       [,NOTFND=symb-addr]
       [,INVREQ=symb-addr]
       [,ENDFILE=symb-addr]
       [,IOERROR=symb-addr]
       [,OFLOW=symb-addr]
       [,ILLOGIC=symb-addr]
```

WEIGHTED RETRIEVAL RESPONSE CODES

The response codes and their corresponding conditions are as follows:

Condition	ASM	COBOL	PL/I
NORESP	X'00'	LOW-VALUES	00000000
DSIDER	X'C1'	'A'	'A'
NOTOPEN	X'C2'	'B'	'B'
NOTFND	X'C8'	'H'	'H'
ENDFILE	X'C4'	'D'	'D'
INVREQ[1]	X'C3'	'C'	'C'
IOERROR	X'C5'	'E'	'E'
OFLOW[2]	X'C6'	'F'	'F'
ILLOGIC[3]	X'C7'	'G'	'G'

Notes:

1. If the data set is not a VSAM file, the field TCAWRAA is set to zero. CICS file control handles other errors of this type, in which case, TCAWRAA contains the address of the FCT entry for the data set.

2. For WTRETST, this response indicates that the system-defined maximum storage GETMAIN (64K) is insufficient to hold all retrieved record keys and these record keys and these records have the same percentage of acceptability. In this case, the terminal operator must specify a relative record number (the relative position of the first record to be retrieved among the retrieval records) and a number of records (NRECDS) to be presented. For WTRETGET, this response means that no records were returned because all had identical percentages of match and not all could be returned because of the limit specified in NRECDS.

3. This response indicates that a VSAM error that does not fall into one of the above categories has occurred. The VSWA contains the VSAM request parameter list that contains the VSAM logical error.

If checking for a response to a weighted retrieval macro is not provided, and if the condition corresponding to that response occurs, program flow proceeds to the instruction following the weighted retrieval macro in the application program.

Example

The following example shows the use of weighted retrieval on the basis of last name, first initial, and mother's maiden name.

Assume that, for purposes of state welfare applications, a VSAM file labeled SRCHFILE is maintained on disk. SRCHRECD is an area of storage that holds individual records retrieved from the file.

The file is to be searched to retrieve up to 100 records that satisfy (or come closest to satisfying) the criteria:

- Last name = SMITH
- First initial = J
- Mother's name = MARY.

Example of Weighted Retrieval

```
         COPY   DFHTCADS
LNAME    DS     CL18
FINIT    DS     CL1
MOM      DS     CL7
         DFHBFTCA OPTION=WTRET
          .
          .
SRCHRECD DSECT
         USING *,RCDBASE
LAST     DS     CL18
FIRST    DS     CL1
MOTHER   DS     CL7
          .
          .
         DFHBIF TYPE=WTRETST,DATASET=SRCHFILE,RDIDADR=KEYFLD,         *
               INPUTNO=100,INPUTST=10,INPUTPC=(100,80),NRECDS=50,     *
               NORESP=STARTOK
          .
         (Error processing)
          .
STARTOK  DS     0H
         L      VSWABAR,TCAWRAA
         DFHBIF TYPE=WTRTPARM,FIELD1=(LNAME,18,C),                     *
               FIELD2=(SRCHRECD,LAST),MATCH=50
         DFHBIF TYPE=WTRTPARM,FIELD1=(FINIT,1,C),                      *
               FIELD2=(SRCHRECD,FIRST),MATCH=30
         DFHBIF TYPE=WTRTPARM,FIELD1=(MOM,7,C),                        *
               FIELD2=(SRCHRECD,MOTHER),MATCH=20
WRGET    DS     0H
         ST     VSWABAR,TCAWRAA
         DFHBIF TYPE=WTRETGET,NORESP=GETOK,ENDFILE=ENDPROC
          .
         (Error processing)
          .
GETOK    DS     0H
         L      RCDBASE,VSWAREA     GET ADDRESSABILITY TO RECORD
          .
         (On first WTRETGET, check if too many records have been skipped
         enough records returned, acceptable range of % returned, and
         the like.)
          .
         B      WRGET               PROCESS RETRIEVED RECORD
          .
ENDPROC  DS     0H
          .
         ST     VSWABAR,TCAWRAA
         DFHBIF TYPE=WRETREL
```

OPERANDS OF DFHBIF MACRO

ALPHA=symb-addr
is the address to which control is to be passed if the field consists entirely of alphabetic characters (A through Z) and/or blanks.

ARG=symb-addr
is the symbolic address of the field that contains the search argument; if omitted, the address is assumed to be in TCATSA1, a fullword field.

ATABLE=
is a description of the table to be searched (the argument table).

symb-addr1
is the address of the first entry in the argument table; if omitted, the address is assumed to be in TCATSA2, a fullword field.

symb-addr2 or YES
is the address of the field in the first entry of the argument table to be compared with the search argument. If YES is specified, the field address is assumed to be in TCATSA4, a fullword field. If this operand entry is omitted, symbolic address2 is assumed to be the same as symbolic address1. If specified, the address represented by symbolic address2 must be equal to or greater than the address represented by symbolic address1. If it is not, bit 4 of TCATSRC is set on and no search is made.

numeric value1
is the length of each entry in the argument table (including any other fields in the entry or slack bytes required for boundary alignment). A value in the range from 1 through 32767 may be specified. If this operand entry is omitted, the length is assumed to be in TCATSH2, a halfword field.

numeric value2 or YES
is the length of the field in the argument table to be compared with the search argument. If YES is specified, the length is assumed to be in TCATSAF, a one-byte field. If this operand entry is omitted, numeric value2 is assumed to be the same as numeric value1. If specified, the value must be between 1 and 255 inclusive. If numeric value1 is not within this range, numeric value2 must be specified.

numeric value3
is the maximum number of entries to be searched. A value in the range from 1 through 32767 may be specified. If this operand entry is omitted, the numeric value is assumed to be in TCATSH1, a halfword field.

If one or more of these operand entries are omitted, but other operand entries follow, the comma that ordinarily follows an omitted entry must be included in the operand.

BIT=
specifies the bit pattern (mask) to be applied to the specified byte.

symb-addr
is the address of a byte that contains the bit pattern.

value
is a single EBCDIC character enclosed in single quotation marks.

If this operand is omitted, the bit pattern is assumed to be in TCABITV, a one-byte field.

BITOFF=symb-addr
is the symbolic address to which control is transferred if:

- For BITSETON, BITSETOFF, or BITFLIP:

  All bits in the specified byte are off after the operation is completed.

- For BITEST:

  All bits that are on in the bit pattern are off in the field that is tested.

BITON=symb-addr
is the symbolic address to which control is transferred if:

- For BITSETON, BITSETOFF, or BITFLIP:

  All bits in the specified byte are on after the operation is completed.

- For BITEST:

  All bits that are on in the bit pattern are on in the field that is tested.

DATASET=symbolic name
is the one- to eight-character identification of the VSAM data set

from which the retrieval is to be made; if omitted, the data set identifier is assumed to be in TCAWTDI, an eight-byte field.

DSIDER=symb-addr
specifies the entry label of the user-written routine to which control is to be passed if the data set specified by the DATASET operand cannot be located. DSIDER signifies "data set identification error."

ENDFILE=symb-addr
specifies the entry label of the user-written routine to which control is to be passed if an end-of-file condition is detected.

ERROR=symb-addr
is the address to which control is to be passed if an error occurs. This branch is taken, for example, if the address specified for the function field to be examined is lower than the address specified for the first function table entry.

FIELD=symb-addr
is the symbolic address of the byte or field to be processed; if omitted, the address is assumed to be in the fullword field TCACKFD (for TYPE=FVERIFY), TCAFLD (for TYPE=DEEDIT), or TCABITF (for TYPE=BITSETON). However, for TYPE=PHONETIC, the data is assumed to be in TCANAME.

FIELDS=(symb-addr[,symb-addr,...])
are the labels of fields defined within the internal fixed-format TIOA to which the input data is to be transferred. The fields must be named in order of increasing displacement from the start of the TIOA, and there must be a one-to-one correspondence between the field names in this macro and the fields in the TIOA. The length of each field is determined by calculations based on the location represented by the symbolic address of the following field. Each field should be at least one byte in length. For positional input, each field for which data may be entered (that is, each position in the receiving area of storage) must be defined.

FIELD1=
specifies the characteristics of the search field to be compared against a corresponding field in records of the data set on which the weighted retrieval function is to operate.

symb-addr
is the symbolic address of the field. If omitted, the address of the field is assumed to be in TCAWPA1, a four-byte field.

numeric value
is the length of the field in bytes and may range from 1 to 32767. If omitted, the length of the field is assumed to be in TCAWPH1, a halfword field.

char
is one character indicating the format of the data in the field as follows:

Char	**Data Format**
C	EBCDIC characters
Z	Zoned decimal numbers
P	Packed decimal numbers
H	Halfword binary
F	Fullword binary

If this parameter is omitted, the character is assumed to be in TCAWPB1, a one-byte field.

If one of these operand entries is omitted but succeeding operand entries follow, the comma that otherwise follows the entry must be included in the operand.

Notes:

1. The application programmer must ensure that the integrity of FIELD1 is not destroyed prior to the first DFHBIF TYPE=WTRETGET macro. These values are used by the built-in functions program (DFHBFP) at that time. In particular, it is not advisable to utilize an area within a TIOA for this value.

2. The largest decimal number that can be contained in a zoned decimal (Z) or packed decimal (P) field cannot exceed 16 digits, including the sign.

FIELD2=
specifies the location of the data in the field of each record of the data set involved in the comparison with the search data in FIELD1.

symb-addr1
is the symbolic address (label) of the first byte of the storage area that will contain the record to be examined. If omitted, the address of the main storage area is assumed to be in TCAWPA3, a four-byte field.

symb-addr2
is the symbolic address (label) of the field within the storage area identified by symbolic address1 to be used

in the weighted retrieval comparison. If omitted, the address of the field is assumed to be in TCAWPA4, a four-byte field.

If the first operand entry is omitted but the second is specified, the comma that otherwise follows the first entry must be included in the operand.

FTABLE=
is a description of the table from which a value is to be retrieved (the function table). If no function value is to be retrieved (for example, if only the index of a matching argument table entry is needed), this operand can be omitted. If this operand is specified, but some entries are omitted, the values of the corresponding entries in the ATABLE operand are assumed to apply.

If a complex table (where each table entry contains both an argument and a function value) is being searched, the argument table and function table, as defined for this macro, are actually within the same table in storage. Alternatively, two separate tables, one containing search fields and one containing function values, may be used.

symb-addr1 or YES
is the address of the first function table entry. If YES is specified, the address is assumed to be in TCATSA3, a fullword field.

symb-addr2 or YES
is the address of the function field within the first function table entry. If YES is specified, the address is assumed to be in TCATSA5, a fullword field. This address must be equal to or greater than symbolic address1. If it is not, bit 5 of TCATSRPC is set on and no search is made.

numeric value1 or YES
is the length of each entry in the function table (including any other fields in the entry or slack bytes required for boundary alignment). A value in the range from 1 through 32767 may be specified. If YES is specified, the value is assumed to be in TCATSH3, a halfword field.

numeric value2 or YES
is the length of the field to be retrieved from the function table. If YES is specified, the length is assumed to be in TCATSFF, a one-byte field. The length must be between 1 and 255 inclusive. If this operand is omitted, the default is the corresponding entry in the ATABLE, or its default if the corresponding entry is not specified in the ATABLE. The default for this operand is not numeric value1 above.

ILLOGIC=symb-addr
specifies the entry label of the user-written routine to which control is to be passed if a VSAM error that does not fall within one of the other CICS response categories occurs.

INDEX=symb-addr
specifies the address of a halfword field in which an index value relative to one, identifying the matching argument-table entry, is to be returned to the application program. In addition, the index value is placed in TCATSH4, a halfword field, whether or not the INDEX operand is specified. Both fields contain zero if no matching entry is found.

INPUTNO=
specifies the maximum number of records that can be examined. A value from 1 to 32767 may be specified.

symb-addr
is the address of a halfword field that contains the maximum number of records that can be examined.

numeric value
is a decimal numeral indicating the maximum number of records that can be examined.

YES
indicates that the maximum number of records to be examined has been placed in TCAWTH1, a halfword field.

If this operand is omitted, a default value of 512 is placed in TCAWTH1.

INPUTPC=
specifies the percentages to be used by the weighted retrieval built-in function to determine the limits of acceptability.

suboperand1
specifies the maximum percentage, a value from 0 to 100; this value can be indicated by the symbolic address of a halfword field

containing the maximum value, a decimal numeral, or YES, which indicates that the value has been placed in TCAWTH3. If omitted, the maximum percentage is assumed to be 100.

suboperand2
specifies the minimum percentage, a value from 0 to 100; this value can be indicated by the symbolic address of a halfword field containing the minimum value, a decimal numeral, or YES, which indicates that the value has been placed in TCAWTH4. If omitted, the minimum percentage is assumed to be 0.

If the first suboperand is omitted, but the second is specified, the comma that otherwise follows the first suboperand must be included. If only one suboperand is given, it is assumed to be the first suboperand (the maximum percentage, 100).

INPUTST=
indicates the number of records with the specified partial key to be skipped before examination of records begins. The maximum value that can be specified is 32767.

symb-addr
is the address of a halfword field that contains the relative number of the record that is to be examined first.

numeric value
is a decimal numeral indicating the relative number of the record that is to be examined first.

YES
indicates that the relative number of the desired record has been placed in TCAWTH2, a halfword field.

If this operand is omitted, a default value of 0 is placed in TCAWTH2. The weighted retrieval begins with the first record having the specified partial key.

INVREQ=symb-addr
specifies the entry label of the user-written routine to which control is to be passed if an invalid type of request is received.

IOERROR=symb-addr
specifies the entry label of the user-written routine to which control is to be passed if an input/output error occurs.

LABEL=symb-addr
is the label to be assigned to the list of keywords. This label must be unique within the application program and may be from one to eight characters in length.

LENGTH=
specifies the length of the field to be processed or the size of the TIOA to be acquired.

symb-addr
is the address of a halfword field that contains the length value.

numeric value
is the length, in bytes, of the field to be processed, or the area to be acquired for the TIOA.

The maximum length of a field is 32767 bytes. The length of the TIOA must be sufficient to accommodate all fields specified in the FIELDS operand.

If this operand is omitted, the length is assumed to be in the halfword field TCACKLN ((for TYPE=FVERIFY), TCAFLN (for TYPE=DEEDIT), or in TCAINH1 (for TYPE=INFORMAT).

MATCH=
specifies a value to be added to the current total counter if the comparison is performed and to the weighted counter if the compared fields are equal. The value may range from -32768 through +32767.

symb-addr
is the symbolic address of a halfword field containing the value.

numeric value
is a decimal numeral in the range stated above.

If this parameter is omitted, the value is assumed to be in TCAWPH2.

Note: All match and no-match values specified for a weighted retrieval operation must have like signs.

NAMES=
indicates that field names may be present as keywords in the input data stream.

symb-addr
is the LABEL parameter specified in a DFHBIF TYPE=DEFLDNM macro in which the keywords that may be specified are defined.

YES
indicates that the label specified in the DFHBIF TYPE=DEFLDNM macro defining the field names is in TCAINA2, a fullword field.

(keyword[,keyword,...])
is a list of the keywords that may be entered by the terminal user to indicate which fields are to receive input data. Each keyword may be from one to four characters in length. Any combination of alphabetic, numeric, and/or special characters may be specified. The keywords must be specified in the order in which the corresponding fields that will hold the data are defined in the fixed-format TIOA.

NOMATCH=
specifies a value to be used during weighted retrieval, or the address to which control is to be passed if matching is unsuccessful.

symb-addr
is the symbolic address of a halfword field containing the value to be subtracted from the weighted counter if the compared fields are not equal, or it is the address to which control is to be passed if no table entry matching the search argument is found.

numeric value
is a decimal numeral in the range -32768 through +32767.

If this parameter is omitted, the value is assumed to be in TCAWPH3.

Note: All match and no-match values specified for a weighted retrieval operation must have like signs.

If this operand is specified, the SUBST operand cannot be specified.

NORESP=symb-addr
specifies the entry label of the user-written routine to which control is to be passed if no error occurs. NORESP signifies "normal response."

NOTFND=symb-addr
specifies the entry label of the user-written routine to which control is to be passed if an attempt at weighted retrieval is unsuccessful. NOTFND signifies "record not found."

NOTOPEN-symb-addr
specifies the entry label of the user-written routine to which control is to be passed if the requested data set is not open.

NRECDS=
indicates the maximum number of records to be made available to the application program. A value from 1 to 32767 can be specified.

symb-addr
is the address of a halfword field that contains the maximum number of records.

numeric value
is a decimal numeral indicating the maximum number of records.

YES
indicates that the maximum number has been placed in TCAWGCNT, a halfword field.

If this operand is omitted, a default value of 200 is assumed.

NULL=
specifies a one-byte "null character" which, if present in either FIELD1 or FIELD2, indicates that no comparison is to be performed.

symb-addr
is the symbolic address of a one-byte field containing the null character.

character value
is a single EBCDIC character within single quotation marks.

YES
indicates that the null character has been placed in TCAWPNL, a one-byte field.

The null character cannot be a binary zero (that is, X'00'); such a specification is ignored.

NUMERIC=symb-addr
is the address to which control is to be passed if the field consists entirely of EBCDIC numbers (X'F0' through X'F9') with an optional trailing minus sign or 'CR'.

OFLOW=symb-addr
specifies the entry label of the user-written routine to which control is to be passed if an overflow condition occurs.

ORDER=
describes the sequence used in ordering the entries of the argument table and is optional if RANGE is not specified. The sequence must be EBCDIC; packed, fullword and halfword binary, and floating-point tables cannot be

searched. When this parameter is specified, a quick binary search is used (rather than a sequential search).

ASCENDING
indicates that table entries are organized in ascending order according to the entries in the field to be compared with the search argument.

DESCENDING
indicates that table entries are organized in descending order according to the entries in the field to be compared with the search argument.

In either case, the field values are interpreted as EBCDIC representations. If this operand is not specified, the argument table is assumed to be unordered and is searched sequentially, starting at the last entry of the table.

PACKED=symb-addr
is the address to which control is to be passed if the field consists entirely of packed decimal characters, that is, of half-bytes with hexadecimal values in the range 0 through 9, except for the rightmost half-byte, which must contain a hexadecimal value in the range A through F.

RANGE=YES
is an optional operand indicating that, if no field compared with the search argument is an exact match, an existing table entry that would be adjacent to such an entry is to be taken as the function value. When this operand is specified, ORDER must be specified; otherwise, a sequential search of the table is made, starting at the last entry.

RANGE=
indicates that the compared fields are to be considered equal if FIELD2 falls within a given range of FIELD1.

suboperand1
specifies the type of range used in the comparison. This entry can be a single character or YES, which indicates that the single character specifying type has been placed in TCAWPTR. The valid characters are as follows:

Char	**Type of Range**
P	Percentage
U	Units
V	Value

suboperand2
specifies the upper limit, exceeding the value in FIELD1, which is to be considered a match. This entry can be a positive numeric value up to 32767 or YES, which indicates that the upper limit has been placed in TCAWPH4.

suboperand3
specifies the lower limit, below the value in FIELD1, which is to be considered a match. This entry can be a positive numeric value up to 32767 or YES, which indicates that the lower limit has been placed in TCAWPH5.

If suboperand3 is omitted, suboperand2 is assumed to apply both above and below the value in FIELD1. For example, if the value in FIELD1 is 165 and RANGE=(U,5) is specified, then any value from 160 through 170 is considered a match. If RANGE=(U,5,10) is specified, then any value between 155 and 170 is considered a match. If RANGE=(P,20) is specified, then any value between 132 and 198 {165×(1±20%)}is acceptable. If RANGE=(V,190,160), then any value between 160 and 190 is acceptable. If the data field contains EBCDIC characters (that is, C is specified in the FIELD1 operand), the RANGE operand is ignored.

Note: The upper bound and lower bound values are computed using the following formulas (where K is the value of FIELD1):

1. For P-type range, specified (P,x,y) or (P,x):

```
UB=K*(1+x/100)  UB=K*(1+x/100)
                or
LB=K*(1-y/100)  LB=K*(1-x/100)
```

2. For U-type range, specified (U,x,y) or (U,x):

```
UB=K+x  UB=K+x
      or
LB=K-y  LB=K-x
```

3. For V-type range, specified (V,x,y):

```
UB=x

LB=y
```

RDIDADR=symb-addr
is the symbolic address of the record identification field that contains the partial key of the record at which the data set is to be positioned prior to the retrieval process; if omitted, the address is assumed to be in

TCAWTRI, a fullword field. (The format of the record identification field for a VSAM data set is described under "Record Identification Field" in "Chapter 3.1. Introduction to Files and Data Bases" on page 49.)

SUBST=
specifies a value to be stored in TARGET if no entry matching the search argument is found in the argument table.

symb-addr
is the address of a field that contains the value to be stored.

'literal value'
is the value to be stored; single quotation marks must enclose the value in this specification but are not stored as part of the data.

If this operand is specified, the TARGET operand must be specified, and the NOMATCH operand cannot be specified.

TARGET=symb-addr
is the symbolic address of the field in which the built-in function value is to be returned to the application program. The address of the function value is placed in TCATSA5, a fullword field, regardless of whether TARGET is specified.

ALPHA=symb-addr
is the address to which control is to be passed if the field consists entirely of alphabetic characters (A through Z) and/or blanks.

ARG=symb-addr
is the symbolic address of the field that contains the search argument; if omitted, the address is assumed to be in TCATSA1, a fullword field.

ATABLE=
is a description of the table to be searched (the argument table).

symb-addr1
is the address of the first entry in the argument table; if omitted, the address is assumed to be in TCATSA2, a fullword field.

symb-addr2 or YES
is the address of the field in the first entry of the argument table to be compared with the search argument. If YES is specified, the field address is assumed to be in TCATSA4, a fullword field. If this operand entry is omitted, symbolic address2 is assumed to be the same as symbolic address1. If specified, the address represented by symbolic address2 must be equal to or greater than the address represented by symbolic address1. If it is not, bit 4 of TCATSRC is set on and no search is made.

numeric value1
is the length of each entry in the argument table (including any other fields in the entry or slack bytes required for boundary alignment). A value in the range from 1 through 32767 may be specified. If this operand entry is omitted, the length is assumed to be in TCATSH2, a halfword field.

numeric value2 or YES
is the length of the field in the argument table to be compared with the search argument. If YES is specified, the length is assumed to be in TCATSAF, a one-byte field. If this operand entry is omitted, numeric value2 is assumed to be the same as numeric value1. If specified, the value must be between 1 and 255 inclusive. If numeric value1 is not within this range, numeric value2 must be specified.

numeric value3
is the maximum number of entries to be searched. A value in the range from 1 through 32767 may be specified. If this operand entry is omitted, the numeric value is assumed to be in TCATSH1, a halfword field.

If one or more of these operand entries are omitted, but other operand entries follow, the comma that ordinarily follows an omitted entry must be included in the operand.

BIT=
specifies the bit pattern (mask) to be applied to the specified byte.

symb-addr
is the address of a byte that contains the bit pattern.

value
is a single EBCDIC character enclosed in single quotation marks.

If this operand is omitted, the bit pattern is assumed to be in TCABITV, a one-byte field.

BITOFF=symb-addr
is the symbolic address to which control is transferred if:

- For BITSETON, BITSETOFF, or BITFLIP:

  All bits in the specified byte are off after the operation is completed.

- For BITEST:

  All bits that are on in the bit pattern are off in the field that is tested.

BITON=symb-addr
is the symbolic address to which control is transferred if:

- For BITSETON, BITSETOFF, or BITFLIP:

  All bits in the specified byte are on after the operation is completed.

- For BITEST:

  All bits that are on in the bit pattern are on in the field that is tested.

ERROR=symb-addr
is the address to which control is to be passed if an error occurs. This branch is taken, for example, if the address specified for the function field to be examined is lower than the address specified for the first function table entry.

FIELD=symb-addr
is the symbolic address of the byte or field to be processed; if omitted, the address is assumed to be in the fullword field TCACKFD (for TYPE=FVERIFY), TCAFLD (for TYPE=DEEDIT), or TCABITF (for TYPE=BITSETON). However, for TYPE=PHONETIC, the data is assumed to be in TCANAME.

FIELDS=(symb-addr[,symb-addr,...])
are the labels of fields defined within the internal fixed-format TIOA to which the input data is to be transferred. The fields must be named in order of increasing displacement from the start of the TIOA, and there must be a one-to-one correspondence between the field names in this macro and the fields in the TIOA. The length of each field is determined by calculations based on the location represented by the symbolic address of the following field. Each field should be at least one byte in length. For positional input, each field for which data may be entered (that is, each position in the receiving area of storage) must be defined.

FTABLE=
is a description of the table from which a value is to be retrieved (the function table). If no function value is to be retrieved (for example, if only the index of a matching argument table entry is needed), this operand can be omitted. If this operand is specified, but some entries are omitted, the values of the corresponding entries in the ATABLE operand are assumed to apply.

If a complex table (where each table entry contains both an argument and a function value) is being searched, the argument table and function table, as defined for this macro, are actually within the same table in storage. Alternatively, two separate tables, one containing search fields and one containing function values, may be used.

symb-addr1 or YES
is the address of the first function table entry. If YES is specified, the address is assumed to be in TCATSA3, a fullword field.

symb-addr2 or YES
is the address of the function field within the first function table entry. If YES is specified, the address is assumed to be in TCATSA5, a fullword field. This address must be equal to or greater than symbolic address1. If it is not, bit 5 of TCATSRPC is set on and no search is made.

numeric value1 or YES
is the length of each entry in the function table (including any other fields in the entry or slack bytes required for boundary alignment). A value in the range from 1 through 32767 may be specified. If YES is specified, the value is assumed to be in TCATSH3, a halfword field.

numeric value2 or YES
is the length of the field to be retrieved from the function table. If YES is specified, the length is assumed to be in TCATSFF, a one-byte field. The length must be between 1 and 255 inclusive. If this operand is omitted, the default is the corresponding entry in the ATABLE, or its default if the corresponding entry is not specified in the ATABLE. The default for this

operand is not numeric valuel above.

ILLOGIC=symb-addr
specifies the entry label of the user-written routine to which control is to be passed if a VSAM error that does not fall within one of the other CICS response categories occurs.

INDEX=symb-addr
specifies the address of a halfword field in which an index value relative to one, identifying the matching argument-table entry, is to be returned to the application program. In addition, the index value is placed in TCATSH4, a halfword field, whether or not the INDEX operand is specified. Both fields contain zero if no matching entry is found.

LABEL=symb-addr
is the label to be assigned to the list of keywords. This label must be unique within the application program and may be from one to eight characters in length.

LENGTH=
specifies the length of the field to be processed or the size of the TIOA to be acquired.

symb-addr
is the address of a halfword field that contains the length value.

numeric value
is the length, in bytes, of the field to be processed, or the area to be acquired for the TIOA.

The maximum length of a field is 32767 bytes. The length of the TIOA must be sufficient to accommodate all fields specified in the FIELDS operand.

If this operand is omitted, the length is assumed to be in the halfword field TCACKLN ((for TYPE=FVERIFY), TCAFLN (for TYPE=DEEDIT), or in TCAINH1 (for TYPE=INFORMAT).

NAMES=
indicates that field names may be present as keywords in the input data stream.

symb-addr
is the LABEL parameter specified in a DFHBIF TYPE=DEFLDNM macro in which the keywords that may be specified are defined.

YES
indicates that the label specified in the DFHBIF TYPE=DEFLDNM macro defining the field names is in TCAINA2, a fullword field.

(keyword[,keyword,...])
is a list of the keywords that may be entered by the terminal user to indicate which fields are to receive input data. Each keyword may be from one to four characters in length. Any combination of alphabetic, numeric, and/or special characters may be specified. The keywords must be specified in the order in which the corresponding fields that will hold the data are defined in the fixed-format TIOA.

NOMATCH=
specifies the address to which control is to be passed if matching is unsuccessful.

symb-addr
is the address to which control is to be passed if no table entry matching the search argument is found. is a decimal numeral in the range -32768 through +32767.

If this parameter is omitted, the value is assumed to be in TCAWPH3.

If this operand is specified, the SUBST operand cannot be specified.

NUMERIC=symb-addr
is the address to which control is to be passed if the field consists entirely of EBCDIC numbers (X'F0' through X'F9') with an optional trailing minus sign or 'CR'.

ORDER=
describes the sequence used in ordering the entries of the argument table and is optional if RANGE is not specified. The sequence must be EBCDIC; packed, fullword and halfword binary, and floating-point tables cannot be searched. When this parameter is specified, a quick binary search is used (rather than a sequential search).

ASCENDING
indicates that table entries are organized in ascending order according to the entries in the field to be compared with the search argument.

DESCENDING
indicates that table entries are organized in descending order according to the entries

in the field to be compared with the search argument.

In either case, the field values are interpreted as EBCDIC representations. If this operand is not specified, the argument table is assumed to be unordered and is searched sequentially, starting at the last entry of the table.

PACKED=symb-addr
is the address to which control is to be passed if the field consists entirely of packed decimal characters, that is, of half-bytes with hexadecimal values in the range 0 through 9, except for the rightmost half-byte, which must contain a hexadecimal value in the range A through F.

RANGE=YES
is an optional operand indicating that, if no field compared with the search argument is an exact match, an existing table entry that would be adjacent to such an entry is to be taken as the function value. When this operand is specified, ORDER must be specified; otherwise, a sequential search of the table is made, starting at the last entry.

SUBST=
specifies a value to be stored in TARGET if no entry matching the search argument is found in the argument table.

symb-addr
is the address of a field that contains the value to be stored.

'literal value'
is the value to be stored; single quotation marks must enclose the value in this specification but are not stored as part of the data.

If this operand is specified, the TARGET operand must be specified, and the NOMATCH operand cannot be specified.

TARGET=symb-addr
is the symbolic address of the field in which the built-in function value is to be returned to the application program. The address of the function value is placed in TCATSA5, a fullword field, regardless of whether TARGET is specified.

PART 7. ERROR HANDLING AND DEBUGGING

CHAPTER 7.1. INTRODUCTION TO ERROR HANDLING AND DEBUGGING

A number of aids to testing, monitoring, and debugging are provided by CICS, as follows:

- Sequential Terminal Support - provides a method in which sequential devices, such as a card reader or disk unit, can be made to simulate the online interactive terminals or subsystems of a CICS network. This enables early testing to be carried out without the need for remote terminals or subsystems in the network to be active. Sequential terminal support is described in "Chapter 7.2. Sequential Terminal Support" on page 293.

- Trace Management - provides a trace table containing entries that reflect the execution of CICS macros by user-written application programs and by CICS management programs. The trace entries can also be stored in auxiliary storage on a sequential data set through the CICS auxiliary trace facility. The trace table and the DFHTR macro used to control its contents are described in "Chapter 7.3. Trace Control (DFHTR Macro)" on page 295.

- Dump Management - provides a dump of main storage that can be analyzed to locate errors in application programs or in CICS. Areas of main storage can be dumped onto a sequential data set, either tape or disk, for subsequent offline formatting and printing by a CICS utility program. The types of dumps and the DFHDC macro that produces them are described in "Chapter 7.4. Dump Control (DFHDC Macro)" on page 299.

- Journal Management - provides a journal or log of the realtime activity that occurs during the execution of the CICS system. This journal is stored in sequential data sets and the information it contains is essential for the reconstruction of that realtime activity. The contents of a journal, and the DFHJC macro used to control these contents, are described in "Chapter 7.5. Journal Control (DFHJC Macro)" on page 305.

- Recovery/Restart (Syncpoint) Management - provides for the emergency restart of CICS after it has terminated abnormally and also allows for erroneous operations to be backed out. The setting up of the syncpoints and the DFHSP macro used to do this are described in "Chapter 7.6. Recovery/Restart (Sync Point) Control (DFHSP Macro)" on page 319.

CHAPTER 7.2. SEQUENTIAL TERMINAL SUPPORT

Even at the simplest level of program testing, the implementer faces problems. It is not efficient to test a program from a terminal if all test data must be keyed into the system from that terminal for each test shot. The programmer cannot easily retain a backlog of proven test data and quickly test programs through the key-driven terminal as changes are made. There is also the risk that a fault developing in a test procedure being used in an operational system could affect the whole system.

CICS allows the application programmer to begin testing programs without the use of a telecommunication device. It is possible to specify through the terminal control table that sequential devices be used as terminals. These sequential devices may be card readers, line printers, disk units, or magnetic tape units. In fact, a terminal control table can include combinations of sequential devices such as: card reader and line printer, one or more disk or tape data sets as input, one or more disk or tape data sets as output. A table that contains references to these card-reader-in/line-printer-out (CRLP) terminals can also include references to other terminals on the system.

The input data submitted from a sequential device must be prepared in the form that it would come from a telecommunications device. A one- to four-character transaction identification only, or if data is included, a one- to four-character transaction identification (followed by a system-defined transaction code delimiter or a blank if less than four) must appear in the first one to four positions of the first input for a transaction. 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 at system generation, 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 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.

At some point in testing, it is necessary to use telecommunication devices to ensure that the transaction formats are satisfactory, the terminal operational approach is satisfactory, and 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 proven 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 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 which all transactions, data set activity, and output activity from the system should be measured closely.

Requests for input or output from a sequential terminal are specified in terminal control macros (DFHTC), just as other requests for input/output operations.

In response to a DFHTC TYPE=READ, 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 of the following occurs:

- An end-of-data indicator is detected in the input stream. (The indicator must be defined by the user at system generation time.)
- 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 READ is considered complete. Any subsequent READ is treated as a system error, which is passed to the user-installation terminal error program (DFHTEP) with a response

code of 4. (Under CICS/DOS/VS, EOF applies to a card reader only.)

In response to a DFHTC TYPE=WRITE from a CRLP terminal, multiple 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 found in TCTTELPL, a field in the TCTTE), 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. Writing of output continues until the end of the terminal input/output area (TIOA) is reached.

For additional information about the DFHTC macro, see "Chapter 4.2. Terminal Control (DFHTC Macro)" on page 105.

CHAPTER 7.3. TRACE CONTROL (DFHTR MACRO)

The CICS trace facility is a debugging aid for application programmers and IBM field engineers. It maintains in main storage a trace table consisting of standard CICS entries and entries defined by the user. The table is filled in a wraparound manner: when it is full, subsequent entries begin to overwrite the entries at the beginning of the table.

Tracing can be activated and deactivated by the DFHTR macro in an application program or by the master terminal transaction CEMT (or CSMT). The macro can also be used to specify the events to be recorded in the table.

The trace entries can also be stored in auxiliary storage on a sequential data set, as well as recorded in the trace table, by the CICS auxiliary trace facility, which is activated and deactivated only by the the master terminal transaction CEMT (or CSMT). The auxiliary trace data set does not wrap around: all entries are preserved so that a complete history is obtained. The CICS trace utility program (DFHTUP), the use of which is described in the appropriate <u>CICS Installation and Operations Guide</u>, can be used to print the contents of the auxiliary trace data set or selected entries from it.

Standard entries can be recorded in the trace table each time one of the following macros is issued by an application program or by a CICS management or service program. This list does not cover all entries; refer to the appropriate <u>CICS Problem Determination Guide</u> for further details.

- DFHKC (Task Control)
- DFHSC (Storage Control)
- DFHPC (Program Control)
- DFHIC (Interval Control)
- DFHDC (Dump Control)
- DFHFC (File Control)
- DFHTD (Transient Data Control)
- DFHTS (Temporary Storage Control)
- DFHJC (Journal Control)
- DFHBMS (Basic Mapping Support)
- DFHBIF (Built-In Functions)
- DFHTC (Terminal Control for VTAM-supported terminals only)
- DFHSP (Sync Point Program)
- DFHDI (Data Interchange Program)
- CICS-DL/I Interface (OS only).

In addition to the standard entries, entries produced by the terminal control program (DFHTCP) for non-VTAM terminals can be recorded in the trace table. These entries, termed field engineering (FE) entries, are normally inhibited but can be activated by the DFHTR macro.

A third class of entry, the user entry, can be defined by the application programmer with the DFHTR macro.

Trace control is branched to by the requesting program and executes as a service routine under the TCA of the requesting program. Registers are saved and restored. Return after the requested service has been performed is to the next sequential instruction in the requesting program.

TRACE TABLE

The CICS trace table, which is built during system initialization, consists of a trace header and a number of fixed-length entries that can be used to trace the flow of transactions through the system. The number of trace table entries is specified in the TRACE parameter of the DFHSIT system macro, or as a startup override.

The trace table can be initially disabled by specifying OFF in the TRACE parameter. The master terminal can be used to turn trace or auxiliary trace on or off during CICS execution. Note that auxiliary trace entries are recorded only when main storage trace is also active.

Each entry in the trace table is 32 bytes in length and aligned on a 32-byte boundary. The table is used in a wraparound manner so that when the last entry is used, the next entry is placed at the beginning of the table. The trace header contains pointers to the last-used entry, and also to the first and last entries in the trace table. The address of the trace header is in field CSATRTBA in the CSA.

The contents of trace table entries for CICS programs are fully described in the appropriate <u>CICS Problem Determination Guide</u>.

For system trace entries only, if consecutive, identical entries are generated, the first entry only is entered into the table. In these cases, a special trace control entry with trace identification X'FD' is created, and a count of the number of times the previous entry is repeated is stored therein. Trace control entries with trace identification X'FE' or X'FF' indicate the turning on or turning off of the trace facility, respectively.

The contents of any fields characterized as "Not used" in the descriptions should be ignored during the analysis of a trace table entry.

Trace Identification

Each standard entry contains a trace identification unique to the functional area concerned, together with information to aid the application programmer in determining where the macro was issued and what type of request was made to the management program.

The application programmer can make direct, nonstandard entries in the trace table by using the DFHTR macro. A trace identification number from 0 through 199 (X'00' through X'C7') and accompanying data is assigned for each trace entry. By defining several unique trace entries, the programmer can trace the logical path through a particular application or group of application programs.

CONTROLLING THE TRACE

The trace control macro DFHTR is used to activate and deactivate tracing, and to insert user-defined entries in the trace table. The trace can be activated and deactivated for the entire CICS system or for the issuing task alone.

The tracing facility can be controlled at various levels, as follows:

1. Master trace control
2. System, FE, or user control
    a. Within system, class control
    b. With user, single task control.

To make a user trace entry, the master trace control flag must be on, together with either the user control flag on or the single task trace flag on. The first is accomplished by issuing

```
  DFHTR TYPE=ON,STYPE=USER
```

the second is done by

```
  DFHTR TYPE=ON,STYPE=SINGLE.
```

To activate all tracing functions, issue

```
  DFHTR TYPE=ON,STYPE=ALL
```

To activate system trace entries only, issue

```
  DFHTR TYPE=ON,STYPE=SYSTEM
```

followed by

```
  DFHTR TYPE=ON,STYPE=(appropriate
                        class name)
```

or use

```
  DFHTR TYPE=ON,STYPE=ALL as above.
```

Tracing of each event specified in a DFHTR TYPE=ON macro continues until terminated by a DFHTR TYPE=OFF macro. The STYPE operand in the DFHTR TYPE=OFF macro specifies which events are no longer to be logged.

The STYPE operand of the DFHTR macro is used to specify whether the macro applies to the entire CICS system or only to the issuing task (SYSTEM and SINGLE parameters).

The application programmer can use the DFHTR TYPE=ENTRY macro to place his own entries in the trace table.

The following example illustrates how to 1, activate the trace facility for the issuing task and then 2, to initiate tracing for all classes of event except FE. It also shows 3, how to suppress tracing of user entries and finally 4, to deactivate the trace facility.

```
1. DFHTR TYPE=ON,STYPE=SINGLE
   .
   .
   .
2. DFHTR TYPE=ON,STYPE=ALL
   .
   .
   .
3. DFHTR TYPE=OFF,STYPE=USER
   .
   .
   .
4. DFHTR TYPE=OFF,STYPE=SINGLE
```

INITIATE TRACE (TYPE=ON)

The format of the DFHTR macro to start logging entries into the trace table is as follows:

```
DFHTR TYPE=ON
      [,STYPE={SINGLE|ALL|
        (system symbol[,sys...])|
              SYSTEM|USER|FE}]
```

TERMINATE TRACE (TYPE=OFF)

The format of the DFHTR macro to stop logging entries into the trace table is as follows:

```
DFHTR TYPE=OFF
      [,STYPE={SINGLE|ALL|
        (system symbol[,sys...])|
              SYSTEM|USER|FE}]
```

SELECTED ENTRY TRACE (TYPE=ENTRY)

The format of the DFHTR macro to cause a given entry to be logged is as follows:

```
DFHTR TYPE=ENTRY
      [,STYPE={SYSTEM|USER|FE}]
      ,ID=number
      [,DATA1={symbol|(symbol)}]
      [,RDATA1={register|(register)}]
      [,DATA2={symbol|(symbol)}]
      [,RDATA2={register|(register)}]
      [,DATA1TP={HBIN|FBIN|
        CHAR|PACK|POINTER}]
      [,DATA2TP={HBIN|FBIN|
        CHAR|PACK|POINTER}]
```

OPERANDS OF DFHTR MACRO

DATA1=
specifies the address of the data to be placed in the first data field (bytes 8 to 11) of the trace table entry.

symbol
is the symbolic address of the data to be placed in the first data field of the table entry.

(symbol)
is the symbolic address of an area that contains the address of the data to be placed in the first data field.

When this operand is included in a high-level language program, DATA1TP is required.

DATA1TP=
specifies the format of the data to be placed in the first data field of the trace table entry. The meanings of the keyword parameters are:

HBIN	Halfword binary COBOL: 9(4) COMP PL/I: FIXED BIN(15)
FBIN	Fullword binary COBOL: 9(8) COMP PL/I: FIXED BIN(31)
CHAR	1 through 4 characters COBOL: X(4) PL/I: CHAR(4)
PACK	1 through 4 bytes, packed decimal COBOL: 9(7) COMP-3 PL/I: FIXED DEC(7)
POINTER	PL/I pointer variable PL/I: POINTER

This operand is valid only for COBOL and PL/I programs. If omitted, the default is FBIN.

DATA2=
is similar to DATA1 except that it is used for the second data field (bytes 12 to 15) of the trace table entry.

When this operand is included in a high-level language program, DATA2TP is required.

DATA2TP=
is similar to DATA1TP except that it is used for the second data field of the trace table entry.

ID=number
specifies the trace identification number for this entry. It must be coded as a self-defining term. A number from 0 through 199 may be specified when STYPE=USER. A number from 200 through 255 may be specified when STYPE=SYSTEM or STYPE=FE, but you should refer to the appropriate CICS Problem Determination Guide for details of the trace identification numbers that are currently allocated to each functional area of CICS.

Note: A flag in the trace entry distinguishes between the three types of entry (USER, SYSTEM, and FE).

RDATA1= **(ASM programs only)**
specifies the register whose contents are to be placed in the

first data field of the trace table entry.

register
the number of the register whose contents are to be placed in the first data field.

(register)
the number of the register whose contents are the address of the data to be placed in the first data field.

RDATA2= (ASM programs only)
is similar to RDATA1 except that it is used for the second data field of the trace table entry.

STYPE=
indicates the type of entries to be logged or for which logging is to be discontinued. If this operand is omitted, USER is assumed.

SINGLE when included in the DFHTR TYPE=ON macro, specifies that the tracing of user entries is to be turned on for the task issuing the macro for the duration of the task or until turned off by a DFHTR TYPE=OFF, STYPE=SINGLE.

When included in the DFHTR TYPE=OFF macro, specifies that the tracing of user entries is to be turned off for the task issuing the macro.

ALL specifies that all tracing facilities (except FE) are to be turned on. This parameter turns on the master system trace flag and all of the individual system trace flags, in addition to performing the function of the USER parameter. It also, when used in the DFHTR TYPE=OFF macro, specifies that all tracing facilities (including Field Engineering) are to be stopped.

(system symbol1[,sys...]) specifies one or more system symbols that turn on or off appropriate system macro trace facilities. The valid system symbols are as follows:

Symbol	**Meaning**
KC	Task Control (DFHKC)
SC	Storage Control (DFHSC)
PC	Program Control (DFHPC)
IC	Interval Control (DFHIC)
DC	Dump Control (DFHDC)
FC	File Control (DFHFC)
TD	Transient Data Control (DFHTD)
TS	Temporary Storage Control (DFHTS)
JC	Journal Control (DFHJC)
BM	Basic Mapping Support (DFHBMS)
BF	Built-In Functions (DFHBIF)
TC	Terminal Control(DFHTC) (for VTAM-supported terminals only)
SP	Syncpoint Control (DFHSP)
DI	Data Interchange Control (DFHDI)
UE	User Exit Interface

For TYPE=ON, each symbol turns on a single system trace flag. Before tracing of any system macros occurs, the master system trace flag must also be turned on by means of the SYSTEM parameter. Note that two DFHTR macros must be issued to accomplish this; SYSTEM and system symbols cannot both be specified on the same macro.

SYSTEM specifies that the entry is a CICS entry. This parameter turns on or off the system master trace flag, which must be on in addition to the individual system trace flags before tracing of any system macros occurs. When used to turn off the master trace flag it does not turn off the individual system trace flags. See also the description of the system symbols above. Therefore, although all tracing activities for the system macros are suppressed, the previous pattern of activity could be reinstated by issuing a DFHTR TYPE=ON,STYPE=SYSTEM macro, without the need to issue a DFHTR TYPE=ON macro with the various system symbols defined.

USER specifies that the entry is a user entry and that when included in a DFHTR TYPE=ON macro specifies that the trace facility is to be turned on for all user entries for all active tasks; that is, causes the trace facility to begin logging user entries to the trace table for all tasks currently active in the system, and for all tasks becoming active subsequently, until the user trace facility is turned off.

FE (ASM programs only) specifies that the entry is a Field Engineering (FE) entry. This is the only parameter that will turn on the FE tracing facilities.

CHAPTER 7.4. DUMP CONTROL (DFHDC MACRO)

Dump management provides the capability of dumping specified areas of main storage onto a sequential data set, either tape or disk. This data set contains information about the user's transaction or application program, and can be subsequently formatted and printed offline (or while the dump data set is closed) using a CICS dump utility program (DFHDUP).

Requests for dump services are communicated to dump control through the DFHDC macro. A CICS snap dump can also be requested by the master terminal operator. Dump control executes at the priority of the requesting program, under control of the TCA of the requesting program saving and restoring registers from this TCA. After a requested dump service has been provided, control is returned to the next executable instruction in the requesting program.

Dump control operates as a serially reusable program resource. Only one service request is processed at a time. If additional requests for dump services are made while a dump is in progress, the tasks associated with those service requests are delayed (suspended) and placed in "hold" status until the dump is completed. Remaining dump requests are serviced on a first-in first-out basis.

The dump management macro (DFHDC) is used to request any of the following services:

- Dump main storage areas related to a transaction and its associated task (or any other main storage areas).
- Dump the following CICS control tables: program control table (PCT), processing program table (PPT), system initialization table (SIT), terminal control table (TCT), file control table (FCT), and destination control table (DCT).
- Dump transaction-oriented storage areas and CICS control tables.
- Dump selected main storage areas.
- For CICS/OS/VS only, dump DL/I control blocks.

To ensure a dump of the TIOA following a terminal control write that precedes a DFHDC macro, the application programmer must issue a SAVE and WAIT with the DFHTC TYPE=WRITE macro.

When the DFHDC macro is executed, information is stored in fields TCADCTR and TCADCDC of the common communication area (CCA) of the TCA, which is used for CICS service requests. Before doing so, however, the macro preserves the previous contents of the fields by copying TCADCTR (2 bytes) to TCACCSV1, and TCADCDC (4 bytes) to TCACCSV2. The previous contents can therefore be seen in the dump. The field TCADCDC is saved only if DMPCODE=value is specified. If DMPCODE=YES is specified, the user must preserve the contents of TCADCDC (if they are to appear in the dump) before storing the dump code in that field.

The dump control module will use the register save area, TCACCRS, of the CCA. To see the previous contents in a dump the application program must obtain 14 words of storage into which to copy the contents of TCACCRS before issuing the DFHDC macro.

Every formatted and transaction dump request will include the short symptom string, TCA, CSA, and trace table, unless one or more of these are suppressed with the SUPPR operand. The trace table will also be suppressed if the trace facility is not currently active.

DUMP TRANSACTION STORAGE (TYPE=TRANSACTION)

The format of the DFHDC macro to specify a dump is as follows:

```
DFHDC TYPE=TRANSACTION
      [,DMPCODE={value|YES}]
      [,SUPPR=([CSA][,TCA][,TRT])|ALL
```

This macro specifies a dump of all main storage areas related to a transaction and its associated task. This dump is normally used during testing and debugging user-written application programs. (CICS automatically provides this service if the related task is abnormally terminated.)

For CICS/OS/VS only, DL/I control blocks will also be dumped.

The following main storage areas can be dumped:

1. Task control area (TCA) and, if applicable, the transaction work area (TWA)

2. Common system area (CSA), including the user's portion of the CSA (CWA)

3. Trace table

4. Contents of general-purpose registers upon entry to dump control from requesting task

5. Either the terminal control table terminal entry (TCTTE) or the destination control table entry associated with the requesting task

6. All transaction storage areas chained off the TCA storage accounting field

7. All program storage areas containing user-written application programs active on behalf of the requesting task. Program storage areas will not be dumped for programs defined in the PPT as RELOAD=YES

8. Register save areas (RSAs) indicated by the RSA chain off the TCA

9. All terminal input/output areas (TIOAs) chained off the terminal control table terminal entry (TCTTE) for the terminal associated with the requesting task (if any).

Whenever the TCTTE is dumped (see 5 above), 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 BMS.

The following example illustrates the coding required to request a dump of transaction storage:

```
DFHDC TYPE=TRANSACTION,DMPCODE=D010
```

DUMP CICS STORAGE (TYPE=CICS)

The format of the DFHDC macro to specify a dump of system tables is:

```
DFHDC TYPE=CICS
     [,DMPCODE={value|YES}]
     [,SUPPR=([CSA][,TCA][,TRT])|ALL]
```

The application programmer can request a dump of PCT, PPT, TCT, FCT, and DCT by issuing the DFHDC TYPE=CICS macro. This dump is typically the first dump taken during testing in which the base of the test must be established; subsequent dumps are usually of the TRANSACTION type.

This macro specifies that PCT, PPT, SIT, TCT, FCT, and DCT are to be dumped. The TCA (and the TWA, if applicable), CSA (and CWA), and trace table are also dumped.

The following example illustrates the coding required to request a dump of PCT, PPT, SIT, TCT, FCT, DCT, CSA, TCA, and the trace table:

```
DFHDC TYPE=CICS,DMPCODE=D020
```

DUMP TRANSACTION STORAGE AND CICS STORAGE (TYPE=COMPLETE)

The format of the DFHDC macro to specify a complete dump is:

```
DFHDC TYPE=COMPLETE
     [,DMPCODE={value|YES}]
     [,SUPPR=([CSA][,TCA][,TRT])|ALL]
```

The application programmer can request a dump of both transaction/task-related storage and the PCT, PPT, SIT, TCT, FCT, and DCT by issuing the DFHDC TYPE=COMPLETE macro. For CICS/OS/VS only, DL/I control blocks will also be dumped.

To request a complete dump is sometimes appropriate during execution of a task, but this macro should not be used excessively. CICS control tables are primarily static areas; therefore, requesting one CICS dump and a number of TRANSACTION dumps is generally more efficient than requesting a comparable number of COMPLETE dumps. This macro specifies that transaction/task-related storage and the PCT, PPT, SIT, TCT, FCT, and DCT are to be dumped.

The following example illustrates the coding required to request a dump of both transaction storage and the PCT, PPT, SIT, TCT, FCT, and DCT:

```
DFHDC TYPE=COMPLETE,DMPCODE=D030
```

DUMP PARTIAL STORAGE (TYPE=PARTIAL)

The format of the DFHDC macro to specify a partial dump is:

```
DFHDC TYPE=PARTIAL
     ,LIST=([TERMINAL][,PROGRAM]
     [,TRANSACTION][,SEGMENT])
     [,DMPCODE={value|YES}]
     [,SUPPR=([CSA][,TCA][,TRT])|ALL]
```

The application programmer can request a dump of selected main storage areas related to the requesting task by issuing the DFHDC TYPE=PARTIAL macro. This type of dump can be used during testing and debugging of user-written application programs. It includes only the storage areas specified.

If SEGMENT is specified in the LIST operand, the application programmer must code two instructions that place the address of the main storage area to be dumped into TCADCSA and the length (in binary) of the area to be dumped into TCADCNB prior to execution of the DFHDC TYPE=PARTIAL macro. The maximum length that can be specified in TCADCNB is 32,767 bytes. The specified area must be a valid area, that is, storage allocated by the operating system within the CICS region/partition boundaries.

It is possible to dump several user areas rather than just one. The application programmer must construct a table of the user areas to be dumped, and their lengths, and place the address of the table in TCADCSA. Also, TCADCNB must be set to zero. Both of these actions must precede the DFHDC TYPE=PARTIAL macro. The table must consist of eight-byte entries, each entry containing a four-byte length field followed by a four-byte address field. The table should then be completed by adding an extra four-byte field containing X'FFFFFFFF'.

The following example shows how to request a PARTIAL storage dump that includes, along with all program storage areas, all transaction storage areas associated with this task:

```
DFHDC TYPE=PARTIAL,LIST=(TRANSACTION,
      PROGRAM),DMPCODE=DT3P
```

This example is applicable to assembler language, COBOL, or PL/I programs. All values passed to CICS are specified in the DFHDC macro. As noted above, when SEGMENT is specified, certain values must be stored in fields of the TCA prior to execution of the DFHDC macro. The programmer can also store the dump code in the TCA prior to execution of the macro.

The following examples show how to request a PARTIAL dump of a selected main storage area, using assembler language, COBOL, or PL/I.

ASM:

```
       ST     R5,TCADCSA           PLACE STORAGE ADDRESS IN TCA
       MVC    TCADCNB,=H'16384'    PLACE LENGTH OF AREA IN TCA
       MVC    TCADCDC,=CL4'AB12'   PLACE DUMP CODE IN TCA
       DFHDC  TYPE=PARTIAL,        REQUEST PARTIAL STORAGE DUMP        *
              LIST=SEGMENT,        DUMP AREA PREVIOUSLY SPECIFIED      *
              DMPCODE=YES          DUMP CODE PREVIOUSLY SPECIFIED
```

COBOL:

```
       MOVE DATADDR TO TCADCSA.    NOTE PLACE STRG ADDRESS IN TCA.
       MOVE 16384 TO TCADCNB.      NOTE PLACE LENGTH OF AREA IN TCA.
       MOVE 'AB12' TO TCADCDC.     NOTE PLACE DUMP CODE IN TCA.
       DFHDC TYPE=PARTIAL,         REQUEST PARTIAL STORAGE DUMP        *
             LIST=SEGMENT,         DUMP AREA PREVIOUSLY SPECIFIED      *
             DMPCODE=YES           DUMP CODE PREVIOUSLY SPECIFIED
```

PL/I:

```
       TCADCSA=ADDR(DATA);         /*PLACE STORAGE ADDRESS IN TCA*/
       TCADCNB=16384;              /*PLACE LENGTH OF AREA IN TCA*/
       TCADCDC='AB12';             /*PLACE DUMP CODE IN TCA*/
       DFHDC TYPE=PARTIAL,         REQUEST PARTIAL STORAGE DUMP        *
             LIST=SEGMENT,         DUMP AREA PREVIOUSLY SPECIFIED      *
             DMPCODE=YES           DUMP CODE PREVIOUSLY SPECIFIED
```

OPERANDS OF DFHDC MACRO

DMPCODE=
is a four-character dump code to be printed out with the requested dump to identify it; this code should be unique so that it is informative concerning the condition that caused the dump.

value
is a combination of four alphanumeric characters to be printed as the dump code.

YES
indicates that the dump code has been placed in TCADCDC.

LIST=
identifies specific areas to be dumped.

TERMINAL
indicates that all storage areas associated with the terminal are to be dumped. These storage areas are as follows:

1. Task control area (TCA) and, if applicable, the transaction work area (TWA)
2. Common system area (CSA), including the user's portion of the CSA (CWA)
3. Trace table
4. All terminal input/output areas (TIOAs) chained off the terminal control table terminal entry (TCTTE) for the terminal associated with the requesting task
5. Contents of general-purpose registers upon entry to dump control from the requesting task
6. 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 BMS.

PROGRAM
indicates that all program storage areas associated with this task are to be dumped. These storage areas include:

1. Task control area (TCA) and, if applicable, the transaction work area (TWA)
2. Common system area (CSA), including the user's portion of the CSA (CWA)
3. Trace table
4. All program storage areas containing user-written application program(s) active on behalf of the requesting task
5. Register save areas (RSAs) indicated by the RSA chain off the TCA
6. Contents of general-purpose registers upon entry to dump control from the requesting task
7. 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.

TRANSACTION
is typically used in combination with other types of PARTIAL dump requests to include all transaction storage areas associated with the task. These areas include:

1. Task control area (TCA) and, if applicable, the transaction work area (TWA)
2. Common system area (CSA), including the user's portion of the CSA (CWA)
3. Trace table
4. Contents of general-purpose registers upon entry to dump control from the requesting task
5. All transaction storage areas chained off the TCA storage accounting field
6. Either the terminal control table terminal entry (TCTTE) or the destination control table

entry associated with the requesting task

7. 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.

SEGMENT
is used to include in the PARTIAL dump any area of main storage specified. In addition to the selected area, the contents of the following storage areas are displayed:

1. Task control area (TCA) and, if applicable, the transaction work area (TWA)

2. Common system area (CSA), including the user's portion of the CSA (CWA)

3. Trace table

4. Contents of general-purpose registers upon entry to dump control from the requesting task

5. 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.

These parameters are not mutually exclusive. They can be specified in any combination and any order. The parentheses are optional when only one parameter is specified. At least one parameter is required. No storage area is dumped more than once as a result of a single DFHDC TYPE=PARTIAL request. For example, if

```
DFHDC TYPE=PARTIAL,
LIST=(TERMINAL,TRANSACTION)
```

is specified, the contents of the TCA and CSA are displayed only once.

SUPPR=
indicates that one or more CICS control tables will not be dumped. The dumps to be suppressed are determined by coding one or more of the following:

CSA	Common system area
TCA	Task control area
TRT	Trace table

These parameters are not mutually exclusive. They can be specified in any combination and any order. The parentheses are optional when only one parameter is specified. At least one parameter is required. Alternatively, all three of the above areas can be suppressed by coding SUPPR=ALL.

CHAPTER 7.5. JOURNAL CONTROL (DFHJC MACRO)

Journal management provides facilities for creating and managing special-purpose sequential data sets, composing "journals", during real-time CICS execution. Journals may contain 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").

In addition to the output services described in this chapter, journal management also provides support for:

- Operational control and disposition of volumes (see the appropriate CICS Installation and Operations Guide).
- Requests to switch volumes and/or read journal data sets during real-time CICS execution (see the appropriate CICS Resource Definition manual).

Requests for journal output services are made by issuing the journal control macro (DFHJC), either directly from a user task or from a CICS management program on behalf of a user task. Data may be directed to any journal specified in the journal control table (JCT), which defines the journals available during a particular CICS execution. The JCT may define one or more journals on tape or direct access storage. Each journal is identified by a number, in the range 2 through 99. The value 1 is reserved for a journal known as the system log.

All buffer space and other work areas needed for journal data set physical operations are acquired and managed by the journal control program (JCP). The user task supplies only the address and length of the data to be output. The data is moved to journal buffer space by JCP when building a journal record. The user task retains the use and control of the data and its CICS storage area.

Journal output requests are serviced by JCP. Journal records are built into blocks compatible with standard variable-blocked format. JCP uses the sequential access method of the host operating system to write the blocks to auxiliary storage.

Each logical journal record begins with the standard 4-byte 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 layout and 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. Normally, he obtains this information from the application system analyst or the person(s) responsible for programs for reading journal data sets. (See the appropriate CICS Resource Definition manual.)

JCP builds journal records for output requests at the priority of the requesting program, under control of the TCA of the requesting program. However, the TCA is not used to communicate requests or to save/restore registers. Instead, a separate control area called a journal control area (JCA) is used; this area must be acquired by the task before any journal output requests are issued.

If no other event is in-process to the journal, output to a journal data set is also initiated under the requestor's TCA. However, output event completion is always processed under a different TCA, that of a high-priority journal task associated with the journal. Journal tasks are activated when CICS execution begins, but are suspended when there are no output events outstanding. In a heavy load situation, where many user tasks request journal output while one output is in-process, a journal task initiates more output immediately after completion of the in-process output event.

The application programmer may specify parameter values for journal control requests in either of two ways:

- By including the parameters in operands of the DFHJC macro by which journal services are requested, or
- By coding instructions that place the parameter values in fields of the JCA prior to issuing the DFHJC macro.

The second of these methods provides greater economy, in that the parameter values can be varied to meet the logic needs of the application, but only a single DFHJC macro need be coded.

Journal output services that may be requested through the journal control macro are introduced and explained in the following paragraphs.

ACQUIRE A JOURNAL CONTROL AREA (TYPE=GETJCA)

The format of the DFHJC macro to acquire a journal control area (JCA) is as follows:

```
DFHJC TYPE=GETJCA
```

This macro specifies that an area to be used for communication between the application program and the CICS journal control program is to be acquired. The address of the JCA is returned in TCAJCAAD to the application program.

If journal output services are requested in an application program through DFHJC macros, the application programmer must provide the symbolic definition of the JCA by copying the CICS storage area map DFHJCADS. The JCA must be acquired for the task prior to any journal output requests by issuing the macro:

```
DFHJC TYPE=GETJCA
```

The JCA may be acquired separately, as shown above, in which case no other operands are needed. Alternatively, the JCA may be acquired by and with the program's first journal output request; for example:

```
DFHJC TYPE=(GETJCA,PUT)
```

If the latter approach is chosen, then it is not possible to place additional parameter values for the output request directly into the JCA prior to the request, because the JCA does not exist prior to this request. If any such request is attempted, warning messages are issued and the request is not processed.

In addition to acquiring the JCA for the task, the DFHJC TYPE=GETJCA macro establishes addressability to the area by moving the contents of the JCA address field (TCAJCAAD) to JCABAR, the base locator specified for the area. Once acquired for the task, the JCA is reused for all subsequent journal requests issued by or on behalf of the task. Data may also be placed in the JCA during a CICS request even though that request may not result in logging. Subsequent TYPE=GETJCA requests only cause JCABAR to be reloaded with the same value. The JCA may not be released by the user.

The following examples show how to acquire the journal control area (JCA) for the task.

ASM Example:

```
       COPY  DFHTCADS                    1
JCABAR EQU   10                          2
       COPY  DFHJCADS                    3
GETJCA DFHJC TYPE=GETJCA                 4
```

The numbers in the above example refer to the following notes:

1. Copy the TCA symbolic definitions
2. Assign a base register for the JCA
3. Copy the JCA symbolic definitions
4. Request the acquisition of the JCA.

COBOL Example:

```
   02 JCABAR PIC S9(8) COMP.             1
01 DFHTCADS COPY DFHTCADS.               2
01 DFHJCADS COPY DFHJCADS.               3
PROCEDURE DIVISION.
   MOVE CSACDTA TO TCACBAR.              4
GETJCA.
      DFHJC TYPE=GETJCA                  5
```

The numbers in the above example refer to the following notes:

1. Define the base locator for the JCA
2. Copy the TCA symbolic definitions
3. Copy the JCA symbolic definitions
4. Load TCA base locator value
5. Request the acquisition of the JCA.

PL/I Example:

```
%INCLUDE DFHTCADS;                       1
%INCLUDE DFHJCADS;                       2
GETJCA:
      DFHJC TYPE=GETJCA                  3
```

The numbers in the above example refer to the following notes:

1. Copy the TCA symbolic definitions
2. Copy the JCA symbolic definitions
3. Request the acquisition of the JCA.

CREATE A JOURNAL RECORD AND WAIT FOR OUTPUT (TYPE=PUT)

The format of the DFHJC macro to create a journal record, initiate its output, and wait for completion is as follows:

```
DFHJC TYPE={PUT|(WRITE,WAIT)}
       ,JFILEID={nn|SYSTEM|YES}
       ,JTYPEID={nnnn|YES}
       ,JCDADDR={symb-addr|YES}
       ,JCDLGTH={decimal value|YES}
      [,PFXADDR={symb-addr|YES}]
      [,PFXLGTH={decimal value|YES}]
      [,STARTIO={YES|NO}]
      [,NORESP=symb-addr]
      [,IDERROR=symb-addr]
      [,LERROR=symb-addr]
      [,IOERROR=symb-addr]
      [,NOTOPEN=symb-addr]
      [,INVREQ=symb-addr]
      [,STATERR=symb-addr]
```

This macro specifies that a journal record is to be created in the journal buffer area and then written out; the requesting task will wait until the physical record has been written. TYPE=(WRITE,WAIT) implies, and is equivalent to, TYPE=PUT.

Because the maximum buffer length that can be used to write a journal record is 32767 bytes, the combined length specified by JCDLGTH and PFXLGTH (or stored in JCALDATA and JCALPRFX,
| respectively) cannot exceed 32688.

The STARTIO=YES operand specifies that the journal record is to be written out immediately. This is the default for all requests other than WRITE without WAIT (that is, an asynchronous WRITE). If STARTIO=NO is specified, initiation of output will be delayed until the journal buffer is full to the shift-up point, output is initiated by another request to the same journal, or one second elapses.

Use of this macro ensures that the journal record is written on the auxiliary storage device associated with the journal before processing continues; the task is said to be "synchronized" with the output event. Most CICS-provided data output service is performed in a synchronous manner.

The application programmer may request synchronous journal output services either by a DFHJC TYPE=PUT macro as above, or by specifying DFHJC TYPE=(WRITE,WAIT). In both cases, certain additional keyword operands are mandatory. These keywords are JFILEID (the journal to receive data), JCDADDR (the address of the user data to be included in the journal record), JCDLGTH (the length of the user data), and JTYPEID (the 2-byte user-specified hexadecimal identifier for the journal record). Optional accompanying keywords are PFXADDR (the address of user prefix data for inclusion in the journal record) and PFXLGTH (the length of the user prefix data); the application programmer may also include keyword operands to direct control to exception-handling routines in the program. See "Test Response to a Request for Journal Services (TYPE=CHECK)" on page 315.

The following examples show how to request and wait for journal output service.

ASM Example:

```
        COPY  DFHTCADS                  1
JCABAR  EQU   10                        2
        COPY  DFHJCADS                  3
FWACBAR EQU   9                         4
        COPY  DFHFWADS                  5
RECORD  DS    0CL90
KEYDATA DS    0CL8
ACCNTNO DS    PL4
AMOUNT  DS    PL4
NAME    DS    CL20
ADDRESS DS    CL40
        DFHJC TYPE=PUT,                 6
              JFILEID=2,
              JCDADDR=KEYDATA,
              JCDLGTH=8,
              JTYPEID=0F01,
              NORESP=OK
OK      DS    0H
```

The numbers in the above example refer to the following notes:

1. Copy the TCA symbolic definitions
2. Assign a base register for the JCA
3. Copy the JCA symbolic definitions
4. Assign a base register for the FWA
5. Copy the FWA symbolic definitions
6. The DFHJC macro requests synchronous output to journal ID 2, of the 'key' data of length 8 bytes, where 0F01 is the ID for the journal record, and OK is the branch address for a normal response.

```
COBOL Example:

   02 JCABAR PIC S9(8) COMP.        1
01 DFHTCADS COPY DFHTCADS.          2
01 DFHJCADS COPY DFHJCADS.          3
01 DFHFWADS COPY DFHFWADS.          4
  02 RECORD.
    03 KEYDATA.
      04 ACCNTNO PIC S9(7) COMP-3.
      04 AMOUNT PIC S9(7) COMP-3.
    03 NAME PIC X(20).
    03 ADDRESS PIC X(40).
PROCEDURE DIVISION.
   MOVE CSACDTA TO TCACBAR.         5
      DFHJC TYPE=PUT,               6
            JFILEID=2,
            JCDADDR=KEYDATA,
            JCDLGTH=8,
            JTYPEID=0F01,
            NORESP=OK
OK.
```

The numbers in the above example refer to the following notes:

1. Define the base locator for the JCA
2. Copy the TCA symbolic definitions
3. Copy the JCA symbolic definitions
4. Copy the FWA symbolic definitions
5. Load the TCA base locator value
6. The DFHJC macro requests synchronous output to journal ID 2, of the 'key' data of length 8 bytes, where 0F01 is the ID for the journal record, and OK is the branch address for a normal response.

```
PL/I Example:

%INCLUDE DFHTCADS;                  1
%INCLUDE DFHJCADS;                  2
%INCLUDE DFHFWADS;                  3
 02 RECORD,
   03 KEYDATA,                      4
     04 ACCNTNO FIXED DECIMAL (7),
     04 AMOUNT FIXED DECIMAL (7),
   03 NAME CHAR (20),
   03 ADDRESS CHAR (40),

DFHJC TYPE=PUT,                     5
      JFILEID=2,
      JCDADDR=KEYDATA,
      JCDLGTH=8,
      JTYPEID=0F01,
      NORESP=OK
OK:
```

The numbers in the above example refer to the following notes:

1. Copy the TCA symbolic definitions
2. Copy the JCA symbolic definitions
3. Copy the FWA symbolic definitions
4. This is an 8-byte minor structure
5. The DFHJC macro requests synchronous output to journal ID 2, of the 'key' data of length 8 bytes, where 0F01 is the ID for the journal record, and OK is the branch address for a normal response.

CREATE A JOURNAL RECORD (TYPE=WRITE)

The general format of the DFHJC macro to create a journal record for subsequent output is as follows:

```
DFHJC TYPE=WRITE
       ,JFILEID={nn|SYSTEM|YES}
       ,JTYPEID={nnnn|YES}
       ,JCDADDR={symb-addr|YES}
       ,JCDLGTH={decimal value|YES}
      [,PFXADDR={symb-addr|YES}]
      [,PFXLGTH={decimal value|YES}]
      [,STARTIO={YES|NO}]
      [,COND={(YES,symb-addr)|NO}]
      [,NORESP=symb-addr]
      [,IDERROR=symb-addr]
      [,LERROR=symb-addr]
      [,NOTOPEN=symb-addr]
      [,INVREQ=symb-addr]
      [,STATERR=symb-addr]
```

This macro causes a journal record to be created in the journal buffer area, but allows the requesting task to retain control and thus to continue with other processing.

At some later time, the task may wish to ensure that the journal record has been written. If the JCA is to be used for any other journal requests, that task should save the event control number (4 bytes) returned in JCAECN after a journal record is successfully created in response to the DFHJC TYPE=WRITE request. The event control number must be restored to the JCA immediately before the DFHJC TYPE=WAIT request used to check and wait for output. If the JCA is not used in the interim for any other journal requests for the task, there is no need to save and restore the event control number.

However, restoring the event control number prior to issuing a DFHJC TYPE=WAIT macro is a good programming practice. CICS management modules also use the JCA of the task for journal requests. For example, automatic journaling is used in the file control program, and logging can be performed for recovery purposes at the user's option.

Additional keyword operands applicable to TYPE=WRITE requests are as described above under "Create a Journal Record and Wait for Output."

The basic process of building journal records in the buffer space of a given journal continues until such time as one of the following situations occurs:

- A request is made for synchronous output of a journal record.
- A request is rejected because of insufficient journal buffer space.
- The available buffer space is reduced below a user-specified level (see the appropriate CICS Resource Definition manual).

At that time, all journal records present in the buffer, including any "deferred" output resulting from asynchronous requests, are written to external storage, as one block.

If a task creates deferred output and delays synchronizing, the deferred output may be written "for free" along with other requests; when the task attempts to synchronize, there will be no need for it to wait. The advantages that may be gained by deferring journal output are:

1. Transactions may get better response times by waiting less
2. The load of physical I/O requests on the host system may be reduced
3. Journal data sets may contain fewer but larger blocks and so better utilize external 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 which 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 keyword governs whether output is to be initiated (YES) or not (NO). The default option is NO for WRITE requests and YES for PUT, (WRITE,WAIT), or WAIT requests. NO should be used whenever possible because, if every journal request uses STARTIO=YES, no improvement over synchronous output requests, in terms of reducing the number of physical I/O operations and increasing the average block size, is possible.

The COND keyword governs what happens if the journal buffer space available at the time is not sufficient to contain the journal record for the request. If the default option COND=NO is taken, the requesting task loses control. The contents of the current buffer are written out, and the journal record for this request 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, then COND=(YES,symbolic address) should be specified. If buffer space at that moment is insufficient, no journal record is built for the request, and control is returned directly to the requesting program at the location identified by symbolic address. The requesting program can perform any housekeeping needed before reissuing the journal output request.

The following example shows how to request deferred journal output, but ensure that the requesting task retains control to perform housekeeping, if necessary.

ASM:

```
         COPY  DFHCSADS              COPY CSA SYMBOLIC DEFINITIONS
COMDATA  DS    CL10                  AND COMMON WORK AREA
         .
         .
         .
         COPY  DFHTCADS              COPY TCA SYMBOLIC DEFINITIONS
SAVEDATA DS    CL10                  SAVE AREA FOR COMMON DATA
MYDATA   DS    CL10                  AREA FOR MY DATA
         .
         .
         .
JCABAR   EQU   10                    ASSIGN BASE REGISTER FOR JCA
         COPY  DFHJCADS              COPY JCA SYMBOLIC DEFINITIONS
         .
         .
         .
         MVC   SAVEDATA,COMDATA      SAVE COMMON DATA
         MVC   COMDATA,MYDATA        REPLACE WITH MY DATA FOR WORKING
         .
         .
         .
         DFHJC TYPE=WRITE,           REQUEST ASYNCHRONOUS OUTPUT       *
               JCDADDR=COMDATA,      OF COMMON DATA AREA,              *
               JCDLGTH=10,           LENGTH=10 BYTES,                  *
               JFILEID=SYSTEM,       TO SYSTEM LOG.                    *
               JTYPEID=0101,         (IDENTIFIER FOR JOURNAL RECORD)   *
               STARTIO=NO,           REQUEST DEFERRED OUTPUT,          *
               COND=(YES,RETRY),     BUT RETAIN CONTROL IF BUFFER FULL. *
               NORESP=OK             BRANCH ADDR FOR GOOD RESPONSE
         .
         .
         .
OK       DS    0H
         .
         .
         .
RETRY    DS    0H                    HOUSEKEEPING:
         MVC   MYDATA,COMDATA        MOVE DATA, THEN
         MVC   COMDATA,SAVEDATA      RESTORE COMMON DATA.
         DFHJC TYPE=WRITE,           REQUEST ASYNCHRONOUS OUTPUT       *
               JCDADDR=MYDATA,       OF DATA,                          *
               JCDLGTH=10,           LENGTH=10 BYTES,                  *
               JFILEID=SYSTEM,       TO SYSTEM LOG.                    *
               JTYPEID=0101,         (IDENTIFIER FOR JOURNAL RECORD)   *
               COND=NO,              IF BUFFER FULL, WE'LL WAIT.       *
               STARTIO=NO,           DEFER OUTPUT.                     *
               NORESP=OK             BRANCH ADDR FOR GOOD RESPONSE
```

```
COBOL:

   02 JCABAR PIC S9(8) COMP.
      .                                NOTE DEFINE BASE LOCATOR FOR JCA.
      .
      .
01 DFHCSADS COPY DFHCSADS.             NOTE COPY CSA SYMBOLIC DEFINITIONS.
   02 COMDATA PIC X(10).               NOTE DEFINE COMMON DATA AREA.
      .
      .
      .
01 DFHTCADS COPY DFHTCADS.             NOTE COPY TCA SYMBOLIC DEFINITIONS.
   02 SAVEDATA PIC X(10).              NOTE SAVE AREA FOR COMMON DATA.
   02 MYDATA PIC X(10).                NOTE AREA FOR MY DATA.
      .
      .
      .
01 DFHJCADS COPY DFHJCADS.             NOTE COPY JCA SYMBOLIC DEFINITIONS.
      .
      .
      .
PROCEDURE DIVISION.
   MOVE CSACDTA TO TCACBAR.            NOTE LOAD TCA BASE LOCATOR VALUE.
      .
      .
      .
   MOVE COMDATA TO SAVEDATA.           NOTE SAVE COMMON DATA.
   MOVE MYDATA TO COMDATA.             NOTE REPLACE WITH MY DATA FOR WORKING.
      .
      .
      .
      DFHJC TYPE=WRITE,                REQUEST ASYNCHRONOUS OUTPUT         *
            JCDADDR=COMDATA,           OF COMMON DATA AREA,                *
            JCDLGTH=10,                LENGTH=10 BYTES,                    *
            JFILEID=SYSTEM,            TO SYSTEM LOG.                      *
            JTYPEID=0101,              (IDENTIFIER FOR JOURNAL RECORD)     *
            STARTIO=NO,                REQUEST DEFERRED OUTPUT,            *
            COND=(YES,RETRY),          BUT RETAIN CONTROL IF BUFFER FULL.  *
            NORESP=OK                  BRANCH ADDR FOR GOOD RESPONSE
      .
      .
      .
OK.
      .
      .
RETRY.                                 NOTE DO HOUSEKEEPING, THEN RETRY.
   MOVE COMDATA TO MYDATA.             NOTE MOVE DATA, THEN.
   MOVE SAVEDATA TO COMDATA.           NOTE RESTORE COMMON DATA.
      DFHJC TYPE=WRITE,                REQUEST ASYNCHRONOUS OUTPUT         *
            JCDADDR=MYDATA,            OF MY DATA,                         *
            JCDLGTH=10,                LENGTH=10 BYTES,                    *
            JFILEID=SYSTEM,            TO SYSTEM LOG.                      *
            JTYPEID=0101,              (IDENTIFIER FOR JOURNAL RECORD)     *
            STARTIO=NO,                REQUEST DEFERRED OUTPUT,            *
            COND=NO,                   BUT IF BUFFER FULL, WE CAN WAIT.    *
            NORESP=OK                  BRANCH ADDR FOR GOOD RESPONSE
```

```
PL/I:

%INCLUDE DFHCSADS;                 /*COPY CSA SYMBOLIC DEFINITIONS*/
DCL 01 DFHCSAWK BASED(CSACBAR)     /*AND COMMON WORK AREA*/
       02 FILL CHAR (512),
       02 COMDATA CHAR (10);
       .
       .
       .
%INCLUDE DFHTCADS;                 /*COPY TCA SYMBOLIC DEFINITIONS*/
   02 SAVEDATA CHAR (10),          /*SAVE AREA FOR COMMON DATA*/
   02 MYDATA CHAR (10),            /*AREA FOR MY DATA*/
       .
       .
       .
%INCLUDE DFHJCADS;                 /*COPY JCA SYMBOLIC DEFINITIONS*/
       .
       .
       .
SAVEDATA=COMDATA;                  /*SAVE COMMON DATA*/
COMDATA=MYDATA;                    /*REPLACE WITH MY DATA FOR WORKING*/
       .
       .
       .
      DFHJC TYPE=WRITE,            REQUEST ASYNCHRONOUS OUTPUT         *
            JCDADDR=COMDATA,       OF COMMON DATA AREA,                *
            JCDLGTH=10,            LENGTH=10 BYTES,                    *
            JFILEID=SYSTEM,        TO SYSTEM LOG.                      *
            JTYPEID=0101,          (IDENTIFIER FOR JOURNAL RECORD)     *
            STARTIO=NO,            REQUEST DEFERRED OUTPUT,            *
            COND=(YES,RETRY),      BUT RETAIN CONTROL IF BUFFER FULL.  *
            NORESP=OK              BRANCH ADDR FOR GOOD RESPONSE
       .
       .
       .
OK:
       .
       .
       .
RETRY:                             /*HOUSEKEEPING:*/
MYDATA=COMDATA;                    /*MOVE DATA, THEN*/
COMDATA=SAVEDATA;                  /*RESTORE COMMON DATA.*/
      DFHJC TYPE=WRITE,            REQUEST ASYNCHRONOUS OUTPUT         *
            JCDADDR=MYDATA,        OF MY DATA,                         *
            JCDLGTH=10,            LENGTH=10 BYTES,                    *
            JFILEID=SYSTEM,        TO SYSTEM LOG.                      *
            JTYPEID=0101,          (IDENTIFIER FOR JOURNAL RECORD)     *
            STARTIO=NO,            REQUEST DEFERRED OUTPUT,            *
            COND=NO,               BUT IF BUFFER FULL, WE CAN WAIT.    *
            NORESP=OK              BRANCH ADDR FOR GOOD RESPONSE
```

__WAIT FOR OUTPUT OF A JOURNAL RECORD (TYPE=WAIT)__

The general format of the DFHJC macro to wait for output of a previously created journal record is as follows:

```
DFHJC TYPE=WAIT
      ,JFILEID={nn|SYSTEM|YES}
      [,STARTIO={YES|NO}]
      [,NORESP=symb-addr]
      [,IDERROR=symb-addr]
      [,IOERROR=symb-addr]
      [,NOTOPEN=symb-addr]
      [,INVREQ=symb-addr]
```

This macro specifies that the requesting task is to be placed in a wait state until the block containing a journal record has been written as output (that is, the journal operation is to be synchronized with continued execution of the task issuing the journal write request). If the block containing the journal record has not been written, the requesting task is placed in a wait state until the write is completed.

Before issuing a synchronizing request, the task must ensure that the event control number (4 bytes) corresponding to the journal record in question is in field JCAECN of the JCA. An event control number is returned in JCAECN after every successful journal output request. Since the JCA is used for every journal request issued by the task (or by CICS on its behalf), the requesting program must save the event control number immediately after an asynchronous output request if it is to be used later. This is necessary because the particular event control number may be overwritten during reuse of the JCA.

If the JCA is not reused between the output request and the synchronization request, the requesting program need not save and restore the event control number. It is the user's responsibility to determine whether or not he needs to save and restore it.

If the requesting program has made a succession of successful asynchronous output requests to the same journal data set, it is only necessary to synchronize on the last of these requests to ensure that all of the journal records have reached the external storage device. This may be done either by issuing a stand-alone DFHJC TYPE=WAIT request, or by making the last output request itself synchronous, a DFHJC TYPE=PUT or TYPE=(WRITE,WAIT).

The following examples show a typical sequence of instructions to request synchronization with the output of a journal record.

ASM:

```
         COPY  DFHTCADS                COPY TCA SYMBOLIC DEFINITIONS
SAVEDECN DS    CL4                     SAVED EVENT CONTROL NUMBER
JDATA    DS    CL36                    DATA TO WRITE TO JOURNAL
         .
         .
         .
JCABAR   EQU   10                      ASSIGN BASE REGISTER FOR JCA
         COPY  DFHJCADS                COPY JCA SYMBOLIC DEFINITIONS
         .
         .
         .
         DFHJC TYPE=WRITE,             REQUEST ASYNCHRONOUS OUTPUT        *
               JCDADDR=JDATA,          OF DATA AT JDATA,                  *
               .                       ETC.                               *
               .                                                          *
               .                                                          *
               NORESP=OK1              BRANCH TO OK1 IF GOOD RESPONSE
         .
         .
OK1      DS    0H
         MVC   SAVEDECN,JCAECN         SAVE EVENT CONTROL NUMBER
         .
         .
         .
         MVC   JCAECN,SAVEDECN         RESTORE EVENT CONTROL NUMBER,
         DFHJC TYPE=WAIT,              AND SYNCHRONIZE WITH OUTPUT.       *
               NORESP=OK2              BRANCH TO OK2 IF GOOD RESPONSE
         .
         .
         .
OK2      DS    0H
         .
         .
         .
```

```
COBOL:

   02 JCABAR PIC S9(8) COMP.
                                       NOTE DEFINE BASE LOCATOR FOR JCA.
      .
      .
      .
01 DFHTCADS COPY DFHTCADS.             NOTE COPY TCA SYMBOLIC DEFINITIONS.
   02 SAVEDECN PIC X(4).               NOTE SAVED EVENT CONTROL NUMBER.
   02 JDATA PIC X(36).                 NOTE DATA TO WRITE TO JOURNAL.
      .
      .
      .
01 DFHJCADS COPY DFHJCADS.             NOTE COPY JCA SYMBOLIC DEFINITIONS.
      .
      .
      .
PROCEDURE DIVISION.
   MOVE CSACDTA TO TCACBAR.            NOTE LOAD TCA BASE LOCATOR VALUE.
      .
      .
      .
      DFHJC TYPE=WRITE,                REQUEST ASYNCHRONOUS OUTPUT         *
            JCDADDR=JDATA,             OF DATA AT JDATA,                   *
            .                          ETC.                                *
            .                                                              *
            .                                                              *
            NORESP=OK1                 BRANCH TO OK1 IF GOOD RESPONSE
      .
      .
      .
OK1.
   MOVE JCAECN TO SAVEDECN.            NOTE SAVE EVENT CONTROL NUMBER.
      .
      .
      .
   MOVE SAVEDECN TO JCAECN.            NOTE RESTORE EVENT CONTROL NUMBER.
      DFHJC TYPE=WAIT,                 AND SYNCHRONIZE WITH OUTPUT.        *
            NORESP=OK2                 BRANCH TO OK2 IF GOOD RESPONSE.
      .
      .
      .
OK2.
      .
      .
      .
```

```
PL/I:

%INCLUDE DFHTCADS;                   /*COPY TCA SYMBOLIC DEFINITIONS*/
   02 SAVEDECN CHAR (4),             /*SAVED EVENT CONTROL NUMBER*/
   02 JDATA CHAR (36);               /*DATA TO WRITE TO JOURNAL*/
       .
       .
       .
%INCLUDE DFHJCADS;                   /*COPY JCA SYMBOLIC DEFINITIONS*/
       .
       .
       .
       DFHJC TYPE=WRITE,             REQUEST ASYNCHRONOUS OUTPUT         *
             JCDADDR=JDATA,          OF DATA AT JDATA,                   *
             .                       ETC.                                *
             .                                                           *
             .                                                           *
             NORESP=OK1              BRANCH TO OK1 IF GOOD RESPONSE
       .
       .
       .
OK1:
SAVEDECN=JCAECN;                     /*SAVE EVENT CONTROL NUMBER*/
       .
       .
       .
JCAECN=SAVEDECN;                     /*RESTORE EVENT CONTROL NUMBER,*/
       DFHJC TYPE=WAIT,              AND SYNCHRONIZE WITH OUTPUT.        *
             NORESP=OK2              BRANCH TO OK2 IF GOOD RESPONSE
       .
       .
       .
OK2:
       .
       .
       .
```

TEST RESPONSE TO A REQUEST FOR JOURNAL SERVICES (TYPE=CHECK)

The general format of the DFHJC macro to check the CICS response to a request for journal services is as follows:

```
DFHJC TYPE=CHECK
      [,NORESP=symb-addr]
      [,IDERROR=symb-addr]
      [,LERROR=symb-addr]
      [,IOERROR=symb-addr]
      [,NOTOPEN=symb-addr]
      [,INVREQ=symb-addr]
      [,STATERR=symb-addr]
|     [,VOLERR=symb-addr]
|     [,ERROR=symb-addr]
```

JOURNAL CONTROL RESPONSE CODES

To test a response code the application programmer must know the actual settings of the response code, which is returned at JCAJCRC. The possible response codes and the conditions to which they correspond are as follows:

Condition	ASM	COBOL	PL/I
NORESP	X'00'	LOW-VALUES (JCARCNR)	00000000
IDERROR	X'01'	12-1-9 (JCARCIDE)	00000001
INVREQ	X'02'	12-2-9 (JCARCIRE)	00000010
STATERR	X'03'	12-3-9 (JCARCSE)	00000011
VOLERR	X'04'	12-4-9 (JCARCVE)	00000100
NOTOPEN	X'05'	12-5-9 (JCARCNOE)	00000101
LERROR	X'06'	12-6-9 (JCARCLE)	00000110
IOERROR	X'07'	12-7-9 (JCARCIOE)	00000111
ERROR	anything not equal to NORESP		

Note: The names enclosed in parentheses in the COBOL column indicate the condition names generated by CICS. These names may be used in testing for the conditions in a COBOL program.

If the application programmer does not provide for checking a particular response code and the corresponding condition occurs, program execution resumes at the instruction immediately following the DFHJC macro which requested the journal service.

OPERANDS OF DFHJC MACRO

COND=
specifies that control is to be returned to the application program if the request cannot be satisfied immediately because insufficient journal buffer space is available. If control is to be returned, the point of return must be specified as a second parameter of this operand.

(YES,symb-addr)
indicates that control is to be returned to the location represented by symbolic address in the application program if the request cannot be satisfied immediately. No journal record will have been created for the request.

NO
indicates that the contents of the current buffer are to be written out and the requesting task placed in a wait state until its request has been satisfied (by the building of a record in buffer space freed by the write operation).

ERROR=symb-addr
is the address to which control is to be returned if any of the response conditions other than NORESP occurs.

IDERROR=symb-addr
is the address to which control is to be returned if the specified journal file identification does not exist in the journal control table (JCT).

INVREQ=symb-addr
is the address to which control is to be returned if the TYPE operand is invalid.

IOERROR=symb-addr
is the address to which control is to be returned if the physical output of a journal record failed because of an unrecoverable I/O error. This operand is applicable only to requests that may cause a wait for completion of output, that is, to TYPE=PUT, TYPE=(WRITE,WAIT), or TYPE=WAIT.

JCDADDR=
is the address of the user data to be built into the journal record.

symb-addr
is the symbolic address of the user data.

YES
indicates that the address of the user data has been placed in JCAADATA prior to issuing this macro.

JCDLGTH=
is the length of the user data to be built into the journal record.

decimal value
is a decimal numeral in the range from 1 to 32000 (or a lower maximum, because of the journal buffer size), indicating the length, in bytes, of the user data.

YES
indicates that the length, in binary, of the user data has been placed in JCALDATA prior to issuing this macro.

JFILEID=
is the one-byte identification of the journal referred to in this journal operation.

nn
is a decimal value in the range from 2 through 99 to be taken as the journal file identification.

SYSTEM
indicates that the system log data set is the journal for this operation.

YES
indicates that the journal file identification has been placed in JCAJFID prior to issuing this macro.

JTYPEID=
is an identifier to be placed in the journal record to identify its origin.

nnnn
is a 1 through 4-character hexadecimal value to be taken as the identifier for the journal record; if fewer than 4 characters are specified, padding with zeros occurs on the right.

YES
indicates that the journal record identification has been placed in JCAJRTID prior to issuing this macro.

LERROR=symb-addr
is the address to which control is to be returned if the computed length for the journal record exceeds the total buffer space allocated for the journal data set, as specified in the JCT entry for the data set.

NORESP=symb-addr
is the address to which control is to be returned if the requested operation was performed successfully.

NOTOPEN=symb-addr
is the address to which control is to be returned when the journal is closed and is not subject to exclusive control.

PFXADDR=
is the address of user prefix data to be included in the journal record.

symb-addr
is the symbolic address of the user prefix data.

YES
indicates that the address of the user prefix data has been placed in JCAAPRFX prior to issuing this macro.

PFXLGTH=
is the length of the user prefix data to be included in the journal record.

decimal value
is a decimal numeral in the range from 1 to 32000 (or a lower maximum, because of the journal buffer size), indicating the length, in bytes, of the user prefix data.

YES
indicates that the length, in binary, of the user prefix data has been placed in JCALPRFX prior to issuing this macro.

STARTIO=
specifies whether output of the journal record is to be initiated immediately.

YES
indicates that output of the journal record is to be initiated.

NO
indicates that no output operation is required at this time.

The default value is YES if a synchronizing request is issued, namely PUT, (WRITE,WAIT), or WAIT. The default is NO for a simple write request. If STARTIO=NO is specified with a synchronizing request the maximum delay allowed before output is initiated is one second.

STATERR=symb-addr
is the address to which control is passed if the current status of the journal prevents the requested operation. For example, the request is a PUT or WRITE in a task that has exclusive control of the journal. (For further details, refer to the appropriate _CICS Customization Guide_.)

VOLERR=symb-addr (OS only)
is the address to which control is to be passed if the volume is not known.

CHAPTER 7.6. RECOVERY/RESTART (SYNC POINT) CONTROL (DFHSP MACRO)

Sync point management works in conjunction with other CICS components, such as transient data management and file management, to provide the user with facilities needed for an emergency restart after an abnormal termination of CICS. In an emergency restart, changes made in protected resources (for example, in transient data intrapartition queues) can be backed out for tasks that were "in flight" at the time of failure. This backout is based upon information about the tasks recorded on a system log during execution.

Each synchronization point in an application program marks the completion of a **logical unit of work** (an LUW). By definition, an LUW is an application programmer-defined unit of work that performs a complete processing function. One task may perform one LUW, or several LUWs, generally delimited by conversational terminal operations (a terminal write, followed by a terminal read).

SPECIFY A SYNCHRONIZATION POINT (TYPE=USER)

The format of the DFHSP macro that specifies completion of a logical unit of work, or sync point, is as follows:

```
DFHSP TYPE=USER
```

A sync point is always requested by CICS at termination of a task.

The completion of a logical unit of work indicates to CICS that:

- All updates or modifications performed by the task are logically complete, and should not be backed out if a system failure occurs.
- Functions requested prior to the sync point, but deferred until the end of the logical unit of work, are to be processed, even if a subsequent system failure occurs. An example of such an operation is a purge of a transient data intrapartition queue, as requested by the application program.
- All resources protected automatically on behalf of the task up to this point are to be released. An example of such a resource may be a transient data intrapartition destination that is logically associated with the task or a resource previously enqueued by the user.
- All resources previously enqueued by the user are dequeued.

The location of a sync point for a task on the system log data set, relative to other logged activity for that task, determines the extent to which CICS (or user programs) may need to provide transaction backout. Generally, sync points are not needed for short duration tasks.

Sync points are also used by CICS to delimit the extent to which user data set modifications may need to be backed out for a task. During emergency restart, CICS collects all user data set modifications for tasks that were engaged in a LUW at the time of uncontrolled shutdown and copies them in a restart data set. The modifications can then be read by the CICS transaction backout program or by user-written programs executed during the post-initialization phase of restart.

Through these facilities, sync point management not only permits emergency restart but also provides the means by which the activity required for such restart can be controlled by the user. The functions performed by other CICS programs involved in sync point, uncontrolled shutdown, or emergency restart activities are explained in greater detail in the appropriate _CICS Facilities and Planning Guide_.

A sync point request for a task that is scheduled to use a DL/I resource implies the release of that resource. This means that if, after issuing a DFHSP TYPE=USER macro, access to a DL/I data base is required, the desired PSB must be rescheduled through the DFHFC TYPE=(DL/I,PSB) macro. The previous position of that data base has been lost. Conversely, when a DL/I termination instruction is issued, CICS will issue a DFHSP TYPE=USER instruction on behalf of the task that is releasing a PSB.

Any BMS logical message that has been started but not completed when a DFHSP macro is issued is forced to completion by means of an implied DFHBMS TYPE=PAGEOUT macro. (If a DFHBMS TYPE=PAGEOUT macro is executed during the building of a page, this could result in an incomplete page being output.)

Note: If sync points are to be issued in a transaction that is eligible for transaction restart, the application programmer must seek advice from the systems programmer.

BACKOUT RECOVERABLE RESOURCES (TYPE=ROLLBACK) (ASM ONLY)

The format of the DFHSP macro that restores recoverable resources is as follows:

```
DFHSP TYPE=ROLLBACK
```

This macro causes all changes to recoverable resources made by the task since its last sync point to be backed out so that those resources are then in the state that they were at the time the sync point was taken.

After the recoverable resources have been restored, a sync point is taken and control is passed to the user.

PART 8. APPENDIXES

APPENDIX A. EXAMPLE OF A CICS APPLICATION PROGRAM

This appendix contains an executable application program that performs a limited message switching function; that is, data collection, message entry, and message retrieval. The sample program, which is not supplied with the CICS product, is shown in assembler language, COBOL, and PL/I.

Message Switching Sample Program in ASM

```
**********************************************************************
*      A S S E M B L E R       E X A M P L E       P R O B L E M     *
**********************************************************************
*        TITLE 'CICS/VS MESSAGE SWITCHING PROGRAM EXAMPLE'           *
         DFHCOVER
**********************************************************************
* * * *              A P P L I C A T I O N   P R O G R A M    * * * *
**********************************************************************
* * *                  D U M M Y   S E C T I O N S              * * *
**********************************************************************
         COPY  DFHCSADS                   COPY COMMON SYSTEM AREA DSECT
         EJECT                            LISTING CONTROL CARD - EJECT
         COPY  DFHTCADS                   COPY TASK CONTROL AREA DSECT
TWATSRL  DS    H                          TEMPORARY STORAGE RECORD LENGTH
         DS    H
TWATDDI  DS    CL4                        DESTINATION IDENTIFICATION
TWAREAI  DS    CL4                        RETRIEVE ALL INDICATOR
TWAQEMCI DS    C                          QUEUE EMPTY MESSAGE CONTROL IND
         EJECT                            LISTING CONTROL CARD - EJECT
TCTTEAR  EQU   11                         TERM CONT TABLE TERM ENT ADR RG
         COPY  DFHTCTTE                   COPY TERM CONT TABLE TERM ENTRY
TIOABAR  EQU   10                         TERM I/O AREA BASE ADDR REG
         COPY  DFHTIOA                    COPY TERMINAL I/O AREA DSECT
TIOADATA DS    0CL80                      DATA AREA
TIOATID  DS    CL4                        TRANSACTION IDENTIFICATION
         DS    C                          DELIMITER
TIOARRI  DS    0CL6                       RESUME REQUEST IDENTIFICATION
TIOARAI1 DS    0CL3                       RETRIEVE ALL INDICATOR 1
TIOADID  DS    CL4                        DESTINATION IDENTIFICATION
TIOASSF  DS    0CL4                       SUSPEND STORAGE FACILITY IDENT
         DS    C                          DELIMITER
TIOAMBA  DS    0C                         TERMINAL MESSAGE BEGINNING ADDR
TIOARAI2 DS    CL3                        RETRIEVE ALL INDICATOR 2
**********************************************************************
         SPACE 8                          LISTING CONTROL CARD - SPACE 8
TDIABAR  EQU   9                          TRANS DATA IN AREA BASE ADDR RG
         COPY  DFHTDIA                    COPY TRANS DATA INPUT AREA
         EJECT                            LISTING CONTROL CARD - EJECT
**********************************************************************
* * * *              A P P L I C A T I O N   P R O G R A M    * * * *
**********************************************************************
CICSATP  CSECT                            CONTROL SECTION - APPL TEST PGM
         USING *,3                        USING REGISTER 3 AT *
         LR    03,14                      LOAD PROGRAM BASE REGISTER
         B     ATPIPIN                    GO TO INIT PROG INSTR ENTRY
**********************************************************************
         EJECT                            LISTING CONTROL CARD - EJECT
**********************************************************************
* * *                  D E C L A R A T I V E S                  * * *
**********************************************************************
MCPDIEM  DC    Y(MCPDEML-4)               TERMINAL MESSAGE LENGTH
         DC    H'0'
         DC    X'15'                      NEW LINE SYMBOL CONSTANT
         DC    08X'17'                    HARDCOPY TERM IDLE CHARACTERS
         DC    C'DESTINATION IDENTIFICATION ERROR - PLEASE RESUBMIT'
         DC    X'15'                      NEW LINE SYMBOL CONSTANT
MCPDEML  EQU   *-MCPDIEM                  TERMINAL MESSAGE TOTAL LENGTH
```

Message Switching Sample Program in ASM (Continued)

```
*********************************************************************
*                        D A T A   C O L L E C T I O N              *
*********************************************************************
DCPDCAML DC    Y(L'DCPDCAMD)              DATA COLL ACKNOWLEDGMENT LEN
         DC    H'0'
DCPDCAMD DC    C' DATA COLLECTION HAS BEEN REQUESTED AND IS ABOUT TO BE*
               GIN  '                     DATA COLLECTION ACKNOWLEDGMENT
DCPEODML DC    Y(L'DCPEODMD)              END OF DATA MESSAGE LENGTH
         DC    H'0'
DCPEODMD DC    C' THE DATA HAS BEEN RECEIVED AND DISPATCHED TO THE DESI*
               GNATED DESTINATION  '      END OF DATA MESSAGE
DCPEOVML DC    Y(L'DCPEOVMD)
         DC    H'0'
DCPEOVMD DC    C' END OF VOLUME REQUEST HAS BEEN RECEIVED  '
DCPSRAM  DC    Y(DCPSRAL-4)               TERMINAL MESSAGE LENGTH
         DC    H'0'
         DC    X'15'                      NEW LINE SYMBOL CONSTANT
         DC    08X'17'                    HARD COPY TERM IDLE CHARACTERS
         DC    C'DATA COLLECTION SUSPENSION HAS BEEN REQUESTED'
         DC    X'15'                      NEW LINE SYMBOL CONSTANT
DCPSRAL  EQU   *-DCPSRAM                  TERMINAL MESSAGE TOTAL LENGTH
DCPRRAM  DC    Y(DCPRRAL-4)               TERMINAL MESSAGE LENGTH
         DC    H'0'
         DC    X'15'                      NEW LINE SYMBOL CONSTANT
         DC    08X'17'                    HARDCOPY TERM IDLE CHARACTERS
         DC    C'DATA COLLECTION RESUMPTION HAS BEEN REQUESTED AND IS '
         DC    C'ABOUT TO BEGIN'
         DC    X'15'                      NEW LINE SYMBOL CONSTANT
DCPRRAL  EQU   *-DCPRRAM                  TERMINAL MESSAGE TOTAL LENGTH
*********************************************************************
         SPACE 4                          LISTING CONTROL CARD - SPACE 4
*********************************************************************
*                        M E S S A G E   E N T R Y                  *
*********************************************************************
MEPMEAML DC    Y(L'MEPMEAMD)              MSG ENTRY ACKNOWLEDGMENT LNGTH
         DC    H'0'
MEPMEAMD DC    C' YOUR MESSAGE HAS BEEN RECEIVED AND DISPATCHED TO THE *
               DESIGNATED DESTINATION  ' MESSAGE ENTRY ACKNOWLEDGMENT
*********************************************************************
         SPACE 4                          LISTING CONTROL CARD - SPACE 4
*********************************************************************
*                         M E S S A G E   R E T R I E V A L         *
*********************************************************************
MRPNMMM  DC    Y(MRPNMML-4)               TERMINAL MESSAGE LENGTH
         DC    H'0'
         DC    X'15'                      NEW LINE SYMBOL CONSTANT
         DC    08X'17'                    HARDCOPY TERM IDLE CHARACTERS
         DC    C'THERE ARE NO MORE '
         DC    C'MESSAGES QUEUED FOR THIS DESTINATION'
         DC    X'15'                      NEW LINE SYMBOL CONSTANT
MRPNMML  EQU   *-MRPNMMM                  TERMINAL MESSAGE TOTAL LENGTH
MRPNMQM  DC    Y(MRPNQML-4)               TERMINAL MESSAGE LENGTH
         DC    H'0'
         DC    X'15'                      NEW LINE SYMBOL CONSTANT
         DC    08X'17'                    HARDCOPY TERM IDLE CHARACTERS
         DC    C'THERE ARE NO MESSAGES QUEUED FOR THIS DESTINATION'
         DC    X'15'                      NEW LINE SYMBOL CONSTANT
MRPNQML  EQU   *-MRPNMQM                  TERMINAL MESSAGE TOTAL LENGTH
*********************************************************************
         EJECT                            LISTING CONTROL CARD - EJECT
```

Message Switching Sample Program in ASM (Continued)

```
***********************************************************************
* * *                   I M P E R A T I V E S                     * * *
***********************************************************************
* *                                                                 * *
***********************************************************************
         DS    0D                       STORAGE ALIGNMENT - DOUBLEWORD
         DC    CL32'MESSAGE CONTROL PROGRAM'
ATPIPIN  DS    0D                       INITIAL PROGRAM INSTRUCTION ENT
         L     TCTTEAR,TCAFCAAA         LOAD TERM CONT AREA ADDR REG
         L     TIOABAR,TCTTEDA          LOAD TERM I/O AREA ADDR REG
         CLC   =C'DSDC',TIOATID         COMPARE TRANSACTION IDENT
         BE    ALPDCPN                  GO TO DATA COLLECTION PROG IF =
         CLC   =C'DSME',TIOATID         COMPARE TRANSACTION IDENT
         BE    ALPMEPN                  GO TO MESSAGE ENTRY PROG IF =
         CLC   =C'DSMR',TIOATID         COMPARE TRANSACTION IDENT
         BE    ALPMRPN                  GO TO MESSAGE RETRIEVAL PROG
         DFHPC TYPE=ABEND,                                              *
               ABCODE=XAPT
         EJECT                          LISTING CONTROL CARD - EJECT
***********************************************************************
* *                       A P P L I C A T I O N   L O G I C         * *
***********************************************************************
* *                         D A T A   C O L L E C T I O N           * *
***********************************************************************
         DC    CL32'DATA COLLECTION PROGRAM'
***********************************************************************
ALPDCPN  DS    0H                       DATA COLLECTION PROGRAM ENTRY
         CLC   =C'RESUME',TIOARRI       COMPARE FOR RESUME REQUEST
         BNE   DCPRRBN                  GO TO RESUME REQUEST BYPASS
         MVC   TIOATDL(DCPRRAL),DCPRRAM MOVE TERMINAL MESSAGE TO OUTPUT
         MVC   TCATSDI(4),=C'DSDC'      MOVE TEMP STRG DATA IDENT
         MVC   TCATSDI+4(4),TCTTETI     MOVE TEMP STRG DATA IDENT
         DFHTS TYPE=GET,                                                *
               TSDADDR=TWATSRL,                                         *
               NORESP=DCPRRNR,                                          *
               RELEASE=YES
         DFHPC TYPE=ABEND,                                              *
               ABCODE=XDCR
DCPFEOV  EQU   *                        FORCED END OF VOLUME ROUTINE
         DFHTD TYPE=FEOV                ISSUE TRANSIENT DATA MACRO
         MVC   TIOATDL((4+L'DCPEOVMD)),DCPEOVML
         DFHTC TYPE=(WRITE)
         B     RETURN
***********************************************************************
DCPRRBN  EQU   *                        RESUME REQUEST BYPASS ENTRY
         MVC   TWATDDI,TIOADID          MOVE DESTINATION IDENTIFICATION
         MVC   TCATDDI,TWATDDI
         CLC   TIOAMBA(4),=C'FEOV'      CHECK FOR FORCED END OF VOL REQ
         BE    DCPFEOV                  BRANCH TO END OF VOLUME ROUTINE
         MVC   TIOATDL((4+L'DCPDCAMD)),DCPDCAML
DCPRRNR  EQU   *                        RESUME REQUEST NORMAL RESPONSE
         DFHTC TYPE=(WRITE)
         DFHTC TYPE=(READ)
***********************************************************************
         SPACE 4                        LISTING CONTROL CARD - SPACE 4
```

Message Switching Sample Program in ASM (Continued)

```
DCPTEWN  DS    0H                           TERMINAL EVENT WAIT ENTRY
         DFHTC TYPE=(WAIT)
         L     TIOABAR,TCTTEDA              LOAD TERM I/O AREA ADDR REG
         CLC   =C'DUMP',TIOATID
         BE    DCPDPTS                      GO TO DUMP TRANSACTION STORAGE
         CLC   =C'EOD',TIOADBA              COMP DATA FOR EOD INDICATION
         BE    DCPEXIT                      GO TO EXIT IF EQUAL
         CLC   =C'SUSPEND',TIOADBA          COMPARE FOR SUSPEND REQUEST
         BNE   DCPSRBN                      GO TO SUSPEND REQUEST BYPASS
         MVC   TWATSRL,=H'32'               MOVE TEMP STRG RECORD LENGTH
         MVC   TCATSDI(4),=C'DSDC'          MOVE TEMP STRG DATA IDENT
         MVC   TCATSDI+4(4),TCTTETI         MOVE TEMP STRG DATA IDENT
         CLC   =C'MAIN',TIOASSF
         BNE   DCPSRMB                      GO TO MAIN STRG FACILITY BYPASS
         DFHTS TYPE=PUT,                                                 *
               TSDADDR=TWATSRL,                                          *
               STORFAC=MAIN
         B     DCPSRAB                      GO TO AUX STRG FACILITY BYPASS
DCPSRMB  EQU   *                            MAIN STORAGE FACILITY BYPASS
         DFHTS TYPE=PUT,                                                 *
               TSDADDR=TWATSRL,                                          *
               STORFAC=AUXILIARY
DCPSRAB  EQU   *                            AUX STORAGE FACILITY BYPASS
         DFHTS TYPE=CHECK,                                               *
               NORESP=DCPSRNR
         DFHPC TYPE=ABEND,                                               *
               ABCODE=XDCS
DCPSRNR  EQU   *                            SUSPEND REQUEST NORMAL RESPONSE
         MVC   TIOATDL(DCPSRAL),DCPSRAM MOVE TERMINAL MESSAGE TO OUTPUT
         DFHTC TYPE=(WRITE)
         B     RETURN                       GO TO RETURN ENTRY
DCPSRBN  EQU   *                            SUSPEND REQUEST BYPASS ENTRY
         MVC   TCATDDI,TWATDDI              MOVE DESTINATION IDENTIFICATION
         XC    TCTTEDA,TCTTEDA              RESET TERMINAL DATA ADDRESS
         DFHTC TYPE=(READ)
         LH    14,TIOATDL                   LOAD TERMINAL DATA LENGTH
         LA    14,4(0,14)                   INCREMENT TERMINAL DATA LENGTH
         STH   14,TIOATDL                   STORE TERMINAL DATA LENGTH
         DFHTD TYPE=PUT,                                                 *
               TDADDR=TIOATDL,                                           *
               NORESP=DCPNRCN,                                           *
               IDERROR=DCPDIEN
         DFHPC TYPE=ABEND,                                               *
               ABCODE=XDCP
***********************************************************************
DCPNRCN  DS    0H                           NORMAL RESP CODE ENTRY ADDRESS
         ST    TIOABAR,TCASCSA              STORE TERM I/O AREA ADDRESS
         DFHSC TYPE=FREEMAIN
         B     DCPTEWN                      GO TO TERM EVENT WAIT ENTRY
***********************************************************************
         SPACE 4                            LISTING CONTROL CARD - SPACE 4
***********************************************************************
DCPDPTS  EQU   *                            DUMP TRANSACTION STOR ROUTINE
         DFHDC TYPE=TRANSACTION,DMPCODE=TRAN
         XC    TCTTEDA,TCTTEDA              CLEAR TERMINAL DATA AREA ADDR
         DFHTC TYPE=(READ)
         B     DCPNRCN                      RETURN TO MAINSTREAM LOGIC
***********************************************************************
         SPACE 4
***********************************************************************
DCPEXIT  EQU   *                            EXIT
         MVC   TIOATDL((4+L'DCPEODMD)),DCPEODML
         DFHTC TYPE=(WRITE)
         B     RETURN                       GO TO RETURN ENTRY
***********************************************************************
         EJECT                              LISTING CONTROL CARD - EJECT
```

Message Switching Sample Program in ASM (Continued)

```
***********************************************************************
*                            M E S S A G E   E N T R Y                *
***********************************************************************
         DC    CL32'MESSAGE ENTRY PROGRAM'
***********************************************************************
ALPMEPN  DS    0H                         MESSAGE ENTRY PROGRAM ENTRY
         MVC   TCATDDI,TIOADID            MOVE DESTINATION IDENTIFICATION
         MVC   TIOATID,TCTTETI            MOVE SOURCE IDENTIFICATION
         LH    14,TIOATDL                 LOAD TERMINAL DATA LENGTH
         LA    14,4(0,14)                 INCREMENT TERMINAL DATA LENGTH
         STH   14,TIOATDL                 STORE TERMINAL DATA LENGTH
         DFHTD TYPE=PUT,                                                X
               TDADDR=TIOATDL,                                          X
               NORESP=MEPNRCN,                                          X
               IDERROR=MEPDIEN
         DFHPC TYPE=ABEND,                                              X
               ABCODE=XMEP
***********************************************************************
MEPNRCN  DS    0H                         NORMAL RESP CODE ENTRY ADDRESS
         MVC   TIOATDL((4+L'MEPMEAMD)),MEPMEAML
         DFHTC TYPE=(WRITE)
         B     RETURN                     GO TO RETURN ENTRY
***********************************************************************
         EJECT                            LISTING CONTROL CARD EJECT
***********************************************************************
*                            M E S S A G E   R E T R I E V A L        *
***********************************************************************
         DC    CL32'MESSAGE RETRIEVAL PROGRAM'
***********************************************************************
         SPACE 4                          LISTING CONTROL CARD - SPACE 4
***********************************************************************
ALPMRPN  DS    0H                         MESSAGE RETRIEVAL PROGRAM ENTRY
         MVC   TWAREAI,TIOARAI2           MOVE RETRIEVE ALL INDICATOR
         MVC   TWATDDI,TCTTETI            MOVE DESTINATION IDENTIFICATION
         CLC   =C'ALL',TIOARAI1           COMPARE ALL INDICATOR FOR ALL
         BNE   MRPAI1B
         MVC   TWAREAI,TIOARAI1           MOVE RETRIEVE ALL INDICATOR
         B     MRPDEBN
MRPAI1B  DS    0H                         ALL INDICATOR 1 BYPASS
         CLC   =CL4'    ',TIOADID         COMPARE DEST IDENT TO BLANKS
         BE    MRPDEBN                    GO TO DEST ID = BL IF EQUAL
         MVC   TWATDDI,TIOADID            MOVE DESTINATION IDENTIFICATION
MRPDEBN  DS    0H                         DESTINATION IDENT EQUALS BLANKS
***********************************************************************
         SPACE 4                          LISTING CONTROL CARD - SPACE 4
***********************************************************************
MRPGTDN  DS    0H                         GET TRANSIENT DATA ENTRY
         MVC   TCATDDI,TWATDDI            MOVE DESTINATION IDENTIFICATION
         DFHTD TYPE=GET,                                                X
               NORESP=MRPNRCN,                                          X
               QUEZERO=MRPQERN,                                         X
               IDERROR=MRPDIEN
         DFHPC TYPE=ABEND,                                              X
               ABCODE=XMRP
***********************************************************************
         SPACE 2                          LISTING CONTROL CARD - SPACE 2
```

Message Switching Sample Program in ASM (Continued)

```
***********************************************************************
MRPNRCN  DS    0H                           NORMAL RESP CODE ENTRY ADDRESS
         L     TDIABAR,TCATDAA              LOAD TRANS DATA AREA ADDRESS
         DFHTC TYPE=(WAIT)
         MVC   MRPMTDI+1(1),TDIAIRL+1       MOVE DATA LENGTH TO MOVE INSTR
MRPMTDI  MVC   TIOATDL(0),TDIAIRL           MOVE TRANS DATA TO TERM AREA
         LH    14,TIOATDL                   LOAD TERMINAL DATA LENGTH
         SH    14,=H'4'                     SUBTRACT 4 FROM LENGTH
         STH   14,TIOATDL                   STORE TERMINAL DATA LENGTH
         DFHTC TYPE=(WRITE,                                                *
               SAVE)
         CLC   =CL3'ALL',TWAREAI            COMPARE RETRIEVE ALL IND TO ALL
         BNE   RETURN                       GO TO RETURN ENTRY IF NOT EQUAL
         MVI   TWAQEMCI,X'FF'               MOVE MESSAGE CONTROL INDICATOR
         B     MRPGTDN                      GO TO GET TRANSIENT DATA ENTRY
***********************************************************************
         SPACE 4                            LISTING CONTROL CARD - SPACE 4
***********************************************************************
MRPQERN  DS    0H                           DESTINATION QUEUE EMPTY ENTRY
         CLI   TWAQEMCI,X'FF'               COMPARE MESSAGE CONTROL IND
         BE    MRPNMQMB                     GO TO NO MSG QUEUED MSG BYPASS
         MVC   TIOATDL(MRPNQML),MRPNMQM MOVE TERMINAL MESSAGE TO OUTPUT
         B     MRPWRCS                      GO TO WRITE & RETURN TO C S
MRPNMQMB DS    0H                           NO MESSAGES QUEUED MSG BYPASS
         DFHTC TYPE=(WAIT)
         MVC   TIOATDL(MRPNMML),MRPNMMM MOVE NO MORE MESSAGE TO T O A
***********************************************************************
MRPWRCS  DS    0H                           WRITE AND RETURN TO CONT SYS
         DFHTC TYPE=(WRITE)
         B     RETURN                       GO TO RETURN ENTRY
***********************************************************************
         EJECT                              LISTING CONTROL CARD - EJECT
***********************************************************************
* *                                                                 * *
***********************************************************************
DCPDIEN  DS    0H                           DESTINATION IDENT ERROR ENTRY
         ST    TIOABAR,TCTTEDA              STORE TERM I/O AREA ADDRESS
MEPDIEN  DS    0H                           DESTINATION IDENT ERROR ENTRY
MRPDIEN  DS    0H                           DESTINATION IDENT ERROR ENTRY
         MVC   TIOATDL(MCPDEML),MCPDIEM MOVE TERMINAL MESSAGE TO OUTPUT
         DFHTC TYPE=(WRITE)
***********************************************************************
         SPACE 4                            LISTING CONTROL CARD - SPACE 4
RETURN   DS    0H                           RETURN TO CONTROL SYSTEM
         DFHPC TYPE=RETURN
***********************************************************************
         LTORG *                            LITERAL ORIGIN AT *
***********************************************************************
         END   CICSATP                      END OF ASSEMBLY - APPL TEST PGM
```

Message Switching Sample Program in COBOL

```
***********************************************************************
         C O B O L       E X A M P L E       P R O B L E M
***********************************************************************
       ID DIVISION.
       PROGRAM-ID.             CICSATP.
       ENVIRONMENT DIVISION.
       DATA DIVISION.
       WORKING-STORAGE SECTION.
      *
       77  DCPDCAML            PIC 99      COMP    VALUE 58.
       77  DCPDCAMD            PIC X(58)   VALUE   ' DATA COLLECTION HAS
      -        ' BEEN REQUESTED AND IS ABOUT TO BEGIN '.
       77  DCPEODML            PIC 99      COMP    VALUE 73.
       77  DCPEODMD            PIC X(73)   VALUE   ' THE DATA HAS BEEN R
      -        'ECEIVED AND DISPATCHED TO THE DESIGNATED DESTINATION
      -        ' '.
       77  MELMEAML            PIC 99      COMP    VALUE 77.
       77  MEPMEAMD            PIC X(77)   VALUE   'YOUR MESSAGE HAS BEE
      -         'N RECEIVED AND DISPATCHED TO THE DESIGNATED DESTINAT
      -        'ION  '.
       77  MRPNMML             PIC 99      COMP    VALUE 68.
       77  MRPNQML             PIC 99      COMP    VALUE 63.
       77  MCPDEML             PIC 99      COMP    VALUE 64.
      *
       01  MESSG1.
           03  MCPDIEM         PIC 99      COMP    VALUE 60.
           03  FILLER          PIC 99      COMP    VALUE ZERO.
           03  MESSAGE1        PIC X(60)   VALUE   '         DESTINATION
      -        ' IDENTIFICATION ERROR - PLEASE RESUBMIT '.
      *
       01  MESSG2.
           03  MRPNMMM         PIC 99      COMP    VALUE 64.
           03  FILLER          PIC 99      COMP    VALUE ZERO.
           03  MESSAGE2        PIC X(64)   VALUE   '         THERE ARE N
      -        'O MORE MESSAGES QUEUED FOR THIS DESTINATION '.
      *
       01  MESSG3.
           03  MRPNMQM         PIC 99      COMP    VALUE 59.
           03  FILLER          PIC 99      COMP    VALUE ZERO.
           03  MESSAGE3        PIC X(59)   VALUE   '         THERE ARE N
      -        'O MORE MESSAGES QUEUED FOR THIS DESTINATION '.
      *
       LINKAGE SECTION.
      *
       01  DFHBLLDS COPY DFHBLLDS.
           03  TCTTEAR         PIC S9(8)   COMP.
           03  TIOABAR         PIC S9(8)   COMP.
           03  TDIABAR         PIC S9(8)   COMP.
      *
       01  DFHCSADS COPY DFHCSADS.
      *
       01  DFHTCADS COPY DFHTCADS.
           03  TWATDDI         PIC X(4).
           03  TWAREAI         PIC X(4).
           03  TWAQEMCI        PIC S9      COMP.
      *
       01  DFHTCTTE COPY DFHTCTTE.
      *
       01  DFHTIOA COPY DFHTIOA.
           03  TIOADATA        PIC X(80).
           03  FILLER          REDEFINES TIOADATA.
               05  EODTEST     PIC X(3).
               05  FILLER      PIC X(77).
           03  FILLER          REDEFINES TIOADATA.
               05  TIOATID     PIC X(4).
               05  FILLER      PIC X.
               05  TIOADID     PIC X(4).
```

Message Switching Sample Program in COBOL (Continued)

```
           05  FILLER      REDEFINES TIOADID.
               07  TIOARAI1    PIC X(3).
               07  FILLER      PIC X.
           05  TIOARAI2        PIC X(3).
           05  FILLER      REDEFINES TIOARAI2.
               07  TIOAMBA     PIC X.
               07  FILLER      PIC XX.
           05  FILLER          PIC X(68).
*
 01  DFHTDIA COPY DFHTDIA.
     02  TDIADBA         PIC X(80).
*
 PROCEDURE DIVISION.
**************
 ATPIPIN.
**************
     MOVE    CSACDTA TO TCACBAR.
     MOVE    TCAFCAAA TO TCTTEAR.
     MOVE    TCTTEDA TO TIOABAR.

     IF      TIOATID = 'BSDC'
     THEN    GO TO ALPDCPN.

     IF      TIOATID = 'BSME'
     THEN    GO TO ALPMEPN.

     IF      TIOATID = 'BSMR'
     THEN    GO TO ALPMRPN.

     DFHPC   TYPE=ABEND,                                        *
         ABCODE=XAPT
*
**************
 ALPDCPN.
**************
* DATA COLLECTION
*
     MOVE    TIOADID TO TWATDDI.
     MOVE    DCPDCAML TO TIOATDL.
     MOVE    DCPDCAMD TO TIOADATA.

     DFHTC   TYPE=(WRITE,READ,WAIT)

 DCPTEWN.
     MOVE    TCTTEDA TO TIOABAR.

     IF      EODTEST = 'EDO'
     THEN    GO TO DCPEXIT.

     MOVE    TWATDDI TO TCATDDI.
     MOVE    ZEROES TO TCTTEDA.
     ADD     4 TO TIOATDL.

     DFHTD   TYPE=PUT,                                          *
         TDADDR=TIOATDL,                                        *
         NORESP=DCPNRCN,                                        *
         IDERROR=DCPDIEN

     DFHPC   TYPE=APEND,                                        *
         ABCODE=XDCP
```

Message Switching Sample Program in COBOL (Continued)

```
 DCPNRCN.
     MOVE    TIOABAR TO TCASCSA.
     DFHSC   TYPE=FREEMAIN
     DFHTC   TYPE=(READ,WAIT)
     GO TO   DCPTEWN.

 DCPEXIT.
     MOVE    DCPEODML TO TIOATDL.
     ADD     4 TO TIOATDL.
     MOVE    DCPEODMD TO TIOADATA.
     DFHTC   TYPE=WRITE
     GO TO   RETURN1.
*
**************
 ALPMEPN.
**************
* MESSAGE ENTRY
*
     MOVE    TIOADID TO TCATDDI.
     MOVE    TCTTETI TO TIOATID.
     ADD     4 TO TIOATDL.

     DFHTD   TYPE=PUT,                                               *
         TDADDR=TIOATDL,                                             *
         NORESP=MEPNRCN,                                             *
         IDERROR=MEPDIAN

     DFHPC   TYPE=ABEND,                                             *
         ABCODE=XMEP

 MEPNRCN.
     MOVE    MEPMEAML TO TIOATDL.
     ADD     4 TO TIOATDL.
     MOVE    MEPMEAMD TO TIOADATA.
     DFHTC   TYPE=WRITE
     GO TO   RETURN1.
*
**************
 ALPMRPN.
**************
* MESSAGE RETRIEVAL
*
     MOVE    TIOARAI2 TO TWAREAI.
     MOVE    TCTTETI TO TWATDDI.

     IF      TIOARAI1 NOT = 'ALL'
     THEN    GO TO MRPAI1B.

     MOVE    TIOARAI1 TO TWAREAI.
     GO TO   MRPDEBN.

 MRPAI1B.
     IF      TIOADID = SPACES
     THEN    GO TO MRPDEBN.

     MOVE    TIOADID TO TWATDDI.

 MRPDEBN.
 MRPGTDN.
     MOVE    TWATDDI TO TCATDDI.

     DFHTD   TYPE=GET,                                                *
         NORESP=MRPNRCN,                                              *
         QUEZERO=MRPQERN,                                             *
         IDERROR=MRPDIEN

     DFHPC   TYPE=ABEND,                                              *
         ABCODE=XMRP
```

Message Switching Sample Program in COBOL (Continued)

```
   MRPNRCN.
       MOVE     TDIAIRL TO TIOATDL.
       MOVE     TDIADBA TO TIOADATA.
       SUBTRACT 4 FROM TIOATDL.
       DFHTC    TYPE=(WRITE,WAIT,SAVE)

       IF       TWAREAI NOT = 'ALL'
       THEN     GO TO RETURN1.

       MOVE     255 TO TWAQEMCI.
       GO TO    MRPGTDN.

   MRPQERN.
       IF       TWAQEMCI = 255
       THEN     GO TO MRPNMQMB.

       MOVE     MRPNMQM TO TIOATDL.
       MOVE     MESSAGE3 TO TIOADATA.
       GO TO    MRPWRCS.

   MRPNMQMB.
       MOVE     MRPNMMM TO TIOATDL.
       MOVE     MESSAGE2 TO TIOADATA.

   MRPWRCS.
       DFHTC    TYPE=WRITE
       GO TO    RETURN1.

   DCPDIEN.
       MOVE     TIOABAR TO TCTTEDA.

   MEPDIEN.
   MRPDIEN.
       MOVE     MCPDIEM TO TIOATDL.
       MOVE     MESSAGE1 TO TIOADATA.
       DFHTC    TYPE=WRITE

  *
  **************
   RETURN1.
  **************
  *
       DFHPC    TYPE=RETURN
```

Message Switching Sample Program in PL/I

```
***********************************************************************
           P L / I        E X A M P L E        P R O B L E M
***********************************************************************
 /* PL/I EXAMPLE PROBLEM */
    DFHCOVER
 CICSATP:  PROCEDURE OPTIONS (MAIN,REENTRANT);
   %INCLUDE DFHCSADS;
   %INCLUDE DFHTCADS;
       2 TWATDDI CHAR (4),
       2 TWAREAI CHAR (4),
       2 TWAQEMCI BINARY FIXED (8);
   %INCLUDE DFHTCTTE;
   %INCLUDE DFHTIOA;
       2 TIOADATA CHAR (80);
     DECLARE 1 TIOA1 BASED (TIOABAR),
               2 FILL1 CHAR (12),
               2 TIOATID CHAR (4),
               2 FILL2 CHAR (1),
               2 TIOARAI1 CHAR (3),
               2 FILL3 CHAR (2),
               2 TIOAMBA CHAR (1);
     DECLARE 1 TIOA2 BASED (TIOABAR),
               2 FILL1 CHAR (12),
               2 EODTEST CHAR (3),
               2 FILL2 CHAR (2),
               2 TIOADID CHAR (4),
               2 FILL3 CHAR (1),
               2 TIOARAI2 CHAR (3);
   %INCLUDE DFHTDIA;
       2 TDIADBA CHAR (80);
     DCL MCPDEML FIXED BIN INIT(56), MCPDIEM CHAR(56) INITIAL
 ('     DESTINATION IDENTIFICATION ERROR - PLEASE RESUBMIT');
     DCL DCPDCAML FIXED BIN INIT(57), DCPDCAMD CHAR(57) INITIAL
 (' DATA COLLECTION HAS BEEN REQUESTED AND IS ABOUT TO BEGIN');
     DCL DCPEODML FIXED BIN INIT(72), DCPEODMD CHAR(72) INITIAL
 (' THE DATA HAS BEEN RECEIVED AND DISPATCHED TO THE DESIGNATED DESTINAT
 ION');
     DCL MEPMEAML FIXED BIN INIT(76), MEPEAMD CHAR(76) INITIAL
 (' YOUR MESSAGE HAS BEEN RECEIVED AND DISPATCHED TO THE DESIGNATED DEST
 INATION');
     DCL MRPNMML FIXED BIN INIT(60), MRPNMMM CHAR(60) INITIAL
 ('     THERE ARE NO MORE MESSAGES QUEUED FOR THIS DESTINATION');
     DCL MRPNQML FIXED BIN INIT(55), MRPNMQN CHAR(55) INITIAL
 ('     THERE ARE NO MESSAGES QUEUED FOR THIS DESTINATION');
 ATPIPIN:  TCTTEAR = TCAFCAAA;
           TIOABAR = TCTTEDA;
           IF TIOATID = 'PSDC' THEN GO TO ALPDCPN;
           IF TIOATID = 'PSME' THEN GO TO ALPMEPN;
           IF TIOATID = 'PSMR' THEN GO TO ALPMRPN;
         DFHPC TYPE=ABEND,                                               *
               ABCODE=XAPT
 /* DATA COLLECTION PROGRAM */
 ALPDCPN:  TWATDDI = TIOADID;
           TIOATDL = DCPDCAML;
           TIOADATA = DCPDCAMD;
         DFHTC TYPE=(WRITE,READ,WAIT)
 DCPTEWN:
           TIOABAR = TCTTEDA;
           IF EODTEST = 'EOD' THEN GO TO DCPEXIT;
           TCATDDI = TWATDDI;
           UNSPEC (TCTTEDA) = 0;
           TIOATDL = TIOATDL + 4;
         DFHTD TYPE=PUT,                                                 *
               TDADDR=TIOATDL,                                           *
               NORESP=DCPNRCN,                                           *
               IDERROR=DCPDIEN
         DFHPC TYPE=ABEND,                                               *
               ABCODE=XDCP
```

Message Switching Sample Program in PL/I (Continued)

```
DCPNRCN:  TCASCSA = TIOABAR;
        DFHSC TYPE=FREEMAIN
        DFHTC TYPE=(READ,WAIT)
          GO TO DCPTEWN;
DCPEXIT:  TIOATDL = DCPEODML;
          TIOADATA = DCPEODMD;
        DFHTC TYPE=WRITE
          GO TO RETURN;
/* MESSAGE ENTRY PROGRAM */
ALPMEPN:  TCATDDI = TIOADID;
          TIOATID = TCTTETI;
          TIOATDL = TIOATDL + 4;
        DFHTD TYPE=PUT,                                                 *
              TDADDR=TIOATDL,                                           *
              NORESP=MEPNRCN,                                           *
              IDERROR=MEPDIEN
        DFHPC TYPE=ABEND,                                               *
              ABCODE=XMEP
MEPNRCN:  TIOATDL = MEPMEAML; TIOADATA = MEPEAMD;
        DFHTC TYPE=WRITE
          GO TO RETURN;
  /* MESSAGE RETRIEVAL PROGRAM */
ALPMRPN:  TWAREAI = TIOARAI2; TWATDDI = TCTTETI;
          IF TIOARAI1  ¬= 'ALL' THEN GO TO MRPAI1B;
          TWAREAI = TIOARAI1;
          GO TO MRPDEBN;
MRPAI1B:  IF TIOADID = ' ' THEN GO TO MRPDEBN;
          TWATDDI = TIOADID;
MRPDEBN:  MRPGTDN: TCATDDI = TWATDDI;
        DFHTD TYPE=GET,                                                 *
              NORESP=MRPNRCN,                                           *
              QUEZERO=MRPQERN,                                          *
              IDERROR=MRPDIEN
        DFHPC TYPE=ABEND,                                               *
              ABCODE=XMRP
MRPNRCN:  TDIABAR = TCATDAA;
          TIOATDL = TDIAIRL - 4;
          TIOADATA = TDIADBA;
        DFHTC TYPE=(WRITE,WAIT,SAVE)
          IF TWAREAI ¬= 'ALL' THEN GO TO RETURN;
          TWAQEMCI = '11111111'B;
          GO TO MRPGTDN;
MRPQERN:  IF TWAQEMCI = '11111111'B THEN GO TO MRPNMQMB;
          TIOATDL = MRPNQML;
          TIOADATA = MRPNMQN;
          GO TO MRPWRCS;
MRPNMQMB: TIOATDL = MRPNMML; TIOADATA = MRPNMMM;
MRPWRCS:
        DFHTC TYPE=WRITE
          GO TO RETURN;
DCPDIEN:  TCTTEDA = TIOABAR;
MEPDIEN: MRPDIEN: TIOATDL = MCPDEML;
          TIOADATA = MCPDIEM;
        DFHTC TYPE=WRITE
RETURN:
          END CICSATP;
```

APPENDIX B. BMS MAP DEFINITION EXAMPLE

This appendix shows the BMS map definition macros used to generate the symbolic storage definition associated with an input map for a display with the format shown below. The appendix also shows, for each programming language, the symbolic storage definition that is generated by the macros.

```
                        • PAYROLL


          • NAME:


          • DATE:       • MMDDYY


          • SEX:        • ?MALE


          • SKILLS:     •              •


          • PAY
```

The following map definition macros would be used to create the symbolic storage definition (DSECT) associated with an input map for an assembler-language application program. (To create the map itself, the TYPE=DSECT operand would be replaced by a TYPE=MAP operand.)

```
MAPSET   DFHMSD TYPE=DSECT,MODE=IN,CTRL=(FREEKB,FRSET),                  *
               LANG=ASM, EXTATT=MAPONLY
MAP1     DFHMDI LINE=1,COLUMN=1,JUSTIFY=(LEFT,FIRST)
         DFHMDF POS=9,LENGTH=7,INITIAL='PAYROLL',ATTRB=BRT,               *
               HILIGHT=UNDERLINE
         DFHMDF POS=40,LENGTH=8,INITIAL='NAME:'
NAME     DFHMDF POS=49,LENGTH=20,ATTRB=IC, COLOR=RED
         DFHMDF POS=80,LENGTH=8,INITIAL='DATE:'
MONTH    DFHMDF POS=89,LENGTH=2,GRPNAME=DATE,INITIAL='MM',ATTRB=NUM
DAY      DFHMDF POS=91,LENGTH=2,GRPNAME=DATE,INITIAL='DD',                *
               JUSTIFY=(LEFT,BLANK)
YEAR     DFHMDF POS=93,LENGTH=2,GRPNAME=DATE,INITIAL='YY'
         DFHMDF POS=120,LENGTH=8,INITIAL='SEX:'
SEX      DFHMDF POS=129,LENGTH=5,ATTRB=DET,INITIAL='?MALE'
         DFHMDF POS=160,LENGTH=8,INITIAL='SKILLS:'
SKILLS   DFHMDF POS=169,LENGTH=4,ATTRB=UNPROT,OCCURS=3
         DFHMDF POS=200,LENGTH=8,INITIAL='PAY:'
PAY      DFHMDF POS=209,LENGTH=6,ATTRB=NUM, COLOR=BLUE
         DFHMSD TYPE=FINAL
         END
```

The assembler DSECT produced as a result of the above statements would be as follows:

```
MAPII    DS     0C
        SPACE
NAMEL    DS     CL2                        INPUT DATA FIELD LENGTH
NAMEF    DS     0C                         DATA FIELD FLAG
         DS     C                          DATA FIELD ATTRIBUTE
NAMEI    DS     CL20                       DATA FIELD
        SPACE
*  START NEW DATA GROUP DATE
DATEL    DS     CL2                        INPUT GROUP FIELD LENGTH
DATEF    DS     0C                         GROUP FIELD FLAG
         DS     C                          GROUP FIELD ATTRIBUTE
DATEI    DS     0C                         GROUP FIELD ORIGIN
         SPACE 2
MONTHI   DS     CL2                        DATA FIELD
        SPACE
DAYI     DS     CL2                        DATA FIELD
        SPACE
YEARI    DS     CL2                        DATA FIELD
        SPACE
SEXL     DS     CL2                        INPUT DATA FIELD LENGTH
SEXF     DS     0C                         DATA FIELD FLAG
         DS     C                          DATA FIELD ATTRIBUTE
SEXI     DS     CL1                        DATA FIELD
        SPACE
SKILLSD  DS     0C                         FIRST OCCURRING FIELD
SKILLSL  DS     CL2                        INPUT DATA FIELD LENGTH
SKILLSF  DS     0C                         DATA FIELD FLAG
         DS     C                          DATA FIELD ATTRIBUTE
SKILLSI  DS     CL4                        DATA FIELD
SKILLSN  EQU    *                          NEXT OCCURRING FIELD
         ORG    SKILLSD+3*(3+4)            ALLOCATE OCCURRING FIELD SPACE
        SPACE
PAYL     DS     CL2                        INPUT DATA FIELD LENGTH
PAYF     DS     0C                         DATA FIELD FLAG
         DS     C                          DATA FIELD ATTRIBUTE
PAYI     DS     CL6                        DATA FIELD
        SPACE
* * * END OF MAP DEFINITION * * *
         SPACE 3
         ORG
MAPSETT EQU *         * END OF MAPSET
* * * END OF MAP SET DEFINITION * * *
         SPACE 3
```

By changing LANG=ASM to LANG=COBOL in the DFHMSD macro, the following symbolic storage definition would be produced.

```
01   MAPII.
     02  NAMEL COMP PIC S9(4).
     02  NAMEF PIC X.
     02  NAMEI PIC X(20).
     02  DATEL COMP PIC S9(4).
     02  DATEF PIC X.
     02  DATEI.
       03  MONTHI PIC X(2).
       03  DAYI PIC X(2).
       03  YEARI PIC X(2).
     02  SEXL COMP PIC S9(4).
     02  SEXF PIC X.
     02  SEXI PIC X(1).
     02  SKILLSD OCCURS 3 TIMES.
       03  SKILLSL COMP PIC S9(4).
       03  SKILLSF PIC X.
       03  SKILLSI PIC X(4).
     02  PAYL COMP PIC S9(4).
     02  PAYF PIC X.
     02  PAYI PIC X(6).
```

Similarly, changing LANG=ASM to LANG=PLI in the DFHMSD macro would produce the following symbolic storage definition:

```
DCL 1 MAPII BASED(BMSMAPBR) UNALIGNED,
     2  NAMEL FIXED BIN(15,0),
     2  NAMEF CHAR(1),
     2  NAMEI CHAR(20),
     2  DATEL FIXED BIN(15,0),
     2  DATEF CHAR(1),
     2  DATEI ,
       3  MONTHI CHAR(2),
       3  DAYI CHAR(2),
       3  YEARI CHAR(2),
     2  SEXL FIXED BIN(15,0),
     2  SEXF CHAR(1),
     2  SEXI CHAR(1),
     2  SKILLSD(3),
       3  SKILLSL FIXED BIN(15,0),
       3  SKILLSF CHAR(1),
       3  SKILLSI CHAR(4),
     2  PAYL FIXED BIN(15,0),
     2  PAYF CHAR(1),
     2  PAYI CHAR(6),
     2  FILL0024 CHAR(1);
/* END OF MAP  DEFINITION  */
```

APPENDIX C. INTER-RELEASE COMPATIBILITY

This appendix defines the compatibility between application programs on the current release of CICS and previous releases of CICS. See the edition notice on page ii for the current releases of CICS.

There are two forms of application program compatibility:

- **Source compatibility**. The source code assembled or compiled on the current release of CICS generates an equivalent object program to that generated by previous releases of CICS
- **Object compatibility**. The object program, when executed under the current release of CICS will give the same results as it will when executed under previous releases of CICS.

In releases of CICS before 1.4, you could not code TIOAPFX=YES in the DFHMSD or DFHMDI macro for an assembler language application program. If you did, CICS disregarded it and used the default (TIOAPFX=NO). However, in releases of CICS later than 1.4 you can code TIOAPFX=YES for an assembler language program. This is a source incompatibility because you will get a different object program under releases of CICS later than 1.4 from that which would be produced under releases earlier than 1.4.

You should be aware that a change to the phonetic conversion built-in function (described in page 266) is an object incompatibility.

With the withdrawal of native ISAM support in CICS Version 1 Release 7, the ISAM I/O error code field, FCFIOEX, is no longer supported. Application programs accessing this field will exhibit both source and object incompatibility and will require modification.

DEFINITION OF THE APPLICATION PROGRAMMER INTERFACE

The remainder of this appendix defines the application programmer interface (API) that applies to users converting from previous releases.

The API is defined as the CICS macros, control block fields, and area prefix fields that are available for use by a user-written application program. With the exception of the single source incompatibility (TIOAPFX=YES) described above, application programs using these macros or fields will execute successfully under releases later than 1.4 of CICS without recompilation.

The macros and fields of the API that are valid for a given release are those documented in the CICS/VS Application Programmer's Reference Manual (Macro Level) for that release.

We do not guarantee compatibility for:

- macros or fields other than those documented in the manual
- fields marked "unused" or "reserved" in former releases
- fields used for purposes other than described in the manual.

Application programs containing such macros or fields should be recompiled, tested, and where necessary modified to ensure correct execution.

CICS MACROS

The following macros are those for which compatibility can be guaranteed for previous releases.

DFHBIF	DFHFC	DFHMDF	DFHTC
DFHBMS	DFHIC	DFHPC	DFHTD
DFHDC	DFHJC	DFHSC	DFHTR
DFHDI	DFHKC	DFHSP	DFHTS

The CICS/OS/VS CALLDLI macro is also part of the API.

Only the operands and parameters of the macros described in the CICS/VS Application Programmer's Reference Manual (Macro Level) for the release from which you are migrating are supported by CICS for use by user-written application programs.

CICS CONTROL BLOCK FIELDS AND AREA PREFIX FIELDS

Many of the fields in CICS areas, for example, the CSA, or prefixes to user I/O areas, for example, a TIOA, are referred to directly when a CICS macro is executed and it is essential that their location, type, and meaning remain unchanged across releases.

The following fields form the API for the current release of CICS.

CSABFNAC	-	Address of built-in functions
CSABMS	-	BMS address
CSACDTA	-	Common System Area Current Dispatched Task Address
CSACTODB	-	Common System Area Current Time of Day in Binary
CSADCNAC	-	Dump Control Entry Address
CSAFCNAC	-	File Control Entry Address
CSAICNAC	-	Time Control Entry Address
CSAICRNX	-	NOP/Branch Flush Routine Interface
CSAJCNA1	-	Journal Control Macro Entry Pointer 1
CSAJCNA2	-	Journal Control Macro Entry Pointer 2
CSAJYDP	-	Common System Area Date in Packed dec (000YYDDD)
CSAKCNAC	-	Task Control Entry Address
CSAOPFLA	-	Common System Area Optional Features List Address
CSAPCNAC	-	Program Control Entry Addr
CSASCNAC	-	Storage Control Entry Addr
CSASPNAC	-	Sync Point Program Ent Addr
CSATCNAC	-	Terminal Control Entry Addr
CSATCRWE	-	Terminal Control Read/Write Entry Address
CSATDNAC	-	Transient Data Control Entry Address
CSATODP	-	Common System Area Time of Day in Packed dec (4 bytes)
CSATRMF1	-	Trace Master Flags
CSATRMF2	-	Trace System Flags
CSATRMF3	-	Trace System Flags
CSATRNAC	-	Trace Control Entry Address
CSATRTBA	-	Common System Area TRace Table Address
CSATSNAC	-	Temporary Storage Entry Address
CSAWABA	-	Common System Area Work Area Beginning Address
FCDS01D	-	Beginning Address Data Area
FCFIOBEX	-	File Control File Input/ Output BDAM Error Code (4 bytes)
FCFIOFCT	-	File Control Entry Table Address
FCFIOLRA	-	Logical Record Address
FCIOERR	-	File Control File Input/ Output DAM Error Code (2 bytes)
FCUFCTA	-	File Control Table Entry Address
FCUPDRA	-	File Input/Output Area Address
FCUWA	-	File Control Update Work Area (data begin address)
FIOADBA	-	Data Beginning Address
FIOAIND	-	File I/O Area Indicator
FWAIND	-	File Work Area Indicator
JCAADATA	-	Journal Control Area Addr of DATA to be written to journal data set
JCAAPRFX	-	Journal Control Area Addr of User-Prefix Data
JCAECN	-	Journal Control Area Event Control Number (4 bytes)
JCAJCRC	-	Journal Control Area Journal Control Response Code (1 byte)
JCAJFID	-	Journal Control Area Journal File IDentification (1 byte)
JCAJRTID	-	Journal Control Area Journal Record Type IDentification (2 bytes)
JCALDATA	-	Journal Control Area Length of DATA to be written to journal data set (2 bytes)
JCALPRFX	-	Journal Control Area Length of user PReFiX (2 bytes)
JCANOTE	-	Note Request Returned-Data
JCARST	-	Run Start Time (HHMMSSS+)
JCATR1	-	Type Request Byte1
JCATR2	-	Type Request Byte2
JCATR3	-	(Reserved for CICS usage)
JCAVCD	-	Volume Creation Date (YYDDD+)
JCAVSN	-	Volume Sequence Number (NNN+)

SAASACA	-	Storage Accounting Area Storage Accounting Chain Address
SAASAD	-	Storage Area Displacement
SAASCI	-	Storage Class Identification
SAASFI	-	Storage Format Identification
TCAATAC	-	Abnormal Termination Abend Code
TCABFPAM	-	Address Pointer Initialization (Built-In Functions)
TCABFTR	-	Task Control Area Built-in Function Type of Request (1 byte)
TCABITF	-	Task Control Area BIT Manipulation address of one-byte bit Field to be operated on
TCABITR	-	Task Control Area BIT Manipulation Result of BITEST operation (1 byte)
TCABITTP	-	Bit Function Type Indicator
TCABITV	-	Task Control Area BIT Manipulation address of bit pattern (mask) to be applied to a specified byte
TCABMSCP	-	Task Control Area basic mapping support Cursor Position (2 bytes)
TCABMSMA	-	Task Control Area basic mapping support Map Address
TCABMSMN	-	Task Control Area basic mapping support Map Name (8 bytes)
TCACCCA	-	Task Control Area Common Control Communication Area
TCACCSV1	-	Save Area for Bytes Overlaid by DFHDC
TCACCSV2	-	Save Area for Bytes Overlaid by DFHDC
TCACHKR	-	Response Indicator (Built-In Function)
TCACKFD	-	Task Control Area Field Verify address of FielD to be ChecKed
TCACKLN	-	Task Control Area Field Verify LeNgth of field to be ChecKed (2 bytes)
TCADCDC	-	Task Control Area Dump Control Dump Code (4 bytes)
TCADCNB	-	Task Control Area Dump Control Number of Bytes in area to be dumped (2 bytes)
TCADCSA	-	Task Control Area Dump Control Storage Address of area to be dumped
TCADCTR	-	Task Control Area Dump Control Type of Request (ASM or PL/I; 2 bytes)
TCADIDNA	-	Task Control Area Batch Data Interchange Destination Name Address
TCADIKYA	-	Task Control Area Batch Data Interchange Key Addr
TCADINRS	-	Task Control Area Batch Data Interchange Number of Records in Request (1 byte)
TCADIRNA	-	Task Control Area Batch Data Interchange Relative Record Number Address
TCADIVNA	-	Task Control Area Batch Data Interchange Volume Name Address
TCADLFUN	-	Task Control Area DL/I FUNction (4 bytes)
TCADLIO	-	Task Control Area DL/I Input/Output area address
TCADLPCB	-	Task Control Area DL/I Program Control Block addr
TCADLPSB	-	Task Control Area DL/I program specification block name (8 bytes)
TCADLSSA	-	Task Control Area DL/I address of segment search argument list
TCADLTR	-	DL/I Type of Invalid Response
TCAFCAA	-	Task Control Area File Control Area Address
TCAFCAAA	-	Task Control Area Facility Control Area Associated Address
TCAFCDI	-	Task Control Area File Control Data set Identification (8 bytes)
TCAFCI	-	Facility Control Indicator
TCAFCNRD	-	Task Control Area File Control Number of Records Deleted (2 bytes binary)
TCAFCRI	-	Task Control Area File Control Record Identification (8 bytes)

TCAFCTR	- Task Control Area File Control Type of Request/ Response (ASM or PL/I; 1 byte)
TCAFCURL	- Task Control Area File Control Undefined Record Length (2 bytes)
TCAFLD	- Task Control Area Field Edit address of FieLD to be edited
TCAFLN	- Task Control Area Field Edit LeNgth of Field to be edited (2 bytes)
TCAICCLS	- Unique Identification of Request Identification
TCAICDA	- Task Control Area Interval Control Data Area
TCAICQID	- Task Control Area Interval Control reQuest IDentification (8 bytes)
TCAICQPX	- Task Control Area Interval Control reQuest Prefix (2 bytes)
TCAICRT	- Task Control Area Interval Control Request Time (4 bytes)
TCAICTEC	- Task Control Area Interval Control Timer Event Control area address
TCAICTI	- Task Control Area Interval Control Transaction Identification (4 bytes)
TCAICTID	- Task Control Area Interval Control Terminal IDentification (4 bytes)
TCAICTR	- Task Control Area Interval Control Type of Request/ Response (ASM or PL/I; 1 byte)
TCAINAM	- Name List Indicator
TCAINA1	- Task Control Area INput Formatting Address of list of offsets for the internal fixed-format TIOA
TCAINA2	- Task Control Area INput Formatting Address of list of field names that may appear in input stream
TCAINH1	- Task Control Area INput Formatting length of TIOA to be acquired for the internal fixed-format representation of data (Halfword field)
TCAINRC	- Task Control Area INput Formatting Response Code (1 byte)
TCAJCAAD	- Task Control Area Journal Control Area ADdress
TCAKCFA	- Task Control Area Task Control (KCP) Facility control area Address
TCAKCRC	- System Macro Return Code
TCAKCTA	- Task Control Area Task Control (KCP) TCA Address
TCAKCTI	- Task Control Area Task Control (KCP) Transaction Identification (4 bytes)
TCAMSFMP	- Task Control Area Mapping Support Function Management Parameter
TCAMSFSC	- Field Separator Characters
TCAMSHDR	- Task Control Area Mapping Support HeaDeR address (4 bytes)
TCAMSIOA	- Task Control Area Mapping Support Input/Output Area Address
TCAMSJ	- Task Control Area Mapping Support Justification (one byte)
TCAMSLDC	- Logical Device Code
TCAMSLDM	- LDC Mnemonic
TCAMSMSA	- Task Control Area Mapping Support Map Set Address
TCAMSMSN	- Task Control Area Mapping Support Map Set Name (8 bytes)
TCAMSOC	- Task Control Area Mapping Support Operator Class (3 bytes)
TCAMSOCN	- Overflow Control Number
TCAMSPGN	- Task Control Area Mapping Support PaGe Number (current page; 2 bytes binary)
TCAMSRC1- TCAMSRC3	- Task Control Area Mapping Support Response Code (one byte each)
TCAMSRID	- Task Control Area Mapping Support Request IDentification
TCAMSRI1	- Task Control Area Mapping Support Return Information (1 byte)
TCAMSRLA	- Task Control Area Mapping Support Routing List Address, or Returned page List Address

TCAMSRTI	- Task Control Area Mapping Support Routing Time or Time interval Indicator (4 bytes packed decimal)
TCAMSTA	- Task Control Area Mapping Support Title Address
TCAMSTI	- Task Control Area Mapping Support error Terminal Identification (4 bytes)
TCAMSTRL	- Task Control Area Mapping Support TRaiLer address (4 bytes)
TCAMSTR1- TCAMSTR7	- Task Control Area Mapping Support Type Request (one byte each)
TCAMSWCC	- Write Control Characters
TCANAME	- Task Control Area Phonetic Conversion 16-byte field containing data (NAME) to be phonetically encoded
TCANXTID	- Task Control Area NeXt Transaction IDentification (4 bytes)
TCAOCLA	- Open/Close List Address
TCAOCTR	- Open/Close Type of Request
TCAPCAC	- Task Control Area Program Control ABEND Code (4 bytes)
TCAPCARO	- Abend Recovery Option
TCAPCERA	- Task Control Area Program Control Exit Routine Addr
TCAPCLA	- Loaded Program Beginning Address
TCAPCPI	- Task Control Area Program Control Program Identification (8 bytes)
TCAPCPSW	- System Recovery Program PSW
TCAPCSR	- Program Control Secondary Request
TCAPCTR	- Type of Request/Response
TCAPHNR	- Task Control Area PHoNetic Conversion error Response indicator (contains X'54' if invalid name was encountered; 1 byte)
TCAPHON	- Task Control Area PHONetic Conversion 4-byte returned value
TCAPURGI	- Task Purge Indicator
TCASCIB	- Task Control Area Storage Control Initialization Byte
TCASCNB	- Task Control Area Storage Control Number of Bytes of storage requested (2 bytes)
TCASCSA	- Task Control Area Storage Control Storage Address of area acquired or to be freed
TCASCTR	- Storage Control Type of Request
TCASPTR	- Sync Point Request
TCASVMID	- Service Module Control Identification
TCATCDC	- Task Control Area Task Control Dispatcher Control indicator (1 byte)
TCATCDP	- Task Control Area Task Control Dispatching Priority (1 byte)
TCATCEA	- Task Control Area Task Control Event control area Address (ECB, CCB or list)
TCATCEI	- Task Control Event Control Indicator
TCATCQA	- Task Control Area Task Control enQueued resource length (high-order byte) and Address (3 low-order bytes)
TCATCTR	- Task Control Type of Request
TCATDAA	- Task Control Area Transient Data Area Address
TCATDDI	- Task Control Area Transient Data Destination Identification (4 bytes)
TCATDTR	- Task Control Area Transient Data Type of Request/ Response (ASM or PL/I; 1 byte)
TCATPAPR	- Application Request Response Code
TCATPCON	- Connection Type Flag
TCATPCS1	- External Control Request Byte 1
TCATPCS2	- External Control Request Byte 2
TCATPLDA	- Logic Device Code Entry Address
TCATPLDC	- Logical Device Code
TCATPLDM	- Logical Device Mnemonic
TCATPLRC	- Locate Return Code
TCATPOC2	- Operation Control Byte 2

TCATPOC3	-	Operation Control Byte 3
TCATPOS1	-	External Operation Request Byte 1
TCATPOS2	-	External Operation Request Byte 2
TCATPPNM	-	Program Name Field
TCATPTA	-	Terminal Address or Identification
TCATRF1	-	Trace Entry Data Area 1
TCATRF2	-	Trace Entry Data Area 2
TCATRID	-	Trace Entry Identification
TCATRID1	-	Trace Entry Identification Extension
TCATRMF	-	TCA Trace Control (Single Task)
TCATRTR	-	Type of Trace Request
TCATSAF	-	Task Control Area Table Search length of Field in Argument table entry to be compared with search argument (1 byte)
TCATSA1	-	Task Control Area Table Search Address of search argument
TCATSA2	-	Task Control Area Table Search Address of first entry in argument table
TCATSA3	-	Task Control Area Table Search Address of first function table entry
TCATSA4	-	Task Control Area Table Search Address of field in first entry in argument table to be compared with search argument
TCATSA5	-	Task Control Area Table Search Address of (1) function field within first function table entry, on input, or (2) function field within function table entry which contained value retrieved, on output
TCATSDA	-	Task Control Area Temporary Storage Data Address
TCATSDI	-	Task Control Area Temporary Storage Data Identification (8 bytes)
TCATSFC	-	Function Code (Built-In Function)
TCATSFF	-	Task Control Area Table Search length of Field in Function table entry to be retrieved (1 byte)
TCATSH1	-	Task Control Area Table Search maximum number of entries to be searched (halfword)
TCATSH2	-	Task Control Area Table Search length of each argument table entry (halfword)
TCATSH3	-	Task Control Area Table Search length of each function table entry (halfword)
TCATSH4	-	Task Control Area Table Search index value (relative to 1) identifying the matching argument table entry returned to application program; if zero, no matching entry was found (halfword)
TCATSRN	-	Task Control Area Temporary Storage Record Number
TCATSRPC	-	Task Control Area Table Search ResPonse Code
TCATSTR	-	Task Control Area Temporary Storage Type of Request/Response (Assembler or PL/I; 1 byte)
TCATSTR2	-	Type of Request (Secondary)
TCAWGAA	-	Task Control Area WeiGhted Retrieval VSWA pointer
TCAWGCNT	-	Task Control Area Weighted Retrieval Count of maximum number of records to be made available to application program; NRECDS parameter (halfword field)
TCAWGH1	-	Task Control Area WeiGhted Retrieval highest percentage of acceptability for a weighted retrieval function (halfword)
TCAWGH2	-	Task Control Area WeiGhted Retrieval lowest percentage of acceptability for a weighted retrieval function (halfword)
TCAWGH3	-	Task Control Area WeiGhted Retrieval percentage of acceptability of this record saved as the result of a weighted retrieval operation (Halfword field)

TCAWGH4	- Task Control Area WeiGhted Retrieval number of records left to be presented to user (Halfword field)
TCAWGH5	- Task Control Area WeiGhted Retrieval number of records dropped to remain within user-specified maximum (NRECDS) (Halfword field)
TCAWPAA	- VSWA Pointer
TCAWPA1	- Task Control Area Weighted Retrieval address of search argument
TCAWPA3	- Task Control Area Weighted Retrieval address of area containing record to be examined
TCAWPA4	- Task Control Area Weighted Retrieval Address of field within area containing record to be examined
TCAWPB1	- Task Control Area Weighted Retrieval character indicating format of search argument (1 byte)
TCAWPH1	- Task Control Area Weighted Retrieval length of search argument (halfword)
TCAWPH2	- Task Control Area Weighted Retrieval match value (halfword)
TCAWPH3	- Task Control Area Weighted Retrieval no match value (halfword)
TCAWPH4	- Task Control Area Weighted Retrieval upper limit of comparison range (halfword)
TCAWPH5	- Task Control Area Weighted Retrieval lower limit of comparison range (halfword)
TCAWPNL	- Task Control Area Weighted Retrieval NulL character (1 byte)
TCAWPTR	- Task Control Area Weighted Retrieval Type of Range (1 byte)
TCAWRAA	- Task Control Area Weighted Retrieval VSWA pointer
TCAWTDI	- Task Control Area Weighted ReTrieval Data Identification (eight bytes)
TCAWTH1	- Task Control Area Weighted ReTrieval maximum number of records to be retrieved (halfword)
TCAWTH2	- Task Control Area Weighted ReTrieval relative number of record with same partial key to be examined first (halfword)
TCAWTH3	- Task Control Area Weighted ReTrieval maximum percentage of acceptability for retrieved records (halfword)
TCAWTH4	- Task Control Area Weighted ReTrieval minimum percentage of acceptability for retrieved records (halfword)
TCAWTRC	- Task Control Area Weighted Retrieval Response Code (1 byte)
TCAWTRI	- Task Control Area Weighted ReTrieval address of partial key of Record at which retrieval is to begin (fullword)
TCTEASCC	- 3270 Alternate Screen Size (Columns)
TCTEASCL	- 3270 Alternate Screen Size (Rows)
TCTEASCZ	- 3270 Alternate Screen Size
TCTEDSCC	- 3270 Default Screen Size (Columns)
TCTEDSCL	- 3270 Default Screen Size (Rows)
TCTEDSCZ	- 3270 Default Screen Size
TCTEEOCI	- EOC or OC Received Indicator
TCTEFMHI	- FMH Area for 3600 Devices
TCTESIDI	- Field containing inbound SIGNAL data (4 bytes)
TCTESIDO	- Area for Outbound Signal Data
TCTETDST	- Data Stream Type Byte
TCTETXTF	- 3270 Text Feature Flag byte (APL/TEXT)
TCTEVLDC	- Logical Device Code
TCTE32EF	- 3270 Data Stream Extensions Flag
TCTE32SF	- 3270 Screen Size Flag

TCTTEAID	- Terminal Control Table Terminal Entry Attention IDentifier (used with the 3270 Information Display System; 1 byte)
TCTTEBMN	- Name of Format Image in Buffer
TCTTECAD	- Cursor Address in Binary
TCTTECIA	- Terminal Control Table Terminal Entry Control Information Area pointer
TCTTECIL	- Length of User Area
TCTTECR	- Request Completion Analysis
TCTTECRE	- Request Completion Extension
TCTTEDA	- Terminal Control Table Terminal Entry Data Address
TCTTEDES	- TCAM Destination Name
TCTTEFIB	- Terminal Feature Flag Byte
TCTTEOCL	- Operator Class Code
TCTTEPCF	- Terminal Control Table Terminal Entry Passbook Control Field (2980 General Banking Terminal System; 1 byte)
TCTTESC	- Terminal Storage Chain Address
TCTTESID	- Terminal Control Table Terminal Entry Station IDentification (2980 General Banking Terminal System; 1 byte)
TCTTETAB	- Terminal Control Table Terminal Entry TABs needed to position print element (2980 General Banking Terminal System; 1 byte)
TCTTETCM	- TCAM Operation Code Flag
TCTTETI	- Terminal Identification
TCTTETID	- Terminal Control Table Terminal Entry Teller IDentification (2980 General Banking Terminal; 1 byte)
TCTTETM	- Terminal Control Table Terminal Entry Terminal Model (1 byte)
TCTTETS	- Terminal Status
TCTTETT	- Terminal Control Table Terminal Entry Terminal Type (1 byte)
TDIADBA	- Transient Data Input Area Data Begin Address
TDIAIRL	- Transient Data Input Area Intrapartition Record Length (2 bytes)
TDIASAL	- Storage Accounting Area Length
TDIASCA	- Transaction Storage Chain Address
TDOADBA	- Transient Data Output Area Data Begin Address
TDOASAL	- Storage Accounting Area Length
TDOASCA	- Transaction Storage Chain Address
TDOAVRL	- Transient Data Output Area Variable Record Length (2 bytes)
TIOACLCR	- Terminal Input/Output Area ControL CharacteR (same as TIOALAC; 1 byte)
TIOADBA	- Terminal Input/Output Area Data Begin Address
TIOALAC	- Terminal Input/Output Area Line Address Control (same as TIOACLCR; 1 byte)
TIOATDL	- Terminal Input/Output Area Transmission Data Length (2 bytes)
TIOAWCI	- Write Control Indicator
TSIOADBA	- Temporary Storage Input/Output Area Data Begin Address
TSIOASAL	- Storage Accounting Area Length
TSIOASCA	- Transaction Storage Chain Address
TSIOAVRL	- Temporary Storage Input/Output Area Variable Record Length (2 bytes)
VSWAERRC	- Error Code
VSWAID	- RPL Identifier
VSWALEN	- VSAM Work Area record LENgth (4 bytes)
VSWAREA	- VSAM Work Area REcord Address
VSWARTNC	- RPL Return Code

APPENDIX D. 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
- 2980 Administrative Station Model 2
- 2980 Teller Station Model 4.

The line codes and processor codes listed in these tables are unique to CICS and are represented as standard EBCDIC characters.

KEY	ENGRAVING		LINE	CPU CODE		HLL
No.	Top(LC)	Front(UC)	Code	Numeric(LC)	Alpha(UC)	ID
0	MSG ACK	1	F1	AA	F1	
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	O	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 29. 2980-1 Character Set/Translate Table

KEY No.	ENGRAVING Top(LC)	Front(UC)[1]	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	¢ |		5C	70 (|)	4A (¢)	
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			

[1] No keyfront engraving on a 2980 Administration Station Model 2

Figure 30. 2980-2 Character Set/Translate Table

KEY	ENGRAVING		LINE	CPU CODE		HLL
No.	Top(LC)	Front(UC)	Code	Numeric(LC)	Alpha(UC)	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	OB	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	O	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 31. 2980-4 Character Set/Translate Table

BIBLIOGRAPHY

CICS PUBLICATIONS

For further information about CICS refer to the following IBM publications:

CICS/VS 1.7

Customer Information Control System/Virtual Storage (CICS/VS):

- General Information, GC33-0155
- Library Guide, GC33-0356
- Application Programming Primer, SC33-0139

CICS/OS/VS 1.7

Customer Information Control System/Operating System/Virtual Storage (CICS/OS/VS) Version 1 Release 7:

- Release Guide, GC33-0132
- Facilities and Planning Guide, SC33-0202
- Installation and Operations Guide, SC33-0071
- CICS-Supplied Transactions, SC33-0240
- Customization Guide, SC33-0239
- Resource Definition (Macro), SC33-0237
- Resource Definition (Online), SC33-0186
- Intercommunication Facilities Guide, SC33-0230
- Recovery and Restart Guide, SC33-0231
- Performance Guide, SC33-0229
- Performance Data, SC33-0212
- Application Programmer's Reference Manual (Command Level), SC33-0241
- Application Programmer's Reference Summary (Command Level), GX33-6047
- Messages and Codes, SC33-0226
- IBM 3270 Data Stream Device Guide, SC33-0232
- IBM 4700/3600/3630 Guide, SC33-0233
- IBM 3650/3680 Guide, SC33-0234
- IBM 3767/3770/6670 Guide, SC33-0235
- IBM 3790/3730/8100 Guide, SC33-0236
- Problem Determination Guide, SC33-0242
- Program Debugging Reference Summary, SX33-6048
- Diagnosis Reference, LC33-0243
- Data Areas, LY33-6035
- Master Index, SC33-0227

CICS/DOS/VS 1.7

Customer Information Control System/Disk Operating System/ Virtual Storage (CICS/DOS/VS) Version 1 Release 7:

- Release Guide, GC33-0130
- Facilities and Planning Guide, SC33-0228
- Installation and Operations Guide, SC33-0070
- CICS-Supplied Transactions, SC33-0080
- Customization Guide, SC33-0131
- Resource Definition (Macro), SC33-0149
- Resource Definition (Online), SC33-0238
- Intercommunication Facilities Guide, SC33-0133
- Recovery and Restart Guide, SC33-0135
- Performance Guide, SC33-0134
- Performance Data, SC33-0219
- Application Programmer's Reference Manual (Command Level), SC33-0077
- Application Programmer's Reference Summary (Command Level), GX33-6012
- Application Programmer's Reference Manual (RPGII), SC33-0085
- Messages and Codes, SC33-0081
- IBM 3270 Data Stream Device Guide, SC33-0096
- IBM 4700/3600/3630 Guide, SC33-0072

IBM 3650/3680 Guide, SC33-0073

IBM 3767/3770/6670 Guide, SC33-0074

IBM 3790/3730/8100 Guide, SC33-0075

Problem Determination Guide, SC33-0089

Program Debugging Reference Summary, SX33-6010

Diagnosis Reference, LC33-0105

Data Areas, LY33-6033

Master Index, SC33-0095

ASSOCIATED PUBLICATIONS

The reader of this book may also want to refer to the following IBM publications:

An Introduction to the IBM 3270 Information Display System, GA27-2739

Component Description: IBM 2721 Portable Audio Terminal, GA27-3029

IBM 3790 Communication System Library Reference Summary, GX23-0205

VTAM Concepts and Planning, GC27-6998

Systems Network Architecture (SNA):

Function Description of Logical Unit Types, GC20-1868

Types of Logical Unit to Logical Unit Sessions, GC20-1869

DPPX/Distributed Presentation Services Version 2 System Programming Guide, SC33-0117

Screen Definition Facility/CICS SDF/CICS Program Reference Manual, SH19-6077

DOS/VS COBOL Compiler and Library Programmer's Guide, SC28-6478

OS/VS COBOL Compiler and Library Programmer's Guide, SC28-6483

OS PL/I Optimizing Compiler Programmer's Guide, SC33-0006

DOS PL/I Optimizing Compiler Programmer's Guide, SC33-0008

IMS/VS Application Programming for CICS/VS Users, SH20-9026

DL/I DOS/VS Application Programming Reference Manual, SH12-5411

DL/I DOS/VS Utilities and Guide for the System Programmer, SH12-5412

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. 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

Customer Information Control System
CICS/VS Version 1 Release 7
Application Programmer's Reference Manual
(Macro Level)

READER'S COMMENT FORM

Order No. SC33-0079-5

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.

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 . . .

Please use pressure-sensitive or other gummed tape to seal this form.

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 the address in the Edition Notice on the back of the title page.)

SC33-0079-5

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 NO. 40 ARMONK, N.Y.

POSTAGE WILL BE PAID BY ADDRESSEE:

International Business Machines Corporation
Department 6R1H,
180 Kost Road,
Mechanicsburg, PA 17055, USA

Fold and tape — Please Do Not Staple — Fold and tape

IBM®