SH20-1047-4

Program Product

Customer Information Control System (CICS) Application Programmer's Reference Manual

Program Numbers 5736-XX6 (DOS-ENTRY)
5736-XX7 (DOS-STANDARD)
5734-XX7 (OS-STANDARD V2)

The IBM Customer Information Control System (CICS) is a transaction-oriented, multiapplication data base/data communication interface between a System/360 or System/370 operating system and user-written application programs. Applicable to most online systems, CICS provides many of the facilities necessary for standard terminal applications: message switching, inquiry, data collection, order entry, and conversational data entry.

CICS is available in three systems—two for DOS users and one for OS users. Because the two CICS/DOS systems are compatible with each other and with the CICS/OS system, it is possible to start with a small data base/data communication configuration and move up through DOS into OS.

This manual provides information of interest to persons defining, designing, and preparing application programs to execute under CICS.

IBM

Fifth Edition (December 1972)

This edition is a major revision obsoleting SH20-1047-3.

This edition applies to Version 1, Modification Level 1, of the CICS/DOS-ENTRY (5736-XX6) and CICS/DOS-STANDARD (5736-XX7) program products and to Version 2, Modification Level 3, of the CICS/OS-STANDARD (5734-XX7) program product; it also applies to all subsequent versions and modifications unless otherwise indicated in new editions or Technical Newsletters.

If changes are made to the information herein, the edition that is applicable and current will be indicated in the latest System/360 and System/370 SRL Newsletter (GN20-0360).

Copies of this and other IBM publications can be obtained through IBM branch offices.

A form has been provided at the back of this publication for reader's comments. If this form has been removed, address comments to: IBM Corporation Technical Publications Department, 1133 Westchester Avenue, White Plains, New York 10604. Comments become the property of IBM.

PREFACE

This publication contains detailed information necessary to design and prepare application programs to execute under three IBM program products: CICS/DOS-ENTRY, CICS/DOS-STANDARD, and CICS/OS-STANDARD V2. It provides application programmers, system programmers, system analysts, and system administrators with information concerning real-time application programming considerations, application program organization, storage definition, the use of CICS macro instructions to request supervisory and data management services, data base considerations, and program testing and debugging.

Throughout this publication, parentheses are used in the notation of CICS macro instructions to indicate those operands where more than one applicable parameter can be specified with a single use of the operand. Where parentheses are not used, only one parameter at a time can be specified as part of the operand. An asterisk in (card) column 72 indicates that the macro instruction is continued on the next line (card). The first operand on a continuation card must begin in column 16.

The words "transaction" and "task" have the same connotation in CICS and are used interchangeably throughout this publication; the processing of a transaction may involve the execution of one or more "programs".

For further information concerning CICS, see the following IBM publications:

General Information Manual (GH20-1028)
System Programmer's Reference Manual (SH20-1043)
Terminal Operator's Guide (SH20-1044)
Operations Guide (CICS/DOS) (SH20-1034)
Operations Guide (CICS/OS) (SH20-1048)
Logic Manual (CICS/DOS-ENTRY) (LY20-0712)
Logic Manual (CICS/DOS-STANDARD) (LY20-0713)
Logic Manual (CICS/OS-STANDARD V2) (LY20-0714)

All references to CICS/OS and CICS/OS-STANDARD in this publication are references to the CICS/OS-STANDARD V2 system.

CONTENTS

INTRODUCTION

The IBM Customer Information Control System (CICS) is a multi-application data base/data communication interface between a System/360 or System/370 operating system and user-written application programs. Applicable to most online systems, CICS provides many of the facilities for standard terminal applications: message switching, inquiry, data collection, order entry, and conversational data entry.

Functions performed by CICS include:

- Control of a mixed telecommunications network
- Concurrent management of a variety of programs
- Controlled access to the data base
- Management of resources for continuous operation
- Prioritization of processing

By eliminating many of the development requirements for such functions of a real-time control system, CICS allows programmers to concentrate instead on implementing applications, dramatically reducing implementation time and cost.

Functions needed to support a data base/data communication system and standard terminal applications are provided by the following CICS management functions:

- Task Management - Provides its own dynamic multitasking facilities necessary for effective, concurrent transaction processing. Functions associated with this facility include priority scheduling, transaction synchronization, and control of serially reusable resources. This CICS function is in addition to the multitasking or multiprocessing capability of the host operating system.

- Storage Management - Controls main storage allocated to CICS. Storage acquisition, disposition, initialization, and request queuing are among the services and functions performed by this component of CICS.

- Program Management - Provides a multiprogramming capability through dynamic program management while offering a real-time program fetch capability.

- Program Interrupt Management - Provides for the interception of program interrupts by CICS to prevent total system termination. Individual transactions that program check are terminated by CICS with a dump (if Dump Management is used), thus preventing the entire CICS partition/region from terminating. Under CICS/OS, supports the runaway task control function of CICS Time Management.

- Time Management - Provides control of various optional task functions (system stall detection, runaway task control, task synchronization, etc.) based on specified intervals of time or the time of day.

- Dump Management - Provides a facility to assist in analysis of programs and transactions undergoing development or modification. Specified areas of main storage are dumped onto a sequential data set, either tape or disk, for subsequent offline formatting and printing using a CICS utility program.

- Terminal Management - Provides polling according to user-specified line traffic control as well as user requested reading and writing.

This facility supports automatic task initiation to process new transactions. The testing of application programs is accommodated by the simulation of terminals through sequential devices such as card readers, line printers, disk, tape, etc.

- File Management - Provides a data base facility using direct access and indexed sequential data management. This function supports updates, additions, random retrieval, and selective retrieval (browsing), of logical data on the data base. Optional access to the Data Language/I (DL/I) facility of the IBM Information Management System (IMS/360) is also provided under CICS/OS. Use of DL/I requires installation of the IMS/360 Version 2, Modification Level 2 (or later) Data Base System (5734-XX6).

- Transient Data Management - Provides the optional queuing facility for the management of data in transit to and from user defined destinations. This function facilitates message switching, data collection, and logging.

- Temporary Storage Management - Provides the optional general purpose "scratch pad" facility. This facility is intended for video display paging, broadcasting, data collection suspension, conservation of main storage, retention of control information, etc.

In addition to these management functions, CICS provides system service programming to identify terminal operators, to give dynamic control of the entire system to a master terminal, to display real-time system statistics, to intercept abnormal conditions not handled directly by the operating system, to provide basic mapping support for the 3270 Information Display System, and to end operation by gathering summary statistics, closing data sets, and returning control to the operating system.

REAL-TIME APPLICATION PROGRAMMING

In the conventional batch processing environment, the application programmer plans a series of runs to edit batches of input transactions, update master files (data sets), and write output reports. To optimize total run time and streamline the cycle, he must concentrate on careful manipulation of data. In accomplishing this, the data becomes intricately tied to his program logic and is of little value to other applications.

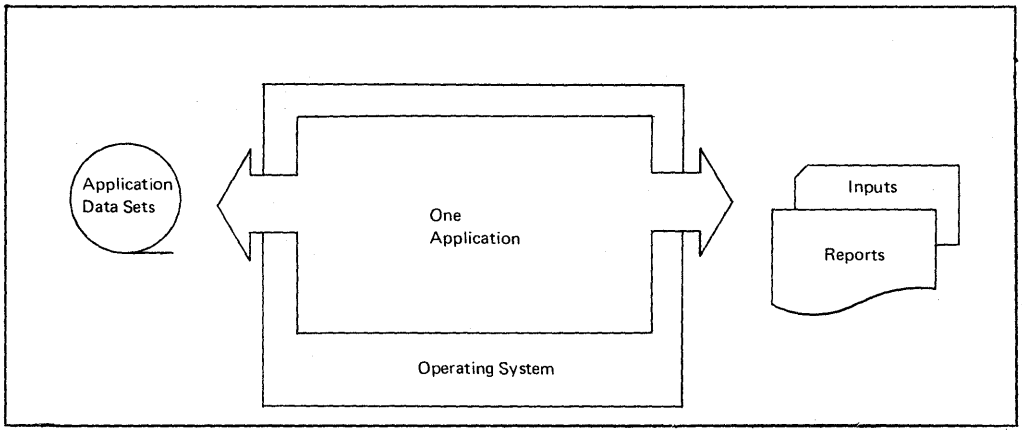


Figure 1. Conventional batch processing

The real-time data base/data communications (DB/DC) environment differs from the conventional batch processing environment primarily in the amount and types of concurrent activities that are likely to occur within the system at a given time. Whereas a batch processing system schedules each application independently and provides data support unique to each application, a DB/DC system controls many transactions arriving on a random nonscheduled basis and provides an integrated data base supporting each application.

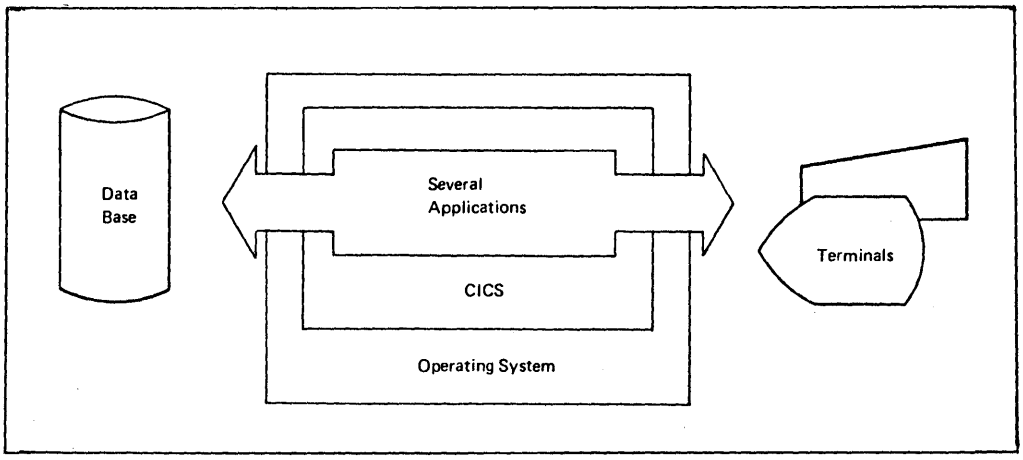


Figure 2. Transaction processing of CICS

In the past, the successful systems have been known as:

- Online information systems
- Real-time informational systems
- Teleprocessing systems
- Data base/data communication systems

These systems required the user to develop a control system that would:

- Host a telecommunication network of mixed devices
- Concurrently manage a wide mixture of transactions being serviced by a variety of programs
- Provide effective controlled access to the data base
- Effectively manage resources, such as main storage, to keep the system in continuous operation
- Prioritize the use of the processing facility
- Provide other real-time facilities necessary for the support of the applications and the environment
- Provide the ancillary system service functions necessary for the successful implementation of data base/data communication systems
- Provide rapid response to the terminals

CICS solves many of these complexities for the application programmer by managing data centrally (in a data base) on behalf of all applications. This shifts the burden of system management considerations from the application programmer to the system programmer and allows the application programmer to concentrate instead on the application.

A key consideration in the selection of a data base/data communication system is that it be appropriate for today's needs and have the growth potential that characterizes the DB/DC environment. CICS is intended to address precisely that consideration; that is, CICS is a family of systems that provides a DB/DC interface to the IBM System/360 and System/370 at most levels of the product line, providing a clearly visible growth or migration path as the user's environment dictates.

Figure 3 shows how the CICS data base supports the information needs of each application, independently and concurrently.

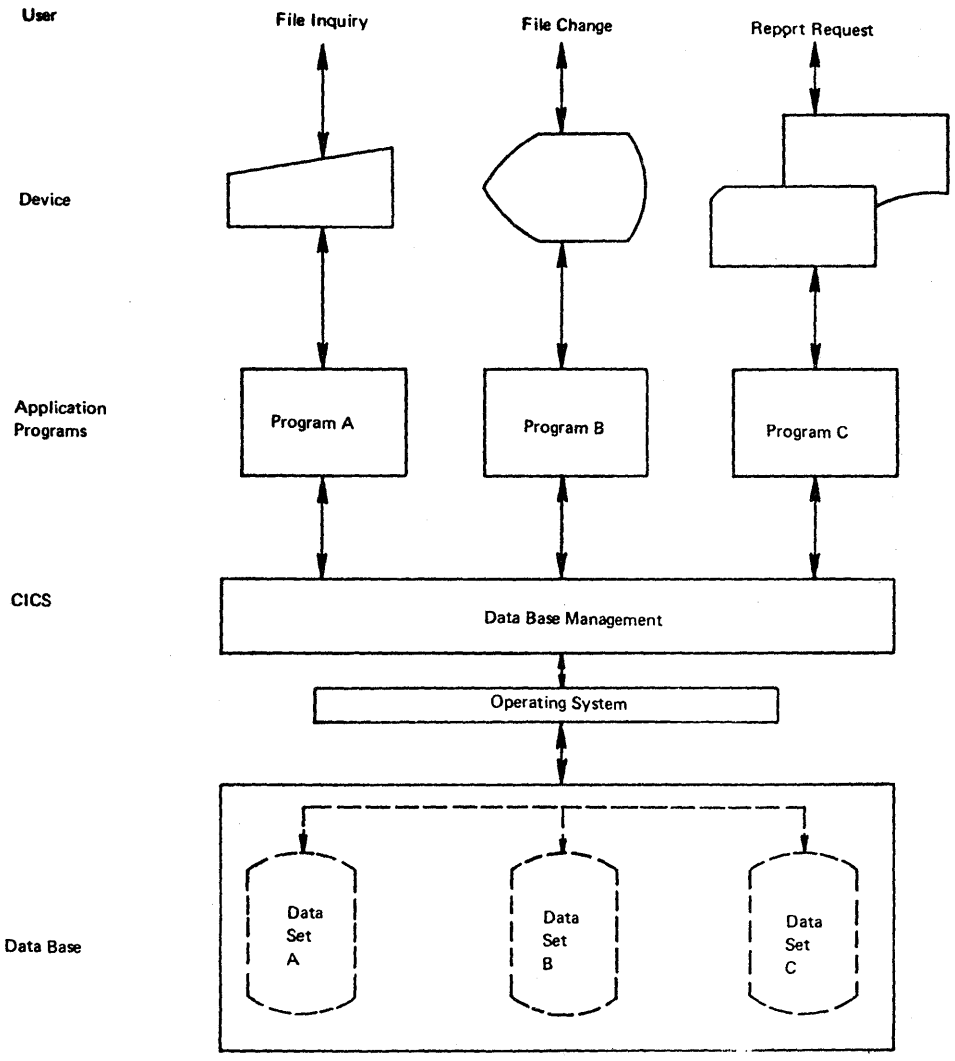


Figure 3. CICS data base concept

PROGRAM STRUCTURE

The user's application programs are processed concurrently by CICS as transactions (tasks). Although application programs may be as large as 32K bytes, it is recommended that each application program be developed modularly and kept to a minimum size. Large application programs can prevent the loading of other required programs during the operation of CICS and thus degrade the overall system performance.

CICS facilitates the modularity of application programs by allowing programs to easily communicate with other programs through the execution of CICS macro instructions. Since application programs do not contain input/output areas or transaction work areas, a 4K application program, when assembled, could contain as many as 1000 machine instructions.

Application programs can be written in Assembler language, ANS COBOL, or PL/I to execute under CICS. Regardless of the language used, it is strongly recommended that the application programmer allow CICS to perform all supervisory and data management services for his applications by issuing CICS macro instructions to invoke the desired services. Although the application programmer is not precluded from direct communication with the operating system, the results of such action are unpredictable and performance may be affected. Such action would also have a limiting effect on migration from CICS/DOS to CICS/OS, if this were ever desired.

An application program written in PL/I must consist of an external (MAIN) procedure. Internal procedure CALL's are allowed in a CICS program; external CALL's are not.

QUASI-REENTRANCE

Application programs must be coded so that they are "serially reusable" between entry and exit points of the program. Entry and exit points of an application program coincide with the use of CICS macro instructions, since an application program temporarily loses control after it begins executing only upon execution of a CICS macro instruction. 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.

This required quality of application programs written to run under CICS is called "quasi-reentrance", since the programs need not meet System/360 or System/370 specifications for true reentrance. Quasi-reentrance allows a single copy of a user-written application program to be used to process several transactions concurrently, thereby reducing the requirement for multiple copies of the same program in main storage.

If intermediate exits are taken in an application program, all switches, data, and intermediate results needed upon subsequent return to that transaction 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 the section on "Symbolic Storage Definitions").

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

CICS TRANSACTION FLOW

CICS executes in a multitasking mode of operation. Therefore, whenever a transaction (task) is waiting for I/O completion, other CICS transactions may continue to execute.

Figure 4 illustrates CICS multitasking where the same application program is used by three different transactions (A, B, and C). The application program performs a data base read and a subsequent write. Solid lines indicate that the transaction is executing; broken lines indicate that the transaction is waiting.

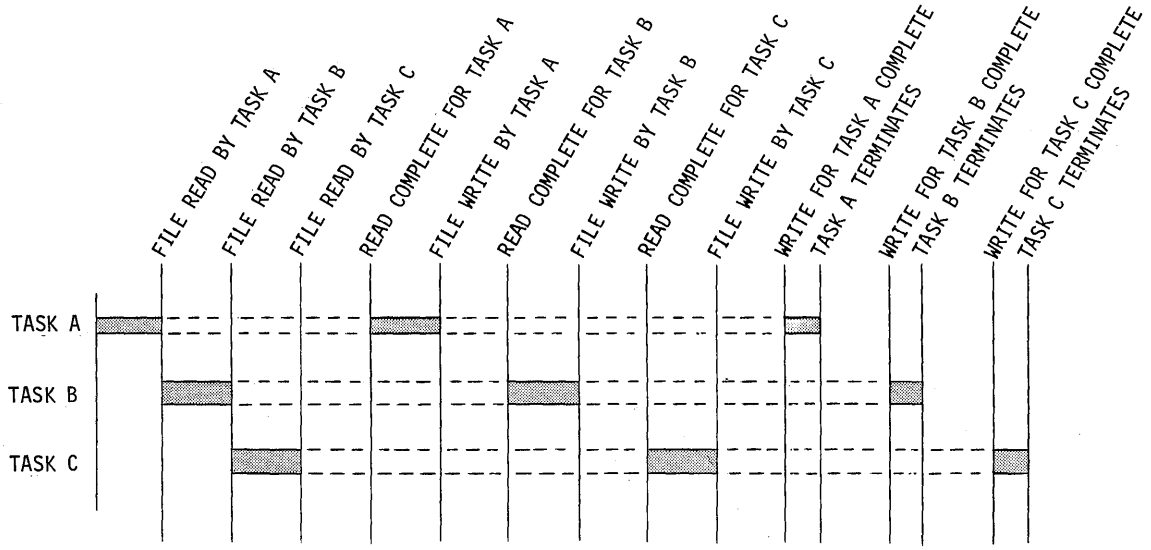


Figure 4. CICS multitasking

Figure 5 illustrates the logical flow of a typical transaction through CICS.

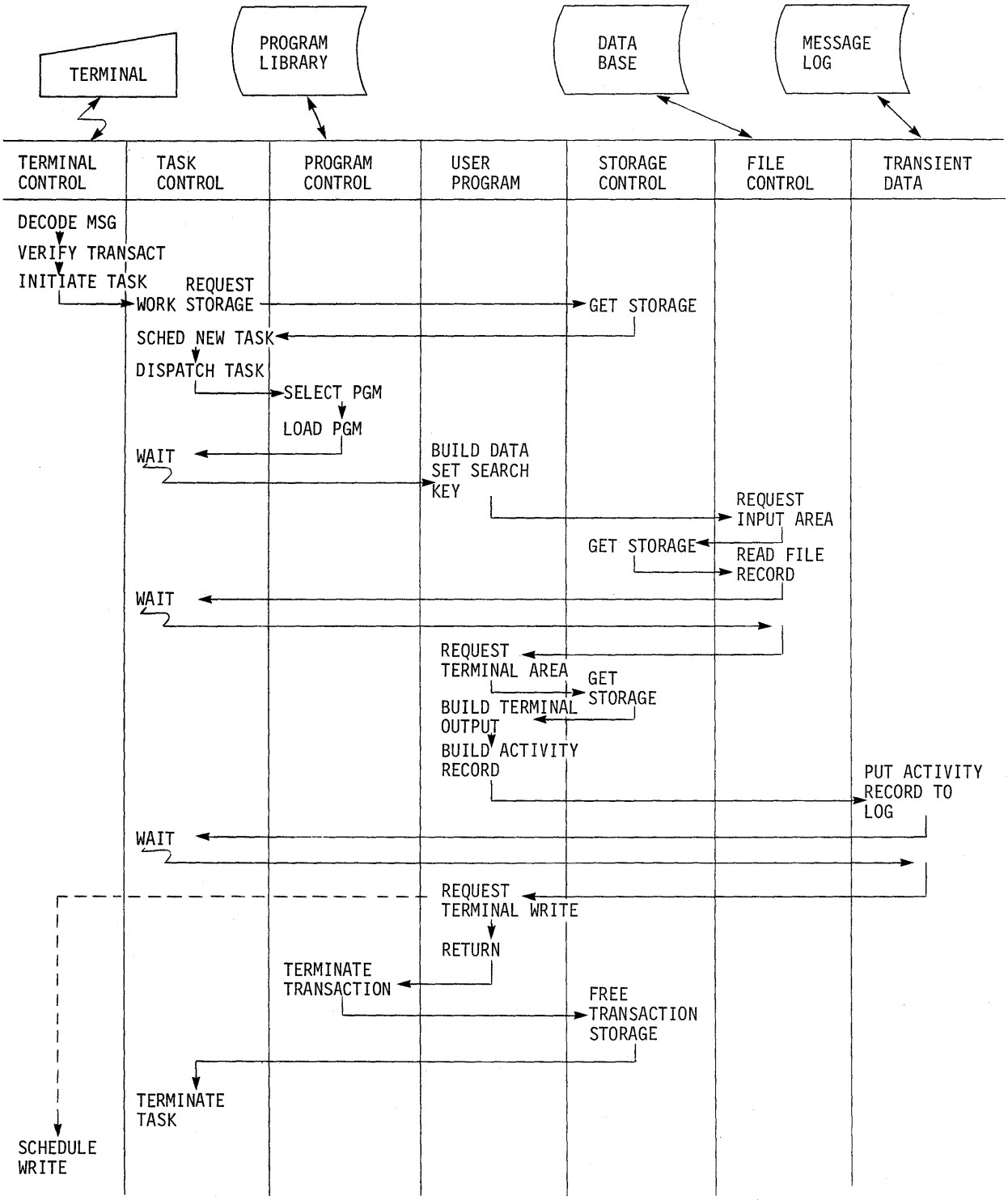


Figure 5. CICS transaction flow

STORAGE DEFINITION

The source 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 in his programming rather than develop actual or direct displacements. This protects the application program in the event of any relocation of CICS.

For the PL/I programmer, the source library contains numerous BASED structures of CICS control areas. These dummy sections are available to the user through use of the %INCLUDE statement. The ANS COBOL programmer makes use of these definitions through use of the COPY statement in the Linkage Section of the Data Division. These definitions are discussed in the Storage Definition section of this manual.

PROGRAM INITIALIZATION

In the initialization section of the application program, the application programmer must establish a symbolic base address for his program as this is not done by CICS prior to entry. Register 12 is reserved by CICS to contain the address of the Task Control Area (TCA) for this task. Register 13 is reserved to contain the address of the Common System Area (CSA). These registers are initialized by CICS prior to entry and must be preserved throughout the execution of the program. For ANS COBOL and PL/I, this situation is resolved by CICS and is of no concern to the application programmer.

Registers 15 through 11 are available to the user and are kept intact when a CICS macro instruction is issued; the contents of register 14 are destroyed any time a CICS macro instruction is issued.

The status of all registers upon program entry or upon return to a program is as follows:

REGISTERS	15 through 11	12	13	14
Initial program entry	Unknown	TCA	CSA	User program address
LINK	Linked from registers	TCA	CSA	User program address
XCTL	Transfer control from registers	TCA	CSA	User program address
LOAD	Unchanged	TCA	CSA	Next sequential instruction
RETURN	Linked from registers	TCA	CSA	Next sequential instruction

Note: Even though register 14 contains the program entry address, it is not advisable to use register 14 as the base register

since it is used by CICS to service requests for CICS supervisory and data management services.

SERVICE INVOCATION

ASSEMBLY TIME SERVICE

The DFHCOVER macro instruction is used to request the Assembler or Compiler to print a cover page on two consecutive pages. In this way, the application program listing may be torn off with one of the cover pages face up. Pertinent information (for example, program name, date, time of assembly, remarks, etc.) may then be written on the cover page.

The DFHCOVER macro instruction requires no operands and should appear with nothing else on the card.

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

If the DFHCOVER macro instruction is coded as part of an ANS COBOL application program, it should be coded preceding the IDENTIFICATION DIVISION card.

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

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. This is accomplished through the *PROCESS card for CICS/DOS and the EXEC card for CICS/OS. For further information concerning PL/I compile-time services, see the publication DOS PL/I Optimizing Compiler Programmer's Guide (SC33-0008) or the publication OS PL/I (F) Programmer's Guide (GC28-6594).

The examples contained in Appendix A show how the DFHCOVER macro instruction is used.

SUPERVISORY AND DATA MANAGEMENT SERVICES

The various CICS supervisory and data management services are invoked through use of CICS macro instructions. These macro instructions are written in Assembler language and, as all Assembler language instructions, are written in the following format:

Name	Operation	Operands	Comments
blank or symbol	DFHxxxxx	One or more operands separated by commas	

The name field of a CICS macro instruction must be left blank if the macro instruction is used in conjunction with a high-level language

(ANS COBOL or PL/I); if a label is desired for the macro instruction, it may be placed on the card preceding the macro instruction.

The operation field of a CICS macro instruction must begin before card 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 the symbolic name for the appropriate program or table. Since DFH is reserved for CICS macro instructions, no other statement may begin with this three-character combination.

The operand field of a CICS macro instruction contains one or more operands separated by commas. In this publication, parentheses are used to indicate those operands where more than one applicable parameter (keyword and otherwise) can be specified with a single use of the operand. Where parentheses are not used, only one parameter at a time can be specified as part of the operand; a choice must be made in the case of more than one applicable parameter. Since a blank character indicates the end of the operand field, the operand field must not contain blanks except after a comma on a continued card or after the last operand of the macro instruction. The first operand on a continuation card must begin in column 16.

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

See the section "Service Invocation" later in this publication for a detailed description of how CICS macro instructions are used to request CICS supervisory and data management services. See Appendix B for a listing of the CICS macro instructions that may be used by the application programmer.

The use of CICS macro instructions in a PL/I application program precludes the use of the following PL/I features:

1. The multitasking functions: COMPLETION, STATUS, PRIORITY.
2. The multitasking options: PRIORITY, TASK, EVENT, REPLY.
3. The PL/I statements: READ, WRITE, GET, PUT, OPEN, CLOSE, DISPLAY, SORT, DELAY, ON.

The use of CICS macro instructions in a PL/I optimizing compiler application program also precludes the use of the following options:

1. REPORT, FLOW, GONUMBER, GOSTMT.

The use of CICS macro instructions in an ANS COBOL application program precludes the use of the following ANS COBOL features:

1. Environment and Data Division entries normally associated with the data management services.
2. File Section of the Data Division.
3. Special features: SORT, REPORT WRITER, SEGMENTATION, EXHIBIT, TRACE, DISPLAY, and ACCEPT. (DISPLAY and ACCEPT can be used in conjunction with the system console.)
4. Options that may lead to the issuance of a STXIT (AB) SVC or STAE SVC: FLOW, STATE, STXIT, or SYMDMP for CICS/DOS; FLOW or STATE for CICS/OS.

Separate ANS COBOL routines or separate PL/I routines may not be link edited together. Separate Assembler-language routines may be link edited together, however, the CALLed routine must conform to CICS application program requirements. CICS provides the user with the LINK

and XCTL (transfer control) facilities to provide the necessary communication between programs. CICS macro instructions should not be coded within an ANS COBOL statement, since each ANS COBOL statement generated by a CICS macro instruction is terminated by a period.

SYMBOLIC STORAGE DEFINITIONS

CICS defines a number of main storage areas which the application program can use during execution. These storage areas are of three types:

1. Control areas
2. Work areas
3. Input/output areas

Information is stored and retrieved from these areas by CICS and by application programs.

Some of the storage areas are statically created by CICS during System Initialization and others are dynamically acquired and released during execution of the system. Some of the areas are acquired or created by CICS; some are acquired directly by the application program; and some are acquired by both CICS and the application program.

All CICS storage areas, with the exception of the Terminal Control Table terminal entry (TCTTE), consist of two logical and unique sections. The control section is used primarily by CICS; the user's section is defined and used exclusively by application programs. This logical division always exists except for the TCTTE, whether the storage is statically created (for example, the Common System Area) or dynamically acquired (for example, a Terminal Input/Output Area).

CICS provides a set of symbolic storage definitions (dummy sections) to describe the layout of the control section of all the applicable storage areas. These storage definitions are contained in the CICS libraries and, when combined with a user-defined layout of the user's section of the storage areas, provide symbolic addressing to the storage areas.

The Storage Accounting field is perhaps the most important field in the control section of the CICS storage areas. (See "Storage Accounting Area" below.) This field, present in all CICS storage areas except the CSA and TCTTE, is always located in the first eight bytes of every storage area.

Note: The application programmer must be aware that the Storage Accounting field exists in all dynamic storage he acquires, and he must take particular care not to alter or destroy the information in it. If the information is altered or destroyed, an abnormal termination of CICS occurs.

Two of the control areas, the Common System Area (CSA) and the Task Control Area (TCA), are required to be symbolically defined in every application program; the third control area (TCTTE), the work areas, and the I/O areas are selected at the option of the user. It is the user's responsibility to copy symbolic storage definitions into his program for the required control areas as well as for any of the other storage areas he requires. Figure 6 lists the CICS storage areas, indicating which are control areas, which are work areas, and which are I/O areas; it also indicates which are acquired by the user and which are acquired by CICS.

	CONTROL AREAS	WORK AREAS	I/O AREAS	ACQ'D BY USER	ACQ'D BY CICS
Common System Area (CSA)	X	X			X
Task Control Area (TCA)	X				X
Transaction Work Area (TWA)		X			X
File Work Area (FWA)		X			X
Storage Accounting Area (SAA)		X		X	
Terminal I/O Area (TIOA)			X	X	X
Transient Data Input Area (TDIA)			X		X
Transient Data Output Area (TDOA)			X	X	
Temporary Storage I/O Area (TSIOA)			X	X	X
File I/O Area (FIOA)			X		X
Terminal Control Table Terminal Entry (TCTTE)	X				X

Figure 6. CICS storage areas

Depending on the programming language used, one of the following statements is required to copy a symbolic storage definition into an application program.

1. Assembler language COPY statement of the form:

   COPY name

2. ANS 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 (name);

In addition to copying the appropriate symbolic storage definitions into his program, the application programmer must establish addressability for these storage definitions. He does this by specifying a symbolic base address for each storage area, thereby effectively mapping the symbolic storage definition over the storage area. Depending on the programming language used, one of the following statements must be used to establish addressability after the dynamic main storage has been acquired:

1. Assembler language statement of the form:

   L symbolic base address register,TCASCSA

2. ANS COBOL statement of the form:

   MOVE TCASCSA TO symbolic base address.

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

   symbolic base address=TCASCSA;

TCASCSA is a four-byte field that contains the address of the dynamic main storage area that was acquired.

Figure 7 contains the symbolic names used in copying the storage area control section definitions and the symbolic base addresses used in establishing addressability.

CICS STORAGE AREA	ABBREVIATION	SYMBOLIC NAMES FOR DEFINED STORAGE	BASE LOCATOR OR BASE ADDRESS REGISTER	ASSEMBLER LANGUAGE GENERAL PURPOSE REGISTER ASSIGNMENT
Common System Area	CSA	DFHCSADS	CSACBAR	13
Common Work Area	CWA	User defined	CSACBAR	13
Transaction Control Area	TCA	DFHTCADS	TCACBAR	12
Transaction Work Area	TWA	User defined	TCACBAR	12
File Work Area	FWA	DFHFWADS	FWACBAR	*
Storage Accounting Area	SAA	DFHSAADS	SAACBAR	*
Transaction Input/Output Area	TIOA	DFHTIOA	TIOABAR	*
File Input/Output Area	FIOA	DFHFIOA	FIOABAR	*
Transient Data Input Area	TDIA	DFHTDIA	TDIABAR	*
Transient Data Output Area	TDOA	DFHTDOA	TDOABAR	*
Temporary Storage Input/Output Area	TSIOA	DFHTSIOA	TSIOABAR	*
Terminal Control Table Terminal Entry	TCTTE	DFHTCTTE	TCTTEAR	*
Application Program Storage	–	–	User defined	*

* Any register except 12, 13, and 14 which are utilized by CICS.

Figure 7. Symbolic names and base addresses of CICS storage areas

All storage that is acquired by the application program through the CICS Storage Management facility is controlled by a technique that chains together all storage associated with a particular transaction. (See the section "Storage Accounting Area".) This feature allows CICS to release all main storage associated with a transaction, either upon request from the user or when the transaction is terminated, normally or abnormally.

The Common System Area (CSA) is the head of the chain, the address of which is provided by CICS. The CSA points to the Task Control Area (TCA) which in turn points to several of the other storage areas. Figure 8 illustrates the chaining of CICS storage areas and indicates the symbolic base address used to locate each storage area.

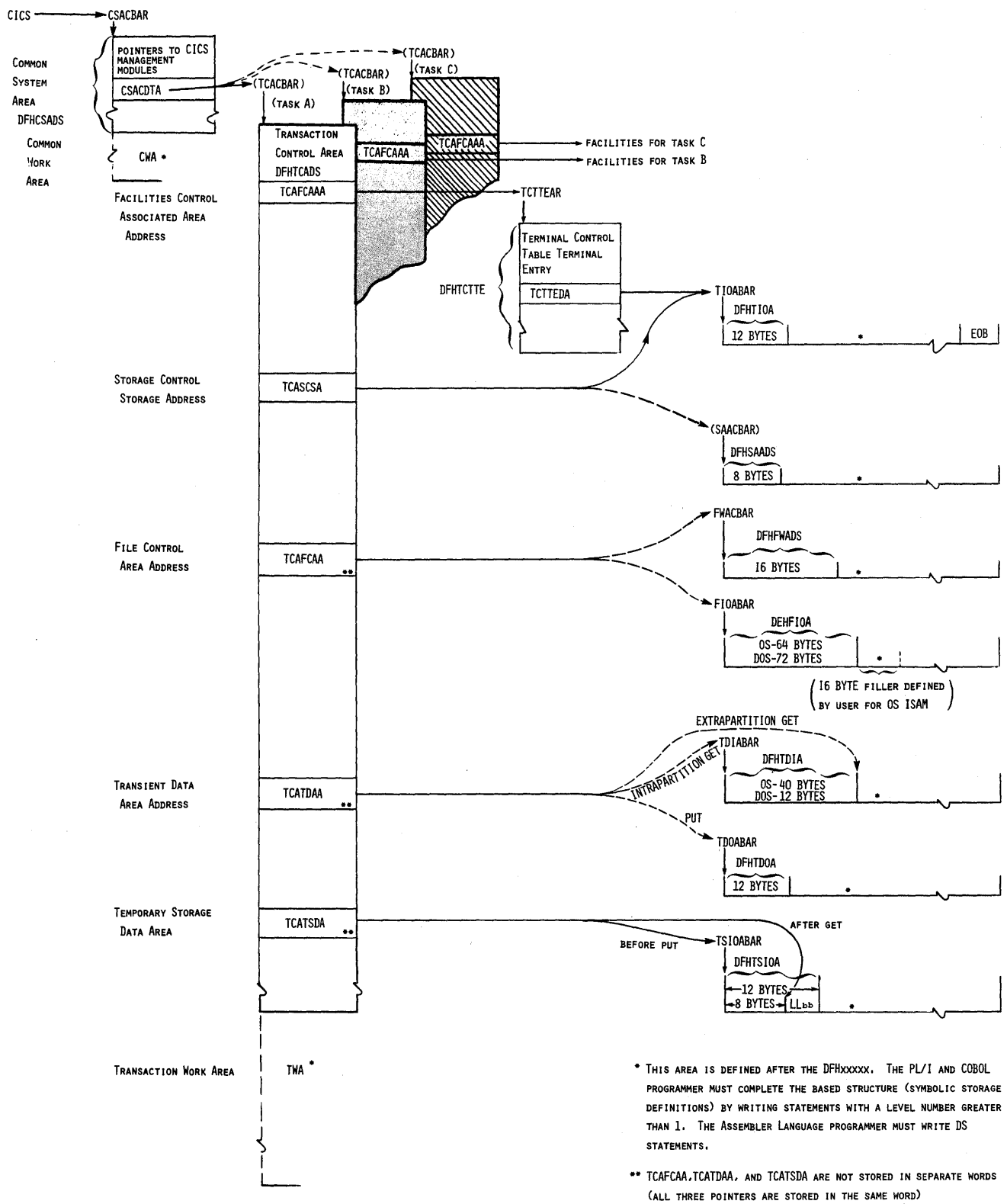


Figure 8. CICS storage areas are chained together

The following sections describe the major CICS storage areas. The fields of special significance for the application programmer are discussed in detail.

COMMON SYSTEM AREA (CSA)

The CSA is an area of static storage that contains areas and data required for the operation of CICS. It also contains a user-defined Common Work Area (CWA) that can be used at the discretion of the user for the retention of temporary data, for the accumulation of statistics, for the passing of parameters, etc.; this work area can be accessed or altered by any number of tasks.

Since the work area of the CSA 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 while it is requesting CICS services (for example, File services). Under these circumstances, another transaction might get control and possibly destroy the data. However, if the user has designed his application programs so they are all aware of a common, user-established format within the CSA work area, there is no reason why the work area cannot be shared by several tasks. An example of this might be a statistics accumulator that is updated by more than one transaction.

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

1. CICS save areas
2. Addresses of CICS management programs
3. Control system and user statistics accumulators
4. Addresses of CICS system control tables
5. Common system constants
6. System control parameters

The fields of the CSA that are of particular significance to the application programmer are as follows.

CSACTODB: This four-byte binary field contains the time of day in hundredths of a second. The time of day is updated periodically during task dispatching, its accuracy depending upon the task mix and frequency of task switching occurrences.

CSATODP: This four-byte field of the form HHMMSSt+ contains the time of day in packed decimal format to tenths of a second. The time of day is updated periodically during task dispatching, its accuracy depending upon the task mix and frequency of task switching occurrences.

CSAWAPA: This field represents the beginning of the Common Work Area (CWA) and provides doubleword storage alignment for it. The entire work area is initially set to binary zeros. The size of the work area is determined by the user at system generation time.

TASK CONTROL AREA (TCA)

The TCA is an area of main storage acquired dynamically by CICS when the task (transaction) is originated by Task Control. It is used to represent the current status of the task and is part of a chain of TCA's organized by dispatching priority. During the execution of the task, the user has the capability of changing the priority through Task Management services; the TCA is then repositioned accordingly.

The TCA provides the following for its associated task:

1. Register save areas
2. Unique fields (parameter areas) for communicating requests to CICS
3. Address of the related Facility Control Area (FCA)
4. Task storage chain addresses

When a task is initiated in CICS, the TCA exists until the task is terminated. The TCA provides no space for any 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 the section "Transaction Work Area".)

The TCA consists of three logical sections:

1. CICS system control section
2. Communication section
3. Transaction Work Area (optional)

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

The communication section is used by CICS and by user-written application programs for communication between the task and CICS management programs and service programs.

The optional Transaction Work Area is reserved for the exclusive use of the task.

In those cases where a task is initiated from a terminal (nearly always the case), CICS places in 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 Terminal Input/Output Area (TIOA). The TCA also contains the address of either a Single Event Control Block or the address of an Event Control Block list.

The fields of the TCA that are of particular significance to the application programmer are as follows.

TCAFCAAA: This four-byte field contains the address of the Facility Control Area associated with the facility that initiated the transaction. This field can contain the address of a Terminal Control Table terminal entry, the address of a Destination Control Table entry, or the address of an automatic task initiator control area.

If the user's application program is to communicate with the terminal, TCAFCAAA must contain the address of the appropriate Terminal Control Table terminal entry (TCTTE). This allows the user's application program to reference any data in the TCTTE.

TCAPCPI: This eight-byte field contains the identification of the requested program. The program identification is left-justified and must meet whatever requirements there are for a label used in a library of the operating system.

There must be an entry in the Processing Program Table (PPT) containing the program identification. This field (TCAPCPI) can be filled prior to issuing a DFHPC TYPE=XCTL, DFHPC TYPE=LINK, DFHPC TYPE=LOAD, or DFHPC TYPE=DELETE macro instruction. If the user's application program places the program identification in TCAPCPI prior to the execution of the macro instruction, the PROGRAM=name operand should be omitted from the macro instruction.

The program identification can be placed in the TCAPCPI field prior to issuing a DFHPC TYPE=LINK macro instruction when an application program is testing to determine to which program to link. On the basis of the test, the application program should place the program identification in the TCAPCPI field and then execute a DFHPC TYPE=LINK macro instruction without the PROGRAM=name operand. Using this technique, the user's application program uses one macro instruction to link to many different programs.

TCAPCAC: This four-byte field contains the termination code for the DFHPC TYPE=ABEND macro instruction. The termination code must be left-justified and must be the user's termination code. The field can be filled by the user's program prior to issuing the DFHPC TYPE=ABEND macro instruction; in this case, the ABCODE=YES operand must be coded.

The termination code is placed in the TCAPCAC field prior to the execution of the DFHPC TYPE=ABEND macro instruction when the user's application program is testing to determine which type of termination is desired.

TCASCSA: This four-byte field contains the address of the storage obtained after the execution of a DFHSC TYPE=GETMAIN macro instruction and must also contain the address of the storage to be released prior to the execution of a DFHSC TYPE=FREEMAIN macro instruction. The application programmer must remember that the first eight bytes at this address are always the Storage Accounting Area used by CICS Storage Management. Care should be taken never to alter the contents of this area.

The address of the storage obtained from a DFHSC TYPE=GETMAIN macro instruction is automatically placed in the TCASCSA field except when a conditional GETMAIN request (COND=YES) has been issued and storage is not available. In this case, CICS Storage Management places binary zeros in this field and returns control to the user. The user's application program must specify a symbolic base address for the storage area and must move the storage address located at TCASCSA to this symbolic base address. The following are examples of the coding required.

For Assembler language:

```
WORK DSECT
     USING *,5
        .
        .
        .
STATE DS CL3
        .
        .
```

```
        CSECT
        BALR 10,0
        USING *,10
            .
            .
            .
        DFHSC TYPE=GETMAIN
        L 5,TCASCSA
```

For ANS COBOL:

```
       LINKAGE SECTION.
                .
                .
                .
           02 WORKREG PICTURE S9(8) USAGE IS COMPUTATIONAL.
                .
                .
                .
       01 WORK.
                .
                .
                .
           02 STATE PICTURE XXX.
                .
                .
                .
       PROCEDURE DIVISION.
           MOVE CSACDTA TO TCACBAR.
                .
                .
                .
       DFHSC TYPE=GETMAIN
           MOVE TCASCSA TO WORKREG.
```

For PL/I:

```
       DECLARE 1 WORK BASED (WORKREG),
                .
                .
                .
                  2 STATE CHAR(3);
                .
                .
                .
       DFHSC TYPE=GETMAIN
       WORKREG=TCASCSA;
```

When storage is to be released, the storage address must be placed in the TCASCSA field prior to the execution of the DFHSC TYPE=FREEMAIN macro instruction. The following are examples of the coding required.

For Assembler language:

```
       WORK DSECT
            USING *,5
                .
                .
                .
       STATE DS CL5
                .
```

```
          .
          .
        CSECT
        BALR 10,0
        USING *,10
          .
          .
          .
        ST 5,TCASCSA
        DFHSC TYPE=FREEMAIN
```

For ANS COBOL:

```
       LINKAGE SECTION.
                .
                .
                .
           02 WORKREG PICTURE S9(8) USAGE IS COMPUTATIONAL.
                .
                .
                .
       01  WORK.
                .
                .
                .
           02 STATE PICTURE XXX.
                .
                .
                .
       PROCEDURE DIVISION.
           MOVE CSACDTA TO TCACBAR.
                .
                .
                .
           MOVE WORKREG TO TCASCSA.
       DFHSC TYPE=FREEMAIN
```

For PL/I:

```
       DECLARE 1 WORK BASED (WORKREG),
                .
                .
                .
                  2 STATE CHAR(3);
                .
                .
                .
       TCASCSA=WORKREG;
       DFHSC TYPE=FREEMAIN
```

TCADCNB: This two-byte field contains the length (in bytes) of the main storage area to be dumped by Dump Control. This field can be filled by the user's application program, prior to execution of the DFHDC TYPE=PARTIAL macro instruction, with a hexadecimal representation of the number of bytes requested.

TCASCNB: This two-byte field contains the number of storage bytes requested. This field can be filled by the user's application program with a hexadecimal representation of the number of bytes requested prior to execution of the DFHSC TYPE=GETMAIN macro instruction. If the user's application program places a value in this field prior to

the execution of a DFHSC TYPE=GETMAIN macro instruction, the NUMBYTE=value operand must be omitted. When the storage is obtained, the TCASCNB is overlaid with a portion of the address of the storage obtained.

TCASCIB: This one-byte field contains the bit configuration for the initialization of main storage. The field can be filled by the user's application program with the desired bit configuration prior to the execution of a DFHSC TYPE=GETMAIN macro instruction, in which case the INITIMG=YES operand must be coded.

TCAFCDI: This eight-byte field contains the data set identification for the data set to which a record is to be written or from which a record is to be retrieved. The user's application program can place the data set identification in this field prior to the execution of a DFHFC TYPE=GET or a DFHFC TYPE=SETL macro instruction.

The data set identification must correspond exactly with the user-established identification of the required data set (as previously established in the File Control Table) and must be left-justified when the user's application program places the identification in the TCAFCDI field. If this field is filled prior to the execution of the DFHFC macro instruction, the DATASET=name operand must be omitted.

TCAFCRI: This four-byte field contains the address of the user's record identification field when making a request for CICS File Management services. The user's application program can place the address in this field prior to the execution of a DFHFC TYPE=GET, DFHFC TYPE=PUT, DFHFC TYPE=SETL, or DFHFC TYPE=GETNEXT macro instruction. The RIDADR=symbol operand is omitted if the TCAFCRI field is filled prior to the execution of the macro instruction.

TCAFCSI: This eight-byte field contains the segment set identification. The user's application program can place the segment set identification in this field prior to issuing the DFHFC TYPE=GET, DFHFC TYPE=PUT, DFHFC TYPE=SETL, or DFHFC TYPE=GETNEXT macro instruction.

The segment set identification must match the user-established identification of the requested segment set (as previously established in the File Control Table) and must be left-justified when the user's application program places the identification in the TCAFCSI field. If this field is filled prior to execution of the DFHFC TYPE=GET or DFHFC TYPE=PUT macro instruction, the SEGSET=YES operand must be coded as part of that macro instruction.

TCAFCAI: This eight-byte field contains the symbolic identification of the first index data set to be searched in an indirect accessing hierarchy. The user's application program can place the desired indirect access identification (as previously established in the File Control Table) in the field prior to the execution of a DFHFC TYPE=GET macro instruction. When the user's application program places the identification in the TCAFCAI field, it must be left-justified and the INDEX=YES operand must be coded as part of the macro instruction.

TCAFCAA: This four-byte field contains the address of the File Input/Output Area (FIOA) or File Work Area (FWA).

TCAFCTR: This one-byte field contains the type of File Control request/response. Request codes are set by issuing the DFHFC macro instruction. Responses are automatically placed in the TCAFCTR field by File Management after completion of the event requested.

TCATDTR: This one-byte field contains the type of Transient Data Control request/response. Request codes are set by issuing the DFHTD macro instruction. Responses are automatically placed in the TCATDTR by Transient Data Management field after completion of a transient data event.

TCATSTR: This one-byte field contains the Temporary Storage Control request/response. Request codes are set by issuing the Temporary Storage macro instruction DFHTS. Responses are automatically placed in the TCATSTR field by Temporary Storage Management after completion of a temporary storage event.

TCAICTR: This one-byte field contains the Interval Control request/response. Requests codes are set by issuing the Interval Control macro instruction DFHIC. Responses are automatically placed in the TCAICTR field by Time Management after completion of an Interval Control service request.

TRANSACTION WORK AREA (TWA)

The TWA is an extension of the TCA and is created at the option of the user to provide a work area for a given transaction (task).

The TWA can be used for the accumulation of data and intermediate results during the execution of the transaction. It can also be used when the amount of working storage for a transaction 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.

Where the TWA is desired for a given transaction, it is the responsibility of the application programmer to define storage for the TWA immediately following his symbolic storage definition of the TCA. The size of the TWA is specified in the Program Control Table entry for each transaction identification. Therefore, the size of the TWA can vary by transaction type according to the user's needs. For information on establishing the TWA, see the Program Control Table in the System Programmer's Reference Manual.

ASSEMBLER LANGUAGE APPLICATION PROGRAMMING

The Assembler language programmer must define storage for the CICS control areas and any other CICS storage areas required for the processing of his program. He accomplishes this by using the Assembler language COPY statement to (1) copy the appropriate symbolic storage definitions into his program and (2) specify the names of the storage areas being defined. All registers are at his disposal, except for registers 12, 13, and 14 (which are used by CICS).

STATIC STORAGE DEFINITION

During CICS initialization, the CSA is statically allocated as part of the CICS Nucleus. For each terminal with which communication is to occur, the Terminal Control Table terminal entry (TCTTE) is included in the statically allocated Terminal Control Table (TCT). The application programmer must provide symbolic storage definition for the CSA and TCTTE (if needed) by using the COPY statement in his program.

Common System Area (CSA)

The statement

```
COPY DFHCSADS
```

copies the symbolic storage definition for the CSA and assigns register 13 as the base register.

If the user has generated a CSA with a work area, he may wish to include his own symbolic definitions for that area following the COPY DFHCSADS statement. For example:

```
          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 symbolic definition is necessary when the user desires to obtain the address of the terminal I/O area (TCTTEDA) or to request a Terminal Control service via the DFHTC macro instruction. The user must code an EQU statement prior to the COPY statement to set up a base register for the TCTTE, equating the label TCTTEAR to a program register. The following is an example of the coding required:

```
TCTTEAR EQU  5
        COPY DFHTCTTE
```

DYNAMIC STORAGE DEFINITION

During initiation and execution of a transaction (task), the TCA, TIOA, and other storage areas required by the transaction are dynamically allocated by CICS Storage Management, either upon request from the application program or upon request from a CICS management function. The application programmer must provide symbolic storage definition for these storage areas by using the COPY statement in his program.

Task Control Area (TCA)

The statement

```
COPY DFHTCADS
```

copies the symbolic storage definition for the TCA (excluding the CICS control section) and assigns register 12 as the base register. If the user's application program uses a Transaction Work Area (TWA), DS statements for that storage area 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:

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

If it is necessary for the Assembler language programmer to obtain access to the CICS system control section of the TCA, a copy of the symbolic storage definition for the entire TCA may be obtained by using the statement

```
DFHTCA CICSYST=YES
```

in place of the statement COPY DFHTICADS. Addressability to the communication section of the TCA and to the Transaction Work Area (TWA) is provided automatically by CICS through register 12. Addressability to the CICS system control section must be provided by the application programmer; for example:

```
L     WRKREG,TCASYAA
USING DFHSYTCA,WRKREG
.
.
.
DROP  WRKREG
```

Terminal Input/Output Area (TIOA)

The statement

```
COPY DFHTIOA
```

copies the symbolic storage definition for the CICS control section of the TIOA. It is desirable that this storage definition precede the user's definition of a terminal input or output message. The user must code an EQU statement prior to the COPY statement to set up a base register for the TIOA, equating the label TIOABAR to a program register. The following is an example of the coding required:

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

File Input/Output Area (FIOA)

The statement

```
COPY DFHFIOA
```

copies the symbolic storage definition for the CICS control section of the FIOA. This storage definition should precede the user's defined layout of a file input or output area when reading an unblocked record without updating or segmenting, when reading blocked records without deblocking, or when checking response codes for the appropriate abnormal response. The user must code an EQU statement prior to the COPY statement to set up a base register for the FIOA, equating the label FIOABAR to a program register. The FIOA is automatically acquired by File Management whenever a request is made by the user to access a data base data set. If ISAM data is being retrieved under CICS/OS, a 16-byte filler must be defined prior to the user's data definition. The following is an example of the coding required:

```
FIOABAR EQU   7
        COPY  DFHFIOA
        DS    16X                OS ISAM FILLER
NAME    DS    CL20
STREET  DS    CL5
```

File Work Area (FWA)

The statement

```
COPY DFHFWADS
```

copies the symbolic storage definition for the CICS control 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 or segmented record, when adding a new record to a file, or when retrieving records using the browse feature. The user must code an EQU statement prior to the COPY statement to set up a base register for the FWA, equating the label FWACBAR to a program register. The following is an example of the coding required:

```
FWACBAR EQU   7
        COPY  DFHFWADS
NAME    DS    CL20
STREET  DS    CL30
ZIPCODE DS    CL5
```

Transient Data Input Area (TDIA)

The statement

```
COPY DFHTDIA
```

copies the symbolic storage definition for the CICS control section of the intrapartition TDIA. It is desirable that this storage definition precede the user's defined layout of the message area used for a transient data GET. The user must code an EQU statement prior to the COPY statement to set up a base register for the TDIA, equating the label TDIABAR to a program register. The following is an example of the coding required:

```
TDIABAR EQU   9
        COPY  DFHTDIA
NAME    DS    CL20
```

```
STREET   DS    CL20
         .
         .
         .
```

Transient Data Output Area (TDOA)

The statement

```
COPY DFHTDOA
```

copies the symbolic storage definition for the CICS control section of the intrapartition TDOA. For consistent documentation of the user's application program, this storage definition should precede the user's defined layout of the message area for a transient data PUT. The user must code an EQU statement prior to the COPY statement to set up a base register for the TDOA, equating the label TDOABAR to a program register. The address of the length field labeled TDOAVRL is given to Transient Data Control either through the TDADDR operand or by placing it in the TCA at 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=XX
        L     TDOABAR,TCASCSA
        .
        .
        .
        DFHID TYPE=PUT,DESTID=POST,TDADDR=TDOAVRL
```

Temporary Storage Input/Output Area (TSIOA)

The statement

```
COPY DFHTSIOA
```

copies the symbolic storage definition for the CICS control section of the TSIOA. This storage definition should precede the user's defined layout of the input/output work areas for temporary storage. The user must code an EQU statement prior to the COPY statement to set up a base register for the TSIOA, equating the label TSIOABAR to a program register. The address of the length field labeled TSIOAVRL is given to Temporary Storage Control either through the TSDADDR=parameter operand of the DFHTS macro instruction or by placing it in the TCA at 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     TSICABAR,TCATSDA
         SH    TSIOABAR,=H'8'
```

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 he will use within his application program. The user must code an EQU statement prior to the COPY statement to set up a base register for the SAA, equating the label SAACBAR to a program 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
         .
         .
         .
```

EXAMPLE OF CICS ASSEMBLER LANGUAGE APPLICATION PROGRAM

Figure 9 illustrates an Assembler language program written to run under CICS. The program issues four CICS macro instructions, asks a question of the terminal operator, receives a reply, dynamically acquires some storage, and sends the operator's message back to the terminal. (The line numbers are not part of the program.)

```
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    CL40
08           COPY  DFHTCTTE
09           COPY  DFHTIOA
10  MESSG    DS    CL40
11           CSECT
12           BALR  BASEREG,0
13           USING *,BASEREG
14           L     TCTTEAR,TCAFCAAA
15           L     TIOABAR,TCTTEDA
16           MVC   MESSG,=C'WHAT LANGUAGE AM I CODED IN'
17           MVC   TIOATDL,=H'27'
18           DFHIC TYPE=(WRITE,READ,WAIT)
19           L     TIOABAR,TCTTEDA
20           MVC   LENGTH,TIOATDL
21           MVC   MESSAGE,MESSG
22           DFHSC TYPE=GETMAIN,                                    *
23                 CLASS=TERMINAL,                                  *
24                 INITIMG=40,                                      *
25                 NUMBYTE=40
26           L     TIOABAR,TCASCSA
27           ST    TIOABAR,TCTTEDA
28           MVC   MESSG,MESSAGE
29           MVC   TIOATDL,LENGTH
30           DFHIC TYPE=WRITE
31           DFHPC TYPE=RETURN
32           LTORG
33           END
```

Figure 9. Example of CICS Assembler language application program

A discussion of the significance of each of the lines of Figure 9 follows.

STATEMENT NUMBER	DESCRIPTION
01	Assigns base register for program.
02-03	Assigns base registers 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-reentrance.
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 instruction 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.

STATEMENT NUMBER	DESCRIPTION
20-21	Saves the message and the length of the message in the TWA save areas.
22-25	CICS macro instruction that requests 40 bytes of terminal type storage initialized to blanks.
26	Establishes addressability for new TIOA (address of newly-acquired storage area is in TCASCSA field of the TCA).
27	Places address of new storage area in TCTTE.
28-29	Moves the message and the length of the message from TWA save areas.
30	CICS macro instruction that writes message to terminal.
31	CICS macro instruction that returns control to CICS and terminates this task.
32-33	Required for Assembler language.

ANS COBOL APPLICATION PROGRAMMING

The application programmer who programs in ANS COBOL must define storage for the control areas and any other storage areas required for the processing of his program. He accomplishes this (1) by use of the COPY statement in the Linkage Section of the Data Division to copy the symbolic storage definitions into his program and specify the names of the storage areas being defined and (2) by use of the MOVE statement in the Procedure Division to establish addressability through the moving of symbolic storage addresses from one location to another.

The programmer uses normal ANS COBOL code with the exception that (1) CICS macro instructions must be used to invoke CICS services and (2) the unique storage areas provided by or acquired through CICS should be used for the retention of data. The Working Storage section of an ANS COBOL program should only be used to contain data constants. Variable data should be placed in the CICS Transaction Work Area (TWA) or in an area of main storage acquired via the DFHSC TYPE=GETMAIN macro instruction.

In the CICS/DOS-ENTRY system, a fresh copy of the program is used each time the task is rolled back in after a rollout. Therefore, any data fields established in the program before rollout occurs must be reestablished after subsequent rollin.

See the section "Supervisory and Data Management Services" for a listing of ANS COBOL features that may not be used.

The statement

```
01 DFHBLLDS COPY DFHBLLDS.
```

must be the first statement in the Linkage Section of the Data Division. This statement copies the symbolic storage definition for the Linkage Section Base Locator (BLL) which provides the means whereby an ANS COBOL program can request dynamically acquired CICS storage areas. Included in this definition is the symbolic base address for the CSA and TCA.

If the ANS COBOL programmer desires to use CICS storage areas other than the CSA and TCA, immediately following the COPY statement for the BLL he must code statements of the form

```
02 name PICTURE S9(8) USAGE IS COMPUTATIONAL.
```

where "name" is the symbolic base address used to locate a specific storage area. These 02 statements must be coded in the same order as the corresponding 01 statements coded subsequently.

If the user is going to communicate with the system via a terminal, he needs a Terminal Input/Output Area (TIOA) and a Terminal Control Table terminal entry (TCTTE). The following is an example of the coding required in the Linkage Section of the Data Division:

```
01 DFHBLLDS COPY DFHBLLDS.
   02 TCTTEAR PICTURE S9(8) USAGE IS COMPUTATIONAL.
   02 TIOABAR PICTURE S9(8) USAGE IS COMPUTATIONAL.
01 DFHCSADS COPY DFHCSADS.
01 DFHTCADS COPY DFHTCADS.
01 DFHTCTTE COPY DFHTCTTE.
01 DFHTIOA COPY DFHTIOA.
```

If the user wishes to access a series of chained storage areas (areas that contain a pointer to the next area in the chain), he must establish addressability to each new storage area in the chain by inserting a paragraph name immediately following any MOVE statement that establishes addressability but prior to the next sequential statement. For example:

```
LINKAGE SECTION.
01 DFHBLLDS COPY DFHBLLDS.
   .
   .
   .
   02 USERPTR PICTURE S9(8) USAGE IS COMPUTATIONAL.
   .
   .
   .
01 DFHTCADS COPY DFHTCADS.
   02 TWAFIELD PICTURE X(4).
   .
   .
   .
01 USERAREA.
   02 FIELD PICTURE X(4).
   02 NEXTAREA PICTURE S9(8) USAGE IS COMPUTATIONAL.
   .
   .
   .
PROCEDURE DIVISION.
   .
   .
   .
   MOVE NEXTAREA TO USERPTR.
ANYNAME.
   MOVE FIELD TO TWAFIELD.
```

In this example, storage areas are chained, each of which is mapped or defined by USERAREA. The first MOVE instruction establishes addressability to the next area in the chain. The second MOVE instruction moves data from the newly addressed area, but only because the paragraph name precedes the second MOVE instruction; in the absence of the paragraph name, data is moved from the previously addressed area rather than from the new area. Note that a paragraph name is not needed if addressability to an area is obtained via a field in some other area (for example, the TCA).

If the object of an "OCCURS DEPENDING ON" clause is defined in the linkage section, special consideration is required to ensure the correct value is used at all times. In the following example, FIELD-COUNTER is defined in the linkage section and if the MOVE FIELD-COUNTER TO

FIELD-COUNTER statement is missing, unpredictable results will occur when referencing DATA.

```
       LINKAGE SECTION.
       01 DFHFWADS COPY DFHFWADS.
          .
          .
          .
          02 FIELD-COUNTER PIC 9(4) USAGE IS COMPUTATIONAL.
          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 extra MOVE statement to FIELD-COUNTER causes COBOL to re-establish the value it uses to compute the current number of occurrences of FIELDS and therefore, can correctly determine the displacement of DATA.

An area defined in the Linkage Section that is greater than 4095 bytes in length requires special consideration. Required are an extra 02-level statement under DFHBLLDS and an extra statement to establish

addressability. For example, if a FWA (File Work Area) exceeds 4095 bytes, the following is an example of the code required:

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

If an application program is to be compiled under CICS/OS using the Full ANS COBOL V4 Compiler (5734-CB2) with the optimization (OPT) feature, a special compiler control statement must be inserted at appropriate places within the program to obtain addressability to a particular area of main storage. This control statement has the form:

```
SERVICE RELOAD fieldname.
```

where "fieldname" is the symbolic name of a specific storage area, and where "fieldname" is also defined in an 01-level statement in the Linkage Section. The first two statements of the Procedure Division must always be

```
SERVICE RELOAD DFHBLLDS.
SERVICE RELOAD DFHCSADS.
```

Statements such as:

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

or

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

might be used to establish addressability for a particular storage area. (Note that the SERVICE RELOAD statement must always be used.)

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

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

Note that the RELOAD statement specifies DFHTCA, not DFHTCADS.

If the application program is to use the Data Language/I (DL/I) facilities of CICS/OS as well as the V4 ANS COBOL Compiler, the first four statements of the Procedure Division must be

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

STATIC STORAGE DEFINITION

During CICS initialization, the Common System Area (CSA) is statically allocated as part of the CICS Nucleus. For each terminal with which communication is to occur, the Terminal Control Table terminal entry (TCTTE) is included in the statically allocated Terminal Control Table (TCT). The ANS COBOL programmer must provide symbolic storage definition 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 the user has appended a Common Work Area (CWA) to the CSA, immediately following the COPY statement in the Linkage Section he must define the record layout of the CWA. The following is an example of the coding required:

```
01 DFHCSADS COPY DFHCSADS.
   02 CWA PICTURE X(400).
      03 FIELD1 PICTURE 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.

DYNAMIC STORAGE DEFINITION

During initiation and execution of a transaction (task), the Task Control Area (TCA), the Terminal Input/Output Area (TIOA), and other storage areas required by the transaction are dynamically allocated by CICS. The ANS COBOL programmer must provide symbolic storage definition for these storage areas as follows.

Task Control Area (TCA)

The statement

```
01 DFHTCADS COPY DFHTCADS.
```

copies the symbolic storage definition for the TCA. The user must code the statement

```
MOVE CSACDTA TO TCAEAR.
```

as the first statement in the Procedure Division to establish addressability for the TCA.

If the user desires to append a Transaction Work Area (TWA) to the TCA, immediately following the COPY statement in the Linkage Section he must define the record layout of the TWA. The following is an example of the coding required:

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

Terminal Input/Output Area (TIOA)

The statement

```
01 DFHTICA COPY DFHTIOA.
```

copies the symbolic storage definition for the CICS control section of the TICA and must be present in all programs that use terminal input records or that 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 PICTURE XXXX.
   02 TICAMSG PICTURE X(20).
   .
   .
   .
```

The user must establish addressability for the TIOA in the Procedure Division of his program by coding in the appropriate place either the statement

```
MOVE TCTTEDA TO TICAEAR.
```

or the statement

```
MOVE TCASCSA TO TICAEAR.
```

The latter statement is used to establish addressability for a new TIOA acquired dynamically through use of a DFHSC TYPE=GETMAIN macro instruction and should be coded immediately following the last operand of that macro instruction.

File Input/Output Area (FIOA)

The statement

```
01 DFHFIOA COPY DFHFIOA.
```

copies the symbolic storage definition for the CICS control section of the FIOA and must be present in all programs requesting a "read without update" for an unblocked, unsegmented data set. If ISAM data is being retrieved under CICS/OS, a 16-byte filler must be defined prior to the user's data definition. The following is an example of the coding required to define the record(s) in the FIOA:

```
01 DFHFIOA COPY DFHFIOA.
   02 FILLER PICTURE X(16).          NOTE OS ISAM FILLER.
   02 KEY PICTURE X(6).
   02 NAME PICTURE X(20).
   02 FIOAREC PICTURE X(74).
   .
   .
   .
```

The user must code the statement

```
MOVE TCAFCAA TO FIOAEAR.
```

in the appropriate place in the Procedure Division of his program to establish addressability for the FIOA.

File Work Area (FWA)

The statement

```
01 DFHFWADS COPY DFHFWADS.
```

copies the symbolic storage definition for the CICS control section of the FWA and must be present in all programs performing file activity with the exception of a "read without update" from an unblocked, unsegmented data set. The following is an example of the coding required to define the record(s) in the FWA:

```
01 DFHFWADS COPY DFHFWADS.
   02 KEY PICTURE X(6).
   02 NAME PICTURE X(20).
   02 FWAREC PICTURE X(24).
   .
   .
   .
```

The user must code the statement

```
MOVE TCAFCAA TO FWACEAR.
```

in the appropriate place in the Procedure Division of his program to establish addressability for the FWA.

Transient Data Input Area (TDIA)

The statement

```
01 DFHTDIA COPY DFHTDIA.
```

copies the symbolic storage definition for the CICS control section of the intrapartition TDIA and must be present in all programs requesting a GET for transient data. The following is an example of the coding required to define the record(s) in the TDIA:

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

The user must code the statement

```
MOVE TCAIDAA TO TDIABAR.
```

in the appropriate place in the Procedure Division of his program 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 control section of the intrapartition TDOA and should be present in all programs requesting a PUT to transient data. The following is an example of the coding required to define the record(s) in the TDOA:

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

The user must code the statement

```
MOVE TCASCSA TO TDOABAR.
```

in the appropriate place in the Procedure Division of his program 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 control 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 the record(s) in the TSIOA:

```
01 DFHTSIOA COPY DFHISIOA.
   02 DATA PICTURE X(10).
```

To establish addressability for the TSIOA, the user must code in the appropriate place in the Procedure Division of his program the statements

```
MOVE TCAISDA TO TSIOABAR.
SUBTRACT 8 FROM TSICABAR.
```

if the request is a GET from temporary storage, or the statement

```
MOVE TCASCSA TO TSIOABAR.
```

if the request is a PUT to temporary storage and the user has just dynamically acquired an I/O area. In the case of a PUT, the symbolic address of the data is located at TSIOAVRL.

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 instruction. The following is an example of the coding required to define the record(s) in the SAA:

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

The user must code the statement

```
MOVE TCASCSA TO SAACBAR.
```

in the appropriate place in the Procedure Division of his program to establish addressability for the SAA.

EXAMPLE OF CICS ANS COBOL APPLICATION PROGRAM

Figure 10 illustrates an ANS COBOL program written to run under CICS. The program issues four CICS macro instructions, asks a question of the terminal operator, receives a reply, dynamically acquires some storage, and sends the operator's message back to the terminal. (The line numbers are not part of the program.)

```
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 PICTURE S9(8) USAGE IS COMPUTATIONAL.
09         02 TIOABAR PICTURE S9(8) USAGE IS COMPUTATIONAL.
10      01 DFHCSADS COPY DFHCSADS.
11      01 DFHTCADS COPY DFHTCADS.
12         02 SAVE-LENGTH PICTURE S9(8) USAGE IS COMPUTATIONAL.
13         02 SAVE-MESSAGE PICTURE X(40).
14      01 DFHTCTTE COPY DFHTCTTE.
15      01 DFHTIOA COPY DFHTIOA.
16         02 TIOAMSG PICTURE X(40).
17      PROCEDURE DIVISION.
18         MOVE CSACDTA TO TCACBAR.
19         MOVE TCAFCAAA TO TCTTEAR.
20         MOVE TCTTEDA TO TIOABAR.
21         MOVE 'IS THIS A COBOL OR A PL/I PROGRAM' TO TIOAMSG.
22         MOVE 33 TO TIOATDL.
23      DFHTC TYPE=(WRITE,READ,WAIT)
24         MOVE TCTTEDA TO TIOABAR.
25         MOVE TIOATDL TO SAVE-LENGTH.
26         MOVE TIOAMSG TO SAVE-MESSAGE.
27      DFHSC TYPE=GETMAIN,                                          *
28            NUMBYTE=40,                                           *
29            INITIMG=40,                                           *
30            CLASS=TERMINAL
31         MOVE TCASCSA TO TIOABAR.
32         MOVE TIOABAR TO TCTTEDA.
33         MOVE SAVE-MESSAGE TO TIOAMSG. NOTE MOVE MSG TO I/O AREA.
34         MOVE SAVE-LENGTH TO TIOATDL.
35      DFHTC TYPE=WRITE
36      DFHPC TYPE=RETURN
```

Figure 10. Example of CICS ANS COBOL application program

A discussion of the significance of each of the lines of Figure 10 follows.

STATEMENT NUMBER	DESCRIPTION
01-05	Required for ANS COBOL.
06	Start of Linkage Section.
07	Copies symbolic storage definition for BLL; contains addresses of CICS storage areas.
08-09	Add addresses for TCTTE and TIOA (required for statements 14 and 15).
10	Copies symbolic storage definition for CSA.
11	Copies symbolic storage definition for TCA.
12-13	Defines save areas in TWA to ensure reentrance (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 ANS COBOL (start of Procedure Division).
18-20	Establishes addressability for TCA, TCTTE, and TIOA (CICS establishes addressability for BLL and CSA).

21	Moves message to output area of TIOA.
22	Moves length of message to data length field of TIOA.
23	CICS macro instruction that writes message to terminal, waits for operator's reply, and reads operator's reply.
24	Establishes addressability for new TIOA using address in TCTTE.
25	Saves length of message in TWA.
26	Saves message in TWA.
27-30	CICS macro instruction that requests 40 bytes of terminal storage initialized to blanks (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 storage area in Terminal Control Table.
33	Moves message to output area (TIOA).
34	Moves length of message to output area.
35	CICS macro instruction that writes message to terminal.
36	CICS macro instruction that returns control to CICS.

PL/I APPLICATION PROGRAMMING

The PL/I programmer must define storage for the CICS control areas and any other CICS storage areas required for the processing of his program. He accomplishes this by using a statement of the form

```
%INCLUDE (name);
```

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

The source code provided by CICS in response to a %INCLUDE statement 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.

A PL/I program written to run under control of CICS must be written with the following considerations and restrictions:

1. Include the REENTRANT option in the initial PROCEDURE statement to satisfy the CICS requirement that code be quasi-reentrant. For example: PL1PROG: PROCEDURE OPTIONS (MAIN,REENTRANT);
2. Use CICS macro instructions to request all CICS services.
3. PL/I object modules from separate compilations cannot be link-edited into a single executable program. For subprogram linkage, use the DFHPC TYPE=LINK macro instruction.

See the section "Supervisory and Data Management Services" for a listing of PL/I features that may not be used.

STATIC STORAGE DEFINITION

During CICS initialization, the Common System Area (CSA) is statically allocated as part of the CICS Nucleus. For each terminal

with which communication is to occur, the Terminal Control Table terminal entry (TCTTE) is included in the statically allocated Terminal Control Table (TCT). The PL/I programmer must provide symbolic storage definition for the CSA and TCTTE (if needed) as follows.

Common System Area (CSA)

The statement

```
%INCLUDE (DFHCSADS);
```

copies the based structure that symbolically defines the CSA. Addressability for the CSA is included.

To define areas in the work area portion of the CSA, the PL/I programmer must provide, immediately following the %INCLUDE (DFHCSADS) macro instruction, coding such as the following:

```
DECLARE 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.

DYNAMIC STORAGE DEFINITION

During initiation and execution of a transaction (task), the Task Control Area (TCA), the Terminal Input/Output Area (TIOA), and other storage areas required by the transaction are dynamically allocated by CICS. The PL/I programmer must provide symbolic definition for these storage areas as follows.

Task Control Area (TCA)

The statement

```
%INCLUDE (DFHTCADS);
```

copies the based structure that symbolically 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 user must complete the declaration of the TCA structure by supplying a dummy 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 symbolically defines the CICS control section of the TIOA and establishes addressability. This statement must be present in all programs that use terminal input records or that output records to the terminal. The application programmer must complete the declaration of the TIOA structure by supplying a dummy ending (for example, a semicolon) or by supplying further declaration of the input/output area. The following is an example of the coding required:

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

File Input/Output Area (FIOA)

The statement

```
%INCLUDE (DFHFIOA);
```

copies the based structure that symbolically defines the CICS control section of the FIOA and must be present in all programs requesting a "read without update" for an unblocked, unsegmented data set (file). The user must complete declaration of the FIOA. He must establish addressability for the FIOA using the statement

```
FIOABAR=TCAFCAA;
```

If ISAM is being retrieved under CICS/OS, a 16-byte filler must be defined prior to the user's data definition. The following is an example of the coding required:

```
%INCLUDE (DFHFIOA);
   2 FILL CHAR(16),              /*OS ISAM FILLER*/
   2 NAME CHAR(20),
   2 ADDR CHAR(20);
   .
   .
   .
FIOABAR=TCAFCAA;
   .
   .
   .
```

File Work Area (FWA)

The statement

```
%INCLUDE (DFHFWADS);
```

copies the based structure that symbolically defines the CICS control section of the FWA. This statement should precede a user-declared file record area when reading or updating an existing blocked or segmented record, when adding a new record to a data set (file), or when retrieving records using the browse technique. The user must complete declaration of the FWA. He must establish addressability for the FWA using the statement

```
FWACBAR=TCAFCAA;
```

The following is an example of the coding required:

```
%INCLUDE (DFHFWADS);
   2 NAME CHAR(20),
   2 ADDR CHAR (20);
   .
   .
   .
FWACBAR=TCAFCAA;
   .
   .
   .
```

Transient Data Input Area (TDIA)

The statement

```
%INCLUDE (DFHTDIA);
```

copies the based structure that symbolically defines the CICS control section of the intrapartition TDIA and must be present in all programs requesting a GET for transient data. The user must complete declaration of the TDIA. He must establish addressability for the TDIA using the statement

```
TDIABAR=TCATDAA;
```

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 symbolically defines the CICS control section of the intrapartition TDOA and should be present in all programs

requesting a PUT to transient data (for consistent documentation of the user's programs). The user must complete declaration for the TDOA. He must establish addressability for the TDOA using the statement

```
TDOABAR=TCASCSA;
```

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 symbolically defines the CICS control section of the TSIOA and should be present in all programs using temporary storage. The application programmer must complete declaration for the TSIOA. He must establish addressability for the TSIOA using coding such as:

```
DCL TSIOABAA FIXED BIN(30) BASED(TSIOABAB);
    TSIOABAR=TCATSIA;
    TSIOABAB=ADDR(TSIOABAR);
    TSIOABAA=TSIOABAA - 8;
```

if the request is a GET from temporary storage, or the statement

```
TSIOABAR=TCASCSA;
```

if the request is a PUT to temporary storage, and the user has just dynamically acquired the I/O area. In the case of a PUT, the symbolic address of the data is located at TSIOAVRL.

Storage Accounting Area (SAA)

The statement

```
%INCLUDE (DFHSAADS);
```

copies the based structure that symbolically defines the SAA and should be present in all programs requesting storage through use of the DFHSC TYPE=GETMAIN,CLASS=USER macro instruction. This statement should precede the definition of user storage. The application programmer must complete declaration for the SAA. He must establish addressability for the SAA using the statement

```
SAACBAR=TCASCSA;
```

The following example illustrates the coding required:

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

EXAMPLE OF CICS PL/I APPLICATION PROGRAM

Figure 11 illustrates a PL/I program written to run under CICS. The program issues four CICS macro instructions, asks a question of the terminal operator, receives a reply, dynamically acquires some storage, and sends the operator's message back to the terminal. (The line numbers are not part of the program.)

```
01     PL1PROG: PROCEDURE OPTIONS (MAIN,REENTRANT);
02     %INCLUDE (DFHCSADS);
03     %INCLUDE (DFHTCADS);
04        2 SAVE_LENGTH BINARY FIXED (15),
05        2 SAVE_MSG CHAR (40);
06     %INCLUDE (DFHTCTTE);
07     %INCLUDE (DFHTIOA);
08        2 TICAMSG CHAR(40);
09     TICAMSG='IS THIS A COBOL OR A PL/I PROGRAM';
10     TIOATDL=33;
11     DFHTC TYPE=(WRITE,READ,WAIT)
12     TIOABAR=TCTTEDA;
13     SAVE_LENGTH=TIOATDL;
14     SAVE_MSG=TIOAMSG;
15     DFHSC TYPE=GETMAIN,                                   *
16           NUMBYTE=40,                                     *
17           INITIMG=40,                                     *
18           CLASS=TERMINAL
19     TICABAR=TCASCSA;
20     TCTTEDA=TIOABAR;
21     TIOAMSG=SAVE_MSG;
22     TIOATDL=SAVE_LENGTH;
23     DFHTC TYPE=WRITE
24     END;
```

Figure 11. Example of CICS PL/I application program

A discussion of the significance of each of the lines of Figure 11 follows.

STATEMENT NUMBER	DESCRIPTION
01	Required for PL/I. REENTRANT option specified to meet requirement of CICS that code be quasi-reentrant.
02	Retrieves symbolic storage definition for CSA and establishes addressability.
03	Retrieves 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	Retrieves symbolic storage definition for TCTTE and establishes addressability.
07	Retrieves 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	Writes the message to the terminal, waits for the operator's reply, and reads the operator's reply.
12	Reestablishes addressability for the TIOA using address in the TCTTE.
13-14	Saves the operator's message and message length in the TCA.
15-18	CICS macro instruction that requests 40 bytes of terminal storage initialized to blanks (terminal storage is chained to Terminal Control Table).
19	Establishes addressability for the new TIOA (the address of the newly acquired storage is in the TCASCSA field of the TCA).
20	Places address of new TIOA in Terminal Control Table.
21-22	Moves message and length of message to output area (TIOA).
23	CICS macro instruction that sends operator's message back to the terminal.
24	Return control to CICS.

SERVICE INVOCATION

CICS provides supervisory and data management services through CICS management programs. These services and related management programs are as follows:

- Task services - Task Management
- Storage services - Storage Management
- Program services - Program Management
- Dump services - Dump Management
- Terminal services - Terminal Management
- File services - File Management
- Transient data services - Transient Data Management
- Temporary storage services - Temporary Storage Management
- Time services - Time Management

Each of the CICS management programs performs the following basic functions:

1. Analyzes the specific service request of application programs or other CICS programs.
2. Performs the requested service by communicating with the operating system, as necessary, through macro instructions.
3. Retains the status of each service request until the service is provided.
4. Maintains statistical information that can be used to evalulate system performance.

TASK SERVICES

Task Management provides the capability to process transactions (tasks) concurrently. Transactions are scheduled, through Task Control, and are processed according to priorities assigned by the user. Control of the central processing unit (CPU) is given to the highest priority task that is ready to be processed. Control of the CPU is returned to the operating system when no further work can be done by CICS or by the user-written application programs.

When a transaction is initiated in CICS, Task Control dynamically allocates storage for the Task Control Area (TCA), attaches the TCA to the TCA chain according to priority, obtains the initial program identification from the Program Control Table (PCT), and transfers control to Program Control.

The Task Management macro instruction (DFHKC) is used to request any of the following services:

1. Initiate a task.
2. Change the priority of a task.
3. Synchronize a task.
4. Synchronize the use of a resource by a task.
5. Purge a task on system overload (if the optional stall protection feature has been installed).

The following operands can be included in the DFHKC macro instruction:

```
DFHKC TYPE=ATTACH,                                          *
      FCADDR=symbolic address,                              *
      TRANSID=name

DFHKC TYPE=CHAP,                                            *
      PRTY=priority value

DFHKC TYPE=WAIT,                                            *
      DCI=SINGLE,LIST,DISP,                                 *
      ECADDR=symbolic address

DFHKC TYPE=ENQ,DEQ,                                         *
      QARGADR=symbolic address,                             *
      QARGLNG=number

DFHKC TYPE=PURGE,NOPURGE
```

The number of tasks that can be active within the sytem at a given time is limited by the availability of main storage and/or by the "maximum number of tasks" control. A new task is not initiated by CICS unless sufficient main storage is available to process it. Instead, the request to initiate a task is queued (stored) until sufficient main storage becomes available.

INITIATE A TASK (ATTACH)

Task initiation within CICS is invoked by issuing the

```
DFHKC TYPE=ATTACH,                                          *
      FCADDR=symbolic address,                              *
      TRANSID=name
```

macro instruction. This macro instruction causes Task Control to obtain the controlling area for a task and insert that task within priority sequence. This macro instruction is intended to be used by other CICS control modules, but is also available for use by the application programmer to initiate additional tasks. Any additional tasks initiated by the application programmer must terminate themselves through use of the Program Control (DFHPC) RETURN macro instruction.

FCADDR= specifies the symbolic address of a facility control area. This is typically the address of the attaching task's TCA or a reserved field in the CSA. The purpose is to establish communication between the attaching task and the attached task.

TRANSID= specifies the transaction identification of the attached task.

If the DFHKC TYPE=ATTACH macro instruction is used by the application programmer, he has the responsibility to provide the facility control area address and transaction identification required by CICS to initiate a new task. He can accomplish this in either of two ways: (1) by including the FCADDR=symbolic address operand and TRANSID=symbolic name operand in the DFHKC TYPE=ATTACH macro instruction to statically assign to fields in the TCA a facility control area address and a transaction identification for the duration of the task, or (2) by coding two instructions, prior to issuing the DFHKC TYPE=ATTACH macro instruction, that provide the capability to dynamically assign to fields in the TCA a facility control area address and a transaction identification. (See the discussion of the TCA in the section "Storage Definition".)

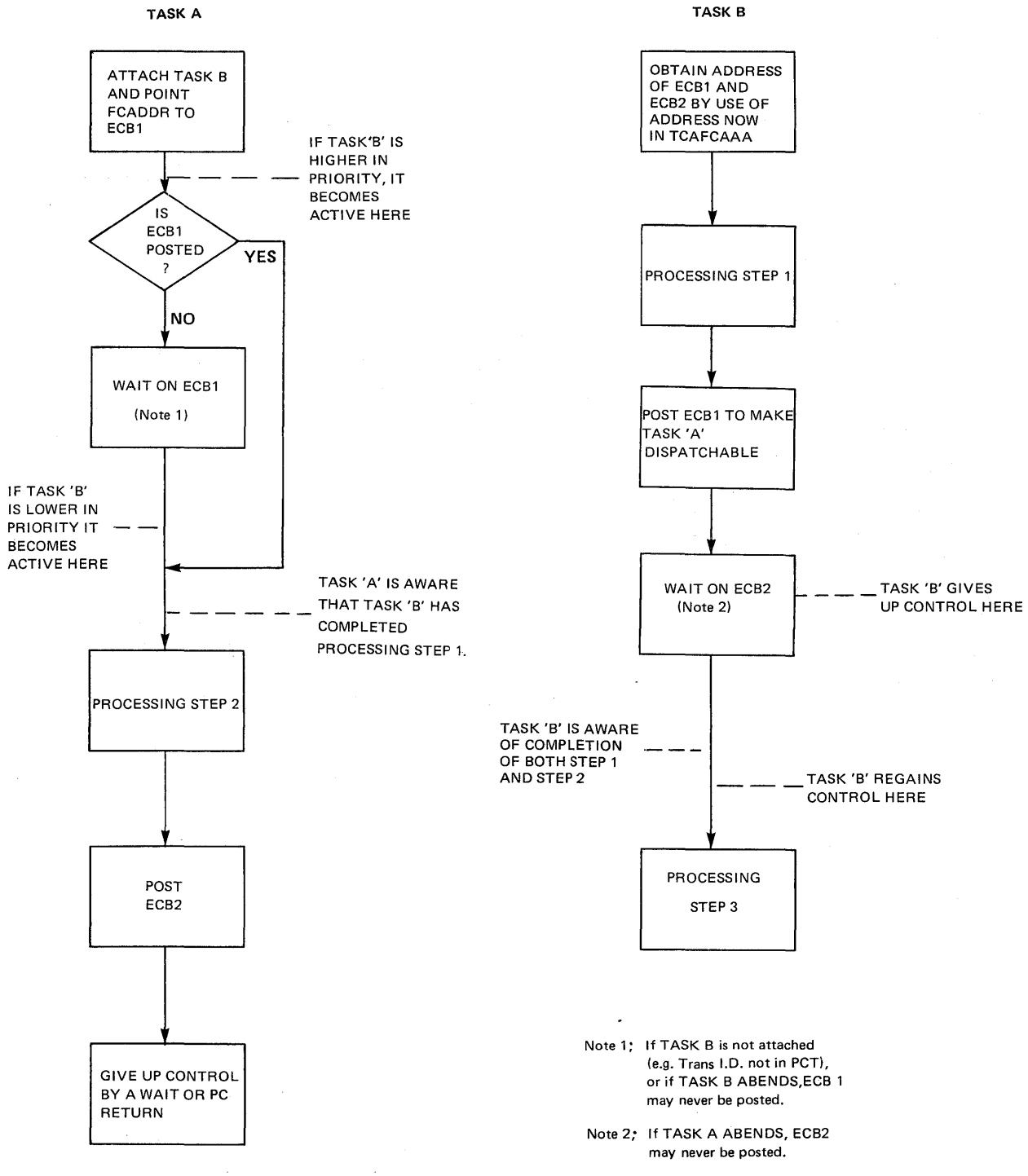
The specified task will not be attached if the transaction identification is not in the PCT or the program name is not in the PPT. If this situation exists or the attached task ABEND, a message is sent to the operator, but the attaching task will not be notified of the condition. Therefore the TYPE=ATTACH macro instruction must be used with extreme caution by the application programmer.

For all transactions associated with a terminal, the Facility Control Area address is the address of the TCTTE for the terminal. This address provides access to control information contained in the Terminal Control Table necessary for communication between the program logic and the terminal.

Although it is possible to attach a task directly to a terminal by using the ATTACH macro instruction, the application programmer or user should consider utilizing one of the following methods:

- Automatic task initiation through Transient Data Management
- Automatic task initiation through Time Management (Interval Control program)
- Identification of the transaction ID to be used with the next input message from the terminal by means of DFHPC TYPE=RETURN macro instruction.

The following flowchart 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. Note that Task B is a non-terminal oriented task, therefore 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.



Note 1; If TASK B is not attached (e.g. Trans I.D. not in PCT), or if TASK B ABENDS,ECB 1 may never be posted.

Note 2; If TASK A ABENDS, ECB2 may never be posted.

Most tasks running under CICS are initiated (attached) at a terminal and are thus associated with a terminal. Tasks initiated by CICS management programs (for example, automatic task initiation of Transient Data Control) may or may not be associated with a terminal. If not associated with a terminal, the Facility Control Area address can serve as a pointer to additional facility control information required for the execution of the task. For example, it can be the address of an entry in the Destination Control Table that is associated with a hardware resource (terminal, data set, etc.).

The transaction identification is used only for the current ATTACH; it is not carried in the TCA for the duration of the task.

The following example illustrates the coding required to statically provide a facility control area address and transaction identification:

```
DFHKC TYPE=ATTACH,          INITIATE NEW TASK                    *
      FCADDR=FACCTL,        USER'S FCA ADDRESS                   *
      TRANSID=TRN1          TRANSACTION IDENTIFICATION
```

The following examples illustrate the coding required to dynamically provide a facility control area address and transaction identification.

For Assembler language:

```
MVC TCAKCTI,=CL4'TRN1'      TRANSACTION IDENTIFICATION
MVC TCAKCFA,=A(FACCTL)      USER'S FCA ADDRESS
     .
     .
     .
DFHKC TYPE=ATTACH           INITIATE NEW TASK
```

For ANS COBOL:

```
MOVE 'TRN1' TO TCAKCTI.     NOTE TRANSACTION IDENTIFICATION.
MOVE FACADR TO TCAKCFA.     NOTE USER'S FCA ADDRESS.
     .
     .
     .
DFHKC TYPE=ATTACH           INITIATE NEW TASK
```

For PL/I:

```
TCAKCTI='TRN1';             /*TRANSACTION IDENTIFICATION*/
TCAKCFA=FACADR;             /*USER'S FCA ADDRESS*/
     .
     .
     .
DFHKC TYPE=ATTACH           INITIATE NEW TASK
```

CHANGE PRIORITY OF A TASK (CHAP)

The dispatching priority of an existing task can be changed by issuing the

```
DFHKC TYPE=CHAP,                                               *
      PRTY=priority value
```

macro instruction. This instruction is used to replace the priority value contained in the TCATCDP field of the TCA with a value specified

by the application programmer. This value must be specified in the range 0-255, where 255 represents the highest priority.

The application programmer can change the priority of a task in either of two ways: (1) by including the PRTY=priority value operand in the DFHKC TYPE=CHAP macro instruction to statically assign to the TCATCDP field a new dispatching priority for the duration of the task, or (2) by coding a single instruction, prior to issuing the DFHKC TYPE=CHAP macro instruction, that provides the capability to dynamically assign to the TCATCDP field a new priority value as often as desired within a given task.

A compute bound task can voluntarily relinquish control to all tasks of equal or higher priority by issuing a

```
DFHKC TYPE=CHAP
```

macro instruction. No priority value is specified.

The following example illustrates the coding required to statically assign a new task dispatching priority value:

```
DFHKC TYPE=CHAP,        CHANGE PRIORITY OF THIS TASK          *
      PRTY=255          NEW PRIORITY VALUE
```

The following examples illustrate the coding required to dynamically assign a new task-dispatching priority value. Note that this value can be specified as a binary, decimal, or hexadecimal number, depending on the programming language used.

For Assembler language:

```
MVI TCATCDP,X'FF'       ASSIGN NEW PRIORITY VALUE
     .
     .
     .
DFHKC TYPE=CHAP         CHANGE PRIORITY OF THIS TASK
```

For ANS COBOL:

```
MOVE 255 TO TCATCDP.    NOTE ASSIGN NEW PRIORITY VALUE.
     .
     .
     .
DFHKC TYPE=CHAP         CHANGE PRIORITY OF THIS TASK
```

For PL/I:

```
TCATCDP=255;            /*ASSIGN NEW PRIORITY VALUE*/
     .
     .
     .
DFHKC TYPE=CHAP         CHANGE PRIORITY OF THIS TASK
```

SYNCHRONIZE A TASK (WAIT)

The application programmer can synchronize a task with the completion of one or more events related to the same task or to another task by issuing the

```
DFHKC TYPE=WAIT,                                              *
      DCI=SINGLE,LIST,DISP,                                   *
      ECADDR=symbolic address
```

macro instruction. This macro instruction provides a method of directly relinquishing control to some other task until such time as the event(s) being waited on are completed. It also allows a task to be designated as "dispatchable" to voluntarily relinquish control to tasks of a higher dispatching priority.

The application programmer must specify the circumstances under which synchronization of a task is to occur by including the DCI=keyword operand (dispatch control indicator) in the DFHKC TYPE=WAIT macro instruction.

If the task is to be synchronized with the completion of a single event or an event of a list of events, the application programmer must specify the symbolic address of either the single event control area or the list of event control areas. He can accomplish this in either of two ways: (1) by including the ECADDR=symbolic address operand in the DFHKC TYPE=WAIT macro instruction, or (2) by coding a single instruction, prior to issuing the DFHKC TYPE=WAIT macro instruction, that places the event control address in the TCATCEA field of the TCA. In either case, the control area(s) referenced must conform to the format and standard posting conventions associated with the operating system (for example, ECB's in OS/360, CCB's in DOS/360).

Relinquish Control to a Task of Higher Priority

The DFHKC TYPE=WAIT,DCI=DISP macro instruction is used by the application programmer to voluntarily relinquish control to a task of higher dispatching priority. Control is returned to the waiting task if no other task of a higher priority is ready to be processed.

The following is an example of the coding required to voluntarily relinquish control to a task of higher dispatching priority:

```
DFHKC TYPE=WAIT,          RELINQUISH CONTROL OF CICS          *
      DCI=DISP            AND REMAIN DISPATCHABLE
```

Note: When binary synchronous communication lines are part of the user's configuration, it is possible for these communication lines to time out if "excessive" CPU time is required by the application program. One way to alleviate this condition is to have the application program issue a DFHKC TYPE=WAIT,DCI=DISP macro instruction to voluntarily relinquish control before the line time out can occur.

Synchronize a Task with a Single Event

The DFHKC TYPE=WAIT,DCI=SINGLE macro instruction is 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 symbolic address of the appropriate event control area must be provided in either of two ways: (1) by including the ECADDR=symbolic address operand in the DFHKC TYPE=WAIT,DCI=SINGLE macro instruction, or (2) by coding a single instruction, prior to issuing the DFHKC TYPE=WAIT,DCI=SINGLE macro instruction, that places the address of the event control area in the TCATCEA field of the TCA. The control area referenced must conform to the format and standard posting conventions associated with the operating system.

The following is an example of the coding required to synchronize a task with a single event, statically providing the symbolic address of the appropriate event control area:

```
DFHKC TYPE=WAIT,          RELINQUISH CONTROL OF CICS          *
      DCI=SINGLE,         WAIT ON SINGLE EVENT                *
      ECADDR=EVENTCTL     ADDRESS OF EVENT CONTROL AREA
```

The following are examples of the coding required to synchronize a task with a single event, dynamically providing the symbolic address of the appropriate event control area.

For Assembler language:

```
ST    SINGADDR,TCATCEA    PLACE SYMBOLIC ADDRESS IN TCA
 .
 .
 .
DFHKC TYPE=WAIT,          RELINQUISH CONTROL OF CICS          *
      DCI=SINGLE          WAIT ON SINGLE EVENT
```

For ANS COBOL:

```
MOVE SINGADDR TO TCATCEA.  NOTE PLACE SYMBOLIC ADDR IN TCA.
 .
 .
 .
DFHKC TYPE=WAIT,          RELINQUISH CONTROL OF CICS          *
      DCI=SINGLE          WAIT ON SINGLE EVENT
```

For PL/I:

```
TCATCEA=SINGADDR;         /*PLACE SYMBOLIC ADDRESS IN TCA*/
 .
 .
 .
DFHKC TYPE=WAIT,          RELINQUISH CONTROL OF CICS          *
      DCI=SINGLE          WAIT ON SINGLE EVENT
```

Synchronize a Task with a List of Events

The DFHKC TYPE=WAIT,DCI=LIST macro instruction is used by the application programmer to synchronize a task with the completion of an element of a list of events. This list consists of a series of contiguous four-byte fields, each field containing the symbolic address of a single event control area. The last four-byte field of the list contains hexadecimal F's.

The symbolic address of the appropriate list of event control areas must be provided in either of two ways: (1) by including the ECADDR=symbolic address operand in the DFHKC TYPE=WAIT,DCI=LIST macro instruction, or (2) by coding a single instruction, prior to issuing the DFHKC TYPE=WAIT,DCI=LIST macro instruction, that places the address of the list of event control areas in the TCATCEA field of the TCA. The control area referenced by each entry in the list must conform to the format and standard posting conventions associated with the operating system.

The following is an example of the coding required to synchronize a task with a list of events, statically providing the symbolic address of the appropriate list of events:

```
DFHKC TYPE=WAIT,            RELINQUISH CONTROL OF CICS          *
      DCI=LIST,             WAIT ON A LIST OF EVENTS            *
      ECADDR=TCPCLIST       ADDRESS OF LIST OF EVENTS
```

The following are examples of the coding required to synchronize a task with a list of events, dynamically providing the symbolic address of the appropriate list of events.

For Assembler language:

```
ST   LISTADDR,TCATCEA       PLACE SYMBOLIC ADDRESS IN TCA
 .
 .
 .
DFHKC TYPE=WAIT,            RELINQUISH CONTROL OF CICS          *
      DCI=LIST              WAIT ON A LIST OF EVENTS
```

For ANS COBOL:

```
MOVE LISTADDR TO TCATCEA.   NOTE PLACE SYMBOLIC ADDR IN TCA.
 .
 .
 .
DFHKC TYPE=WAIT,            RELINQUISH CONTROL OF CICS          *
      DCI=LIST              WAIT ON A LIST OF EVENTS
```

For PL/I:

```
TCATCEA=LISTADDR;           /*PLACE SYMBOLIC ADDRESS IN TCA*/
 .
 .
 .
DFHKC TYPE=WAIT,            RELINQUISH CONTROL OF CICS          *
      DCI=LIST              WAIT ON A LIST OF EVENTS
```

SINGLE-SERVER RESOURCE SYNCHRONIZATION (ENQ/DEQ)

In the CICS environment where tasks (transactions) are processed concurrently, it is sometimes desirable to protect a given resource from concurrent use by another task. The application programmer can, by adhering to an installation convention, give sole control of a serially reusable resource to a single task until that task is completely finished with that resource. He can accomplish this by issuing the

```
DFHKC TYPE=ENQ,                                             *
      QARGADR=symbolic address,                             *
      QARGLNG=number
```

macro instruction, identifying the resource. This macro instruction, when executed, causes the task to be synchronized with the availability of the specified resource; control is returned to the task when the resource is available. When all programs accessing a resource adhere to the convention of "engueing upon" the resource, that resource is afforded this "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 instruction receives the resource when it becomes available. The other tasks obtain the resource, in turn, in the order in which they enqueue upon it.

The application programmer can release single-server protection from a resource by issuing the

```
DFHKC TYPE=DEQ,                                              *
      QARGADR=symbolic address,                              *
      QARGLNG=number
```

macro instruction. Task Control automatically "dequeues" all active single-server resource protection requests associated with that task upon termination of the task.

When issuing the DFHKC TYPE=ENQ macro instruction, the application programmer must identify the single-server resource he is enqueuing upon by using either of the following methods:

1. Specify a symbolic main storage address that represents the single-server resource. If this method is used, the application programmer must provide the symbolic main storage address by including the QARGADR=symbolic address operand in the DFHKC TYPE=ENQ macro instruction or by coding instructions, prior to issuing the DFHKC TYPE=ENQ macro instruction, that place the address in the low-order three bytes of the four-byte TCATCQA field of the TCA. He must place binary zeros in the high-order byte.
2. Provide a unique argument, limited to 255 bytes and contained at a specified symbolic main storage address, that identifies the resource. If this method is used, the application programmer must provide the symbolic main storage address of the argument along with the length of the argument, by including the QARGADR=symbolic address and QARGLNG=number operands in the DFHKC TYPE=ENQ macro instruction or by coding instructions, prior to issuing the DFHKC TYPE=ENQ macro instruction, that place the symbolic address in the low-order three bytes of the four-byte TCATCQA field of the TCA and the length of the argument (in bytes) in the high-order byte.

The following are examples of the coding required to enqueue upon a single-server resource using method 1.

For Assembler language:

```
DFHKC TYPE=ENQ,                    ENQ ON SINGLE-SERVER RESOURCE   *
      QARGADR=CSAWABA              SPECIFY SYMBOLIC ADDRESS

            OR

LA   WORKREG,CSAWABA
ST   WORKREG,TCATCQA
.
.
.
DFHKC TYPE=ENQ
```

For ANS COBOL:

```
01  DFHCSADS COPY DFHCSADS.
    02  CSAWABA PICTURE X(50).
.
.
.
DFHKC TYPE=ENQ,                    ENQ ON SINGLE-SERVER RESOURCE   *
      QARGADR=CSAWABA              SPECIFY SYMBOLIC ADDRESS
```

For PL/I:

```
%INCLUDE DFHCSADS;
DECLARE 1 DFHEXCSA BASED (CSACBAR),
    2 FILLER CHAR (512),
    2 CSAWABA CHAR (50);
.
.
.
DFHKC TYPE=ENQ,                 ENQ ON SINGLE-SERVER RESOURCE   *
      QARGADR=CSAWABA           SPECIFY SYMBOLIC ADDRESS

            OR

TCATCQA=ADDR(CSAWABA);
.
.
.
DFHKC TYPE=ENQ
```

The following are examples of the coding required to enqueue upon a single-server resource using method 2.

For Assembler language:

```
DFHKC TYPE=ENQ,                                                 *
      QARGADR=SOCSECNO,                                         *
      QARGLNG=9

            OR

LA   WORKREG,SOCSECNO
ST   WORKREG,TCATCQA
MVI TCATCQA,9
.
.
.
DFHKC TYPE=ENQ
```

For ANS COBOL:

```
DFHKC TYPE=ENQ,                                                 *
      QARGADR=SOCSECNO,                                         *
      QARGLNG=9
```

For PL/I:

```
DFHKC TYPE=ENQ,                                                 *
      QARGADR=SOCSECNO,                                         *
      QARGLNG=9

            OR

%INCLUDE DFHTCADS;
DECLARE 1 DFHEXTCA BASED (TCACBAR),
    2 FILLER CHAR (20),
    2 TCATCQAL BIT(8);
.
.
.
TCATCQA=ADDR(SOCSECNO);
TCATCQAL='00001001'B;
```

```
.
.
.
DFHKC TYPE=ENQ
```

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.

PURGE A TASK ON SYSTEM OVERLOAD (PURGE/NOPURGE)

Certain overload conditions can occur in CICS where all of a given system resource (for example, main storage) has been allocated and where each task requires still more of that resource. The result is a situation where no task is able to continue processing and no new task can be initiated; the system stalls.

If the optional stall protection feature was provided during system generation, CICS has the capability to detect certain system overload conditions and take corrective action. Corrective action consists, in part, of purging (deleting) the lowest priority task in the system that is designated as purgeable.

A task is initially defined as purgeable or not purgeable by the user in the Program Control Table (PCT) entry associated with the transaction identification for that task. The application programmer can dynamically change the purgeability status of a task by issuing the

```
DFHKC TYPE=PURGE
```

macro instruction to indicate the task is purgeable, or the

```
DFHKC TYPE=NOPURGE
```

macro instruction to indicate the task is not purgeable. The designated status remains in effect until another change is initiated or until the task is terminated.

The DFHKC TYPE=PURGE and DFHKC TYPE=NOPURGE macro instructions have no effect on the execution of a task if the stall protection feature is not provided by the user during system generation.

STORAGE SERVICES

Storage Management controls all dynamic main storage for CICS and for the user-written application programs. Requests to acquire or release main storage are communicated to Storage Control via a CICS macro instruction.

CICS management programs issue requests for main storage to provide input/output areas, program load areas, and user-defined work areas needed to process a transaction. The user's application program can issue requests for main storage to provide intermediate work areas and any other main storage not automatically provided by CICS but needed to process a transaction. Any main storage acquired by the user's application program can be initialized to whatever bit configuration the user desires; for example, to binary zeros or EBCDIC blanks.

All main storage associated with a transaction is chained. This allows CICS to release all main storage associated with a transaction

upon request by the user or when the transaction is either normally or abnormally terminated. Main storage is accounted for as follows:

1. Task Control Areas (TCA's) are chained off the Common System Area (CSA).
2. Transaction storage is chained off the Task Control Area (TCA).
3. 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).
4. Program storage is accounted for in the Program Processing Table (PPT).
5. Suspended tasks are accounted for by the suspending program (Task Control, Storage Control, Temporary Storage Control).

If there is insufficient main storage to satisfy a storage acquisition request, Storage Control causes the processing of that task to be delayed by placing it in a "wait" queue until sufficient main storage becomes available. In the meantime, no new tasks are initiated by CICS until the "short on storage" condition is alleviated. The only exception to this method of allocating main storage occurs in the CICS/DOS-ENTRY system where, under certain circumstances, a "short on storage" condition causes the transaction to be abnormally terminated unless the COND=YES operand has been included in the DFHSC TYPE=GETMAIN macro instruction. (See the section "Purge a Task on System Overload" for corrective action that can be taken if a "system stall" condition occurs.)

The Storage Management macro instruction (DFHSC) is used to request any of the following services:

1. Acquire and initialize main storage.
2. Release main storage.

The following operands can be included in the DFHSC macro instruction:

```
DFHSC TYPE=GETMAIN,                                          *
      INITIMG=number,YES,                                    *
      NUMBYTE=number,                                        *
      COND=YES or (YES,symbolic address) or                  *
           (NO,symbolic address),                            *
      CLASS=TERMINAL,USER,TRANSDATA,TEMPSTRG

DFHSC TYPE=FREEMAIN,                                         *
      RELEASE=ALL
```

OBTAIN AND INITIALIZE MAIN STORAGE (GETMAIN)

Requests for main storage are made by issuing the

```
DFHSC TYPE=GETMAIN,                                          *
      INITIMG=number,YES,                                    *
      NUMBYTE=number,                                        *
      COND=YES or (YES,symbolic address) or                  *
           (NO,symbolic address),                            *
      CLASS=TERMINAL,USER,TRANSDATA,TEMPSTRG
```

macro instruction. This instruction is used by the application programmer to obtain main storage of a specified size and class and is used, optionally, to initialize that storage to whatever bit configuration the application programmer desires. The address of the storage area obtained upon execution of this instruction is

automatically placed in the TCASCSA field of the TCA by CICS; the storage itself is doublewcrd aligned.

Whenever the applicatien programmer uses the DFHSC TYPE=GETMAIN macro instruction, he must do the fcllowing:

1. Specify the class of storage desired using the CLASS=class operand in conjunction with the DFHSC TYPE=GETMAIN macro instruction.
2. Calculate the number of bytes required and either specify that amcunt in the NUMBYTE=number operand, or dynamically place it in the TCASCNB field before issuing the DFHSC macro instruction.
3. Specify a symbolic base address for the storage area.
4. Move the storage address located at TCASCSA to the symbolic base address. (This address always points to the Storage Accounting Area.)
5. Copy the symbolic storage definition for the appropriate input/cutput area or Storage Accounting Area prior to the symbolic definition of the user's program storage area.

The fcllowing is an example of the coding required to request a new area of main storage:

```
DFHSC TYPE=GETMAIN,       CBTAIN NEW STORAGE AREA              *
      INITIMG=00,         INITIALIZE WITH BINARY ZEROS         *
      NUMBYTE=1024,       SIZE OF STORAGE REQUESTED            *
      CLASS=TERMINAL      CLASS OF STORAGE REQUESTED
```

The fcllowing are examples of the coding required to dynamically request a new area of main storage.

For Assembler language:

```
MVI   TCASCIB,B'0'        INITIALIZE WITH BINARY ZEROS
MVC   TCASCNB,=H'1024'    SIZE OF STORAGE REQUESTED
      .
      .
      .
DFHSC TYPE=GETMAIN,       CBTAIN NEW STORAGE AREA              *
      INITIMG=YES,        INITIALIZE WITH BINARY ZEROS         *
      COND=YES,           RETURN CONTROL IF NO STORAGE         *
      CLASS=TERMINAL      CLASS OF STORAGE REQUESTED
CLC   TCASCSA,=F'0'       CHECK TCASCSA FIELD FOR ZEROS
BE    NOSTRG              BRANCH TO NOSTRG IF NO STORAGE
L     TICABAR,TCASCSA     LOAD REGISTER IF STORAGE FOUND
```

For ANS COBOL:

```
MOVE 0 TO TCACSIB.        NOTE INITIALIZE WITH BINARY ZEROS.
MOVE 1024 TO TCASCNB.     NOTE SIZE OF STORAGE REQUESTED.
      .
      .
      .
DFHSC TYPE=GETMAIN,       CBTAIN NEW STORAGE AREA              *
      INITIMG=YES,        INITIALIZE WITH BINARY ZEROS         *
      CCND=YES,           RETURN CONTROL IF NO STORAGE         *
      CLASS=TERMINAL      CLASS OF STORAGE REQUESTED
IF TCASCSA EQUAL 0 GO TC NOSTRG.
MOVE TCASCSA TO TICABAR.
```

For 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=(NO,NOSTRG),     RETURN CONTROL IF NO STORAGE         *
      CLASS=TERMINAL        CLASS OF STORAGE REQUESTED
TIOABAR=TCASCSA;            /*LOAD REGISTER IF STORAGE FOUND*/
```

The DFHSC TYPE=GETMAIN macro instruction can include the following operands.

INITIMG: This operand is used to initialize an acquired storage area to the bit configuration specified in hexadecimal, for example, to binary zeros (00) or EBCDIC blanks (40). The application programmer can, at his option, place the initialization value in the TCASCIB field of the TCA prior to the execution of the DFHSC TYPE=GETMAIN macro instruction; in this case the INITIMG=YES operand must be included in the macro instruction.

NUMBYTE: This operand is used to specify the size of the storage area being requested. A value up to 65535 can be specified. The application programmer can, at his option, indicate the number of storage bytes requested prior to execution of the DFHSC TYPE=GETMAIN macro instruction by placing this value in the TCASCNB field of the TCA; in this case the NUMBYTE=number operand is omitted.

Note: Depending upon the class of storage specified (see the CLASS operand below), CICS Storage Management automatically increments the amount of storage requested to allow for the Storage Accounting Area (SAA) 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) class of storage, the amount requested must include four additional bytes to allow for a portion of the CICS control information.

COND: This operand is used by the application programmer to conditionally acquire main storage. Control is always returned to the user, even if the storage requested is not available. If storage is not available, the TCASCSA field of the TCA is filled with binary zeros.

The COND=YES operand causes control to be given to the instruction immediately following the DFHSC TYPE=GETMAIN macro instruction. If the application programmer uses this operand, he must check the TCASCSA field for zeros to determine whether the requested storage area was acquired.

The COND=(YES,symbolic address) operand causes CICS to test whether or not the requested storage was acquired. If storage was acquired, CICS causes a branch to the location specified in the symbolic address parameter; if storage was not acquired, control is returned to the application program at the instruction immediately following the DFHSC TYPE=GETMAIN,COND=(YES,symbolic address) macro instruction.

The COND=(NO,symbolic address) operand causes CICS to test whether or not the requested storage was acquired. If storage was not acquired,

CICS causes a branch to the location specified in the symbolic address parameter; if storage was acquired, control is returned to the application program at the instruction immediately following the DFHSC TYPE=GETMAIN,COND=(NO,symbolic address) macro instruction.

CLASS: This operand is used to specify the class of storage being requested. If the task itself does not release acquired storage when it is no longer needed, the storage is released by CICS upon termination of the task. CLASS must be coded with one of the following parameters: TERMINAL, USER, TRANSDATA, or TEMPSTRG.

CLASS=TERMINAL specifies a storage area to be used for terminal input/output (TIOA). This area is chained to the Terminal Control Table terminal entry. All requests for storage related to terminal input/output must specify this class.

CLASS=USER specifies a storage area to be associated with the user's application program and used by that program. This area is chained to the TCA associated with the task in which the storage is requested.

CLASS=TRANSDATA specifies a transient data record storage area (TDIA, TDOA). This area is chained to the TCA associated with the task in which the storage is requested and is used by Transient Data Control.

CLASS=TEMPSTRG specifies a temporary storage input/output area (TSIOA). This area is chained to the TCA associated with the task in which storage is requested and is used by Temporary Storage Control.

The CLASS=USER, CLASS=TRANSDATA, and CLASS=TEMPSTRG specifications have essentially the same effect; that is, the number of bytes acquired is always eight more than the number specified in the NUMBYTE operand (to allow for the storage accounting field), and the storage is always chained off the TCA. The only advantage of using the CLASS=TRANSDATA or CLASS=TEMPSTRG specification instead of the CLASS=USER specification is for the purpose of code documentation.

RELEASE MAIN STORAGE (FREEMAIN)

Previously acquired main storage is released by issuing the

```
DFHSC TYPE=FREEMAIN,                                          *
      RELEASE=ALL
```

macro instruction. If the task itself does not release acquired storage, the storage is released by CICS upon termination of the task.

If the application programmer uses the DFHSC TYPE=FREEMAIN macro instruction to release a single storage area, he must place the address of that area in the TCASCSA field of the TCA <u>prior</u> to the execution of the DFHSC TYPE=FREEMAIN macro instruction. If all terminal storage acquired by means of the DFHSC TYPE=GETMAIN,CLASS=TERMINAL macro instruction is to be released, the RELEASE=ALL operand can be coded in conjunction with the DFHSC TYPE=FREEMAIN macro instruction to achieve that result; in this case it is not necessary to place any address in the TCA.

The following is an example of the coding required to release all main storage currently allocated to a specific terminal:

```
DFHSC TYPE=FREEMAIN,                                          *
      RELEASE=ALL           RELEASE ALL TERMINAL STORAGE
```

The following are examples of the coding required to release a single main storage area:

For Assembler language:

```
ST     TIOABAR,TCASCSA        PLACE STORAGE AREA ADDRESS IN TCA
       .
       .
       .
DFHSC TYPE=FREEMAIN           RELEASE STORAGE AREA
```

For ANS COBOL:

```
MOVE TIOABAR TO TCASCSA.      NOTE PLACE STRG AREA ADDR IN TCA.
       .
       .
       .
DFHSC TYPE=FREEMAIN           RELEASE STORAGE AREA
```

For PL/I:

```
TCASCSA=TIOABAR;              /*PLACE STORAGE AREA ADDRESS IN TCA*/
       .
       .
       .
DFHSC TYPE=FREEMAIN           RELEASE STORAGE AREA
```

PROGRAM SERVICES

All program communication within CICS is accomplished by Program Management through Program Control. Requests for program services are communicated to Program Control via CICS macro instructions.

User-written application programs must be coded so that they are "serially reusable" between entry and exit points of the program. Entry and exit points of a program coincide with the use of CICS macro instructions, since an application program temporarily loses control after it begins executing only upon execution of a CICS macro instruction. 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.

This required quality of application programs written to run under CICS is called "quasi-reentrance", since the programs need not meet System/360 or System/370 specifications for true reentrance. Quasi-reentrance allows a single copy of a user-written application program to be used to process several transactions concurrently, thereby reducing the requirement for multiple copies of the same program in main storage.

Transactions can share the use of common work areas. However, each transaction 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 transaction. The application programmer must provide that intermediate storage area by symbolically defining it in his program.

CICS automatically saves program control information and general purpose registers, when applicable, in the Task Control Area (TCA).

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 Program Management macro instruction (DFHPC) is used to request any of the following services:

1. Link one user-written application program to another, anticipating subsequent return to the requesting program.
2. Transfer control from one user-written application program to another anticipating no return to the requesting program.
3. Load a designated application program or table into main storage and return control to the requesting program.
4. Return control from one user-written application program to another or to CICS.
5. Release a previously loaded application program from main storage.
6. Abnormally terminate a transaction and its related task.

The following operands can be included in the DFHPC macro instruction:

```
DFHPC TYPE=LINK,                                          *
      PROGRAM=name

DFHPC TYPE=XCTL,                                          *
      PROGRAM=name

DFHPC TYPE=LOAD,                                          *
      PROGRAM=name,                                       *
      LOADLST=NO

DFHPC TYPE=RETURN,                                        *
      TRANSID=transaction code

DFHPC TYPE=DELETE,                                        *
      PROGRAM=name

DFHPC TYPE=ABEND,                                         *
      ABCODE=value,YES
```

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. Figure 12 illustrates this difference between program linkage and transfer of program control.

Parameters can be passed from one program to another 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).

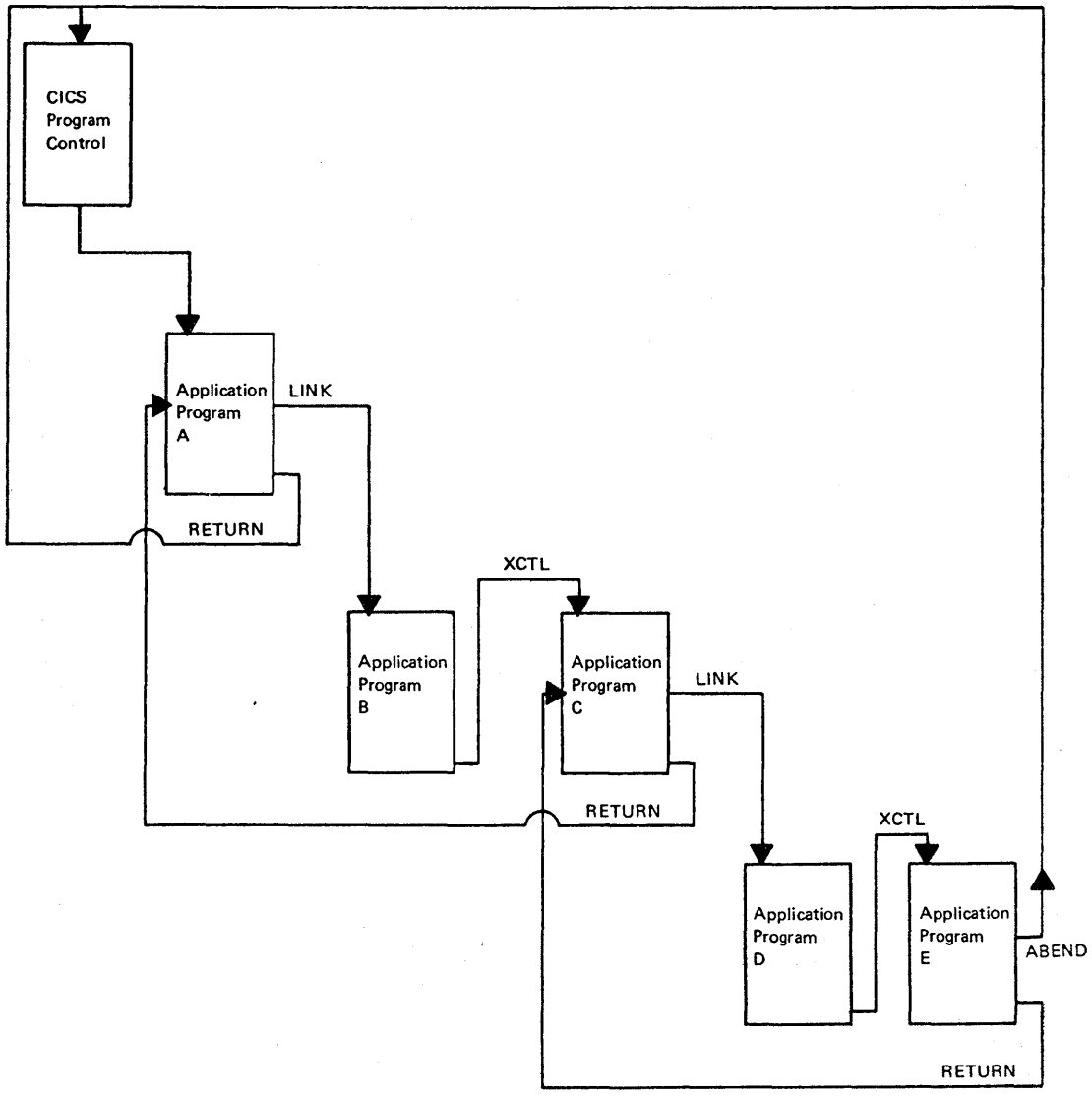


Figure 12. Application programs are executed at various logical levels

PASS PROGRAM CONTROL ANTICIPATING SUBSEQUENT RETURN (LINK)

Program control is passed from a user-written application program at one logical level to a user-written application program at the next lower logical level in response to the

```
DFHPC TYPE=LINK,                                                    *
      PROGRAM=name
```

macro instruction. When the DFHPC TYPE=RETURN macro instruction is executed in the linked-to program, control is returned to the program initiating the linkage at the next sequential (executable) instruction.

The application programmer must provide the name (identification) of the program to which control is passed. He can do this in either of two ways: (1) by including the PROGRAM=name operand in the DFHPC TYPE=LINK macro instruction, or (2) by coding a single instruction, prior to issuing the DFHPC TYPE=LINK macro instruction, that places the program name in the TCAPCPI field of the TCA. This same program name must also have been placed in the Processing Program Table (PPT) prior to execution of CICS.

The following is an example of the coding required to request a link to another application program:

```
DFHPC TYPE=LINK,                                                    *
      PROGRAM=PROG1
```

The following are examples of the coding required to dynamically link to another application program.

For Assembler language:

```
MVC   TCAPCPI,=CL8'PROG1'  PLACE LINKED-TO PROGRAM NAME IN TCA
      .
      .
      .
DFHPC TYPE=LINK            LINK TO PROGRAM AT NEXT LOWER LEVEL
```

For ANS COBOL:

```
MOVE 'PROG1' TO TCAPCPI.   NOTE LINKED-TO PROGRAM NAME TO TCA.
      .
      .
      .
DFHPC TYPE=LINK            LINK TO PROGRAM AT NEXT LOWER LEVEL
```

For PL/I:

```
TCAPCPI='PROG1';           /*PLACE LINKED-TO PRGM NAME IN TCA*/
      .
      .
      .
DFHPC TYPE=LINK            LINK TO PROGRAM AT NEXT LOWER LEVEL
```

TRANSFER PROGRAM CONTROL (XCTL)

Program control is transferred from one user-written application program to another at the same logical level in response to the

```
DFHPC TYPE=XCTL,                                              *
      PROGRAM=name
```

macro instruction. The program from which control is transferred is released. Any return from the transfered-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 Program Control.

The application programmer must provide the name (identification) of the program to which control is transferred. He can do this in either of two ways: (1) by including the PROGRAM=name operand in the DFHPC TYPE=XCTL macro instruction, or (2) by coding a single instruction, prior to issuing the DFHPC TYPE=XCTL macro instruction, that places the program name in the TCAPCPI field of the TCA. This same program name must also have been placed in the Processing Program Table (PPT) prior to execution of CICS.

The following is an example of the coding required to request a transfer of control to another application program:

```
DFHPC TYPE=XCTL,                                              *
      PROGRAM=PROG2
```

The following are examples of the coding required to dynamically transfer control to another application program.

For Assembler language:

```
MVC   TCAPCPI,=CL8'PROG2'  PLACE TRANSFERRED-TO PRGM NAME IN TCA
      .
      .
      .
DFHPC TYPE=XCTL            TRANSFER PROGRAM CONTROL
```

For ANS COBOL:

```
MOVE 'PROG2' TO TCAPCPI.   NOTE TRANSFERRED-TO PRGM NAME TO TCA.
      .
      .
      .
DFHPC TYPE=XCTL            TRANSFER PROGRAM CONTROL
```

For PL/I:

```
TCAPCPI='PROG2';           /*PLACE PROGRAM NAME IN TCA*/
      .
      .
      .
DFHPC TYPE=XCTL            TRANSFER PROGRAM CONTROL
```

LOAD THE SPECIFIED PROGRAM (LOAD)

Programs or tables are fetched from the library where they reside and are loaded into main storage in response to the

```
DFHPC TYPE=LOAD,                                                *
      PROGRAM=name,                                             *
      LOADLST=NO
```

macro instruction, identifying the program to be loaded. This facility is used to (1) load a program that will be used repeatedly, thereby reducing system overhead through a one-time load, and (2) load a table or some non-executable code to which control is definitely not to be passed.

The loaded program remains in main storage until the DFHPC TYPE=DELETE macro instruction is issued or until the transaction that issued the LOAD is terminated, either normally or abnormally (unless LOADLST=NO was specified). If the LOADLST=NO operand is used, the loaded program is to remain resident until it is deleted by the user. CICS returns the address of the loaded program to the user in the TCAPCLA field of the TCA. Note that the LOADLST operand is not available in the CICS/DOS-ENTRY system.

The application programmer must provide the name (identification) of the program to be loaded. He can do this in either of two ways: (1) by including the PROGRAM=name operand in the DFHPC TYPE=LOAD macro instruction, or (2) by coding a single instruction, prior to issuing the DFHPC TYPE=LOAD macro instruction, that places the program name in the TCAPCPI field of the TCA. This same program name must also have been placed in the Processing Program Table (PPT) prior to execution of CICS.

The following is an example of the coding required to load a user-written application program:

```
DFHPC TYPE=LOAD,                                                *
      PROGRAM=PROG3
```

The following are examples of the coding required to dynamically load user-written application programs.

For Assembler language:

```
MVC   TCAPCPI,=CL8'PROG3'   PLACE PROGRAM NAME IN TCA
      .
      .
      .
DFHPC TYPE=LOAD             LOAD THE SPECIFIED PROGRAM
```

For ANS COBOL:

```
MOVE 'PROG3' TO TCAPCPI.    NOTE PLACE PRGM NAME IN TCA.
      .
      .
      .
DFHPC TYPE=LOAD             LOAD THE SPECIFIED PROGRAM
```

For PL/I:

```
TCAPCPI='PROG3';          /*PLACE PROGRAM NAME IN TCA*/
      .
      .
      .
DFHPC TYPE=LOAD           LOAD THE SPECIFIED PROGRAM
```

RETURN PROGRAM CONTROL (RETURN)

Program control is returned to the next higher logical level in response to the

```
DFHPC TYPE=RETURN,                                          *
      TRANSID=transaction code
```

macro instruction. The execution of this macro instruction causes control to be returned to the program at the next higher logical level, restoring the registers and releasing save areas for the lower-level program. The program to which control is being returned must have relinquished control upon execution of a DFHPC TYPE=LINK macro instruction and must reside one logical level higher than the program returning control. Upon normal termination of transaction processing, control is returned to CICS Program Control. (Figure 12 shows how application programs are executed at various logical levels.)

The application programmer can, at his option, alter the transaction identification for the next program associated with that terminal in either of two ways: (1) by including the TRANSID=transaction code operand in the DFHPC TYPE=RETURN macro instruction, or (2) by coding a single instruction, prior to issuing the DFHPC TYPE TYPE=RETURN macro instruction, that places the new transaction identification in the TCANXTID field of the TCA.

Note that the TRANSID operand has no effect if a default transaction code has been assembled into the Terminal Control Table terminal entry (TCTTE).

Any higher-level program specifying a TRANSID overrides the TRANSID specification of a lower-level program. TCANXTID is unaltered if TRANSID is not coded.

The DFHPC TYPE=RETURN macro instruction can be used to terminate any tasks initiated by the application programmer through use of the Task Control (DFHKC) ATTACH macro instruction.

DELETE A LOADED PROGRAM (DELETE)

A program previously loaded through use of the DFHPC TYPE=LOAD, LOADLST=NO macro instruction is deleted (released) by the

```
DFHPC TYPE=DELETE,                                          *
      PROGRAM=name
```

macro instruction.

The application programmer must provide the name (identification) of the program to be deleted. He can do this in either of two ways: (1) by including the PROGRAM=name operand in the DFHPC TYPE=DELETE macro instruction, or (2) by coding a single instruction, prior to issuing the DFHPC TYPE=DELETE macro instruction, that places the program name in the TCAPCPI field of the TCA.

The following is an example of the coding required to delete a user-written application program previously loaded with the LOADLST=NO specification:

```
DFHPC TYPE=DELETE,                                              *
      PROGRAM=PROG4
```

The following are examples of the coding required to dynamically delete user-written application programs previously loaded with the LOADLST=NO specification.

For Assembler language:

```
MVC   TCAPCPI,=CL8'PROG8'  PLACE PROGRAM NAME IN TCA
      .
      .
      .
DFHPC TYPE=DELETE          DELETE THE SPECIFIED PROGRAM
```

For ANS COBOL:

```
MOVE 'PROG4' TO TCAPCPI.   NOTE PLACE PRGM NAME IN TCA.
      .
      .
      .
DFHPC TYPE=DELETE          DELETE THE SPECIFIED PROGRAM
```

For PL/I:

```
TCAPCPI='PROG4';           /*PLACE PROGRAM NAME IN TCA*/
      .
      .
      .
DFHPC TYPE=DELETE          DELETE THE SPECIFIED PROGRAM
```

ABNORMALLY TERMINATE A TRANSACTION (ABEND)

The application programmer can abnormally terminate a transaction and its related task by issuing the

```
DFHPC TYPE=ABEND,                                               *
      ABCODE=value,YES
```

macro instruction. In the situation where 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.

The application programmer can, at his option, request a dump of main storage related to the terminated transaction. He can accomplish this in either of two ways: (1) by including the ABCODE=value operand in the DFHPC TYPE=ABEND macro instruction, or (2) by coding a single instruction, prior to issuing the DFHPC TYPE=ABEND,ABCODE=YES macro instruction, that places a four-character abnormal termination code in the TCAPCAC field of the TCA. This abnormal termination code is placed in the output dump by Dump Control when providing the formatted storage dump and should be unique so as to be informative concerning the condition that caused the abend. If the ABCODE operand is not included in the DFHPC TYPE=ABEND macro instruction, no dump is taken.

The following is an example of the coding required to abnormally terminate a transaction and its related task and also request a dump of related main storage:

```
DFHPC TYPE=ABEND,                                              *
      ABCODE=1234
```

The following are examples of the coding required to dynamically terminate a transaction and its related task and at the same time request a dump of related main storage.

For Assembler language:

```
MVC   TCAPCAC,=CL4'1234'   PLACE TERMINATION CODE IN TCA
      .
      .
      .
DFHPC TYPE=ABEND,          TERMINATE PGRM, TRANS, & TASK       *
      ABCODE=YES           USE ABCODE ALREADY SPECIFIED
```

For ANS COBOL:

```
MOVE '1234' TO TCAPCAC.    NOTE TERMINATION CODE TO TCA.
      .
      .
      .
DFHPC TYPE=ABEND,          TERMINATE PGRM, TRANS, & TASK       *
      ABCODE=YES           USE ABCODE ALREADY SPECIFIED
```

For PL/I:

```
TCAPCAC='1234';            /*PLACE TERMINATION CODE IN TCA*/
      .
      .
      .
DFHPC TYPE=ABEND,          TERMINATE PGRM, TRANS, & TASK       *
      ABCODE=YES           USE ABCODE ALREADY SPECIFIED
```

DUMP SERVICES

Dump Management provides the capability, through Dump Control, to dump specified areas of main storage onto a sequential data set, either tape or disk. This data set contains only the information pertinent to the user's transaction or application program, and is subsequently formatted and printed offline (or while the dump data set is closed) using a CICS utility program (DFHDUP).

Requests for dump services are communicated to Dump Control via CICS macro instructions. Dump Control then executes, at the priority of the requesting program, under control of the requesting program's TCA, saving and restoring registers from this TCA. After the 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 with only one service request being 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 are placed in a "hold" status until the dump is completed. Remaining dump requests are serviced on a first in first out (FIFO) basis.

The Dump Management macro instruction (DFHDC) is used to request any of the following services:

1. Dump main storage areas related to a transaction and its associated task (or any other main storage areas).
2. Dump all CICS management modules and tables.
3. Dump transaction-oriented storage areas and CICS management modules and tables.
4. Dump selected main storage areas related to the requesting task.

The following operands can be included in the DFHDC macro instruction:

```
DFHDC TYPE=TRANSACTION,                                          *
      DMPCODE=value

DFHDC TYPE=CICS,                                                 *
      DMPCODE=value

DFHDC TYPE=COMPLETE,                                             *
      DMPCODE=value

DFHDC TYPE=PARTIAL,                                              *
      LIST=TERMINAL,PROGRAM,SEGMENT,TRANSACTION,                 *
      DMPCODE=value
```

Note: To ensure a dump of the TIOA following a Terminal Control write, the application programmer must issue a SAVE and WAIT with the DFHTC TYPE=WRITE macro instruction that precedes the DFHDC macro instruction. Since the Communications Area in the requesting task's TCA is used for Dump Control, the information provided in that area prior to the dump will be overlaid.

DUMP TRANSACTION STORAGE (TRANSACTION)

The application programmer can request the dump of all main storage areas related to a transaction and its associated task by issuing the

```
DFHDC TYPE=TRANSACTION,                                          *
      DMPCODE=value
```

macro instruction. This dump is normally used during the testing and debugging of user-written application programs. CICS automatically provides this service in the event the related task is abnormally terminated.

The following storage areas are dumped by CICS in response to the DFHDC TYPE=TRANSACTION macro instruction:

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. Task Extension Area (TXA)--applies only to the CICS/DOS-ENTRY system.
4. Trace Table.
5. Contents of general purpose registers upon entry to Dump Control from requesting task.
6. Either the Terminal Control Table terminal entry (TCTTE) or the Destination Control Table entry associated with the requesting task.
7. All transaction storage areas chained off the TCA storage accounting field.

8. All program storage areas containing user-written application program(s) active on behalf of the requesting task. (In the CICS/DOS-ENTRY system, only the program in main storage is dumped.)
9. Register save areas (RSA's) indicated by the RSA chain off the TCA.
10. All terminal storage areas (TIOA's) chained off the Terminal Control Table terminal entry (TCTTE) for the terminal associated with the requesting task (if any).

The application programmer can, at his option, provide a four-character dump code, which identifies the dump, by including the DMPCODE=value operand in the DFHDC TYPE=TRANSACTION macro instruction. This user-specified code is printed out with the requested dump and should be unique so as to be informative concerning the condition that caused the dump.

The following example illustrates the coding required to request a dump of transaction storage:

```
      DFHDC TYPE=TRANSACTION,     REQUEST TRANSACTION STORAGE DUMP     *
            DMPCODE=D010          USER-SPECIFIED DUMP CODE
```

DUMP CICS STORAGE (CICS)

The application programmer can request a dump of all CICS management modules and CICS control tables by issuing the

```
      DFHDC TYPE=CICS,                                                 *
            DMPCODE=value
```

macro instruction. This dump is typically used in a testing situation where the first dump taken is a CICS dump to ascertain the base of the test; subsequent dumps are usually of the TRANSACTION type.

The application programmer can, at his option, provide a four-character dump code, which identifies the dump, by including the DMPCODE=value operand in the DFHDC TYPE=CICS macro instruction. This user-supplied code is printed out with the requested dump and should be unique so as to be informative concerning the condition that caused the dump.

The following example illustrates the coding required to request a dump of CICS management modules and CICS control tables:

```
      DFHDC TYPE=CICS,            REQUEST CICS STORAGE DUMP          *
            DMPCODE=D020          USER-SPECIFIED DUMP CODE
```

DUMP TRANSACTION STORAGE AND CICS STORAGE (COMPLETE)

The application programmer can request a combination CICS and TRANSACTION dump by issuing the

```
      DFHDC TYPE=COMPLETE,                                             *
            DMPCODE=value
```

macro instruction. This dump might be appropriate if requested in limited numbers during execution of a task. Since CICS management modules and CICS control tables are primarily static areas, one CICS dump and a number of TRANSACTION dumps would be a more efficient testing aid than a comparable number of COMPLETE dumps.

The application programmer can, at his option, provide a four-character dump code, which identifies the dump, by including the DMPCODE=value operand in the DFHDC TYPE=COMPLETE macro instruction. This user-supplied code is printed out with the requested dump and should be unique so as to be informative concerning the condition that caused the dump.

The following example illustrates the coding required to request a combination CICS and TRANSACTION dump:

```
DFHDC TYPE=COMPLETE,          REQUEST COMPLETE STORAGE DUMP      *
      DMPCODE=D030            USER-SPECIFIED DUMP CODE
```

DUMP PARTIAL STORAGE (PARTIAL)

The application programmer can request a dump of selected main storage areas, related to the requesting task, by issuing the

```
DFHDC TYPE=PARTIAL,                                              *
      LIST=TERMINAL,PROGRAM,SEGMENT,TRANSACTION,                 *
      DMPCODE=value
```

macro instruction. This type of dump can be used during the testing and debugging of user-written application programs. It includes only those types of storage areas specified.

The application programmer must indicate what types of storage areas he wants dumped. He accomplishes this by specifying in the LIST operand of the DFHDC TYPE=PARTIAL macro instruction one or more of the following parameters: TERMINAL, PROGRAM, TRANSACTION, SEGMENT.

The application programmer can, at his option, provide a four-character dump code identifying the dump by including the DMPCODE=value operand in the DFHDC TYPE=PARTIAL macro instruction. This user-specified code is printed out with the requested dump and should be unique so as to be informative concerning the condition that caused the dump. If more than one parameter is included in the LIST operand, a single dump code can be used to identify the entire dump.

A discussion of the parameters that can be included in the LIST operand of the DFHDC TYPE=PARTIAL macro instruction follows.

TERMINAL: This keyword parameter is used to include in the PARTIAL dump all storage areas associated with the terminal. 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. Task Extension Area (TXA)--applies only to the CICS/DOS-ENTRY system.
4. Trace Table.
5. All terminal storage areas (TIOA's) chained off the Terminal Control Table terminal entry (TCTTE) for the terminal associated with the requesting task.
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.

The following example illustrates the coding required to request a PARTIAL storage dump including all terminal storage areas:

```
DFHDC TYPE=PARTIAL,          REQUEST PARTIAL STORAGE DUMP      *
      LIST=TERMINAL,         AREAS ASSOCIATED WITH TERMINAL    *
      DMPCODE=DTML           USER-SPECIFIED DUMP CODE
```

PROGRAM: This parameter is used to include in the PARTIAL dump all program storage areas associated with this task. These storage areas are:

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. Task Extension Area (TXA)--applies only to the CICS/DOS-ENTRY system.
4. Trace Table.
5. All program storage areas containing user-written application program(s) active on behalf of the requesting task.
6. Register save areas (RSA's) indicated by the RSA chain off the TCA.
7. Contents of general purpose registers upon entry to Dump Control from the requesting task.
8. Either the Terminal Control Table terminal entry (TCTTE) or the Destination Control Table entry associated with the requesting task.

The following example illustrates the coding required to request a PARTIAL storage dump including all program storage areas associated with this task:

```
DFHDC TYPE=PARTIAL,          REQUEST PARTIAL STORAGE DUMP      *
      LIST=PROGRAM,          PROGRAM STORAGE AREAS             *
      DMPCODE=DPGM           USER-SPECIFIED DUMP CODE
```

TRANSACTION: This parameter is used to include in the PARTIAL dump all transaction storage areas associated with this task, typically in combination with other types of storage areas such as TERMINAL or PROGRAM.

The following storage areas are dumped by CICS in response to the DFHDC TYPE=PARTIAL,LIST=TRANSACTION macro instruction:

1. Task Control Area (TCA) and, if applicable, the Transaction Work Area (TWA).
2. Common System Area (CSA), except for the user's portion of the CSA (CWA).
3. Task Extension Area (TXA)--applies only to the CICS/DOS-ENTRY system.
4. Trace Table.
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.
7. All transaction storage areas chained off the TCA storage accounting field.

The following example illustrates the coding required 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,          REQUEST PARTIAL STORAGE DUMP       *
      LIST=(TRANSACTION,     AREAS ASSOCIATED WITH TRANSACTION  *
      PROGRAM),              PROGRAM STORAGE AREAS              *
      DMPCODE=DT&P           USER-SPECIFIED DUMP CODE
```

SEGMENT: This parameter is used to include in the PARTIAL dump any area of main storage specified. For example, use of this parameter enables the application programmer to dump the area of main storage used for communication between the Terminal Abnormal Condition program (DFHTACP) and the Terminal Error program (DFHTEP). In addition, the following storage areas are provided:

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. Task Extension Area (TXA)--applies only to the CICS/DOS-ENTRY system.
4. Trace Table.
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.

The application programmer must code two instructions, prior to issuing the DFHDC TYPE=PARTIAL,LIST=SEGMENT macro instruction, that place the address of the main storage area to be dumped into the TCADCSA field of the TCA and the length of the area to be dumped into the TCADCNB field of the TCA.

The following are examples of the coding required to include in the PARTIAL dump any area of main storage.

For Assembler language:

```
ST    R5,TCADCSA             PLACE STORAGE ADDRESS IN TCA
MVC   TCADCNB,=H'16384'      PLACE LENGTH OF AREA IN TCA
      .
      .
      .
DFHDC TYPE=PARTIAL,          REQUEST PARTIAL STORAGE DUMP       *
      LIST=SEGMENT,          DUMP AREA PREVIOUSLY SPECIFIED     *
      DMPCODE=DMSA           USER-SPECIFIED DUMP CODE
```

For ANS COBOL:

```
MOVE R5 TO TCADCSA.          NOTE PLACE STRG ADDRESS IN TCA.
MOVE 16384 TO TCADCNB.       NOTE PLACE LENGTH OF AREA IN TCA.
      .
      .
      .
DFHDC TYPE=PARTIAL,          REQUEST PARTIAL STORAGE DUMP       *
      LIST=SEGMENT,          DUMP AREA PREVIOUSLY SPECIFIED     *
      DMPCODE=DMSA           USER-SPECIFIED DUMP CODE
```

For PL/I:

```
TCADCSA=R5;                  /*PLACE STORAGE ADDRESS IN TCA*/
TCADCNB=16384;               /*PLACE LENGTH OF AREA IN TCA*/
```

```
         .
         .
DFHDC TYPE=PARTIAL,        REQUEST PARTIAL STORAGE DUMP        *
      LIST=SEGMENT,        DUMP AREA PREVIOUSLY SPECIFIED      *
      DMPCODE=DMSA         USER-SPECIFIED DUMP CODE
```

TERMINAL SERVICES

Terminal Management provides communication between the terminals and user-written application programs through Terminal Control. Terminal Control is responsible for the polling and addressing of terminals, code translation, transaction initiation, task and line synchronization, and the line control necessary to read from or write to a terminal. The user-written application program is thus relieved, as much as possible, from having to control the physical terminal environment.

Requests for terminal services are communicated to Terminal Control via CICS macro instructions. However, when such requests are issued in an application program, Terminal Control is not entered directly. Instead, indicators are set in the Task Control Area (TCA) and in the Terminal Control Table (TCT) which allow Terminal Control to provide the requested service(s). Individual application programs thus interface with a terminal logically and symbolically.

Terminal Control operates as a system-provided task, contending with user-provided tasks in the system. It executes under control of its own TCA and is the highest-priority task in CICS. Terminal Control is always the first task to be dispatched by CICS; it scans the TCT and provides whatever services are requested.

The Basic Telecommunications Access Method (BTAM) is used by CICS for most terminal management. The Telecommunications Access Method (TCAM) can optionally be specified. However, the Sequential Access Method (SAM) is used where key-driven terminals are to be simulated by sequential devices such as a card reader. The Graphics Access Method (GAM) is used only in the CICS/OS-STANDARD system to support local 2260 terminals.

Note: The multipunched character 0-2-8 must be used in each physical input record immediately following the last data character to simulate the "end of block" (EOB). For sequential devices, the last entry in the input stream must be 'CSSF GOODNIGHT' to provide a logical close. For MFT/MVT users of the CICS/OS-STANDARD system having blocked SYSIN or SYSOUT, overriding DD cards must be provided for those CICS data sets being used to simulate terminals.

The Terminal Management macro instruction (DFHTC) is used to request any of the following services:

1. Write data to a terminal.
2. Read data from a terminal.
3. Synchronize terminal input/output for a transaction.
4. Converse with a terminal.
5. Transmit a page of data to a terminal.
6. Transmit to the common buffer of a 2980 General Banking Terminal System.
7. Test for the presence of a passbook in the 2980 General Banking Terminal System Models 1 and 4.

The following operands can be included in the DFHTC macro instruction:

```
DFHTC TYPE=(WRITE,WRITEL,READ,READL,WAIT,ERASE,SAVE,OIU,      *
           DISCONNECT,RESET,READB,COPY,ERASEAUP,CBUFF,        *
           PASSBK,TRANSPARENT,PSEUDOBIN,NOTRANSLATE),         *
      LINEADR=number,YES,                                     *
      CTLCHAR=hexadecimal number,YES,                         *
      DEST=symbolic name,YES,                                 *
      EOF=symbolic address

DFHTC TYPE=(GET,PUT,ERASE,SAVE,TRANSPARENT,PSEUDOBIN),        *
      LINEADR=number,YES,                                     *
      CTLCHAR=hexadecimal number,YES,                         *
      DEST=symbolic name,YES,                                 *
      EOF=symbolic address

DFHTC TYPE=(PAGE,SAVE),                                       *
      LINEADR=number,YES,                                     *
      CTLCHAR=hexadecimal number,YES,                         *
      DEST=symbolic name,YES

DFHTC TYPE=(CONVERSE,ERASE,SAVE),                             *
      LINEADR=number,YES,                                     *
      CTLCHAR=hexadecimal number,YES,                         *
      DEST=symbolic name,YES

DFHTC EOF=symbolic address
```

WRITE, WRITEL, READ, READL, WAIT, ERASE, SAVE, OIU, DISCONNECT, RESET, READB, COPY, ERASEAUP, TRANSPARENT, PSEUDOBIN, and NOTRANSLATE are optional keyword parameters and may be specified in any combination or in any order, as applicable. Each parameter, independent of its position, affects the setting of an associated bit in the Terminal Control Table Terminal Entry (TCTTE) so the order in which each parameter is specified has no effect on the meaning. For example, (WRITE, READ, SAVE) is equivalent to (WRITE, SAVE, READ) and (SAVE, WRITE, READ) etc. CBUFF and PASSBK are stand-alone parameters that have implied writes and waits. GET, PUT, PAGE, and CONVERSE are used for coding convenience; they are combinations of the other parameters as follows:

1. GET - same as READ, WAIT
2. PUT - same as WRITE, WAIT
3. PAGE - same as ERASE, WRITE, READ, WAIT
4. CONVERSE - same as WRITE, READ, WAIT

Note: When coding an application program in ANS COBOL, a WAIT must be included with every READ, READL, WRITE, WRITEL, READB, COPY, and ERASEAUP, except in the case of the final WRITE of the program.

The DISCONNECT parameter is used by the application programmer to break the line connection between the terminal and the computer; it applies only to switched lines. If the terminal is a buffered device, the data in the buffer(s) is lost.

The RESET parameter is used by the application programmer to relinquish use of the communication line; it applies only to binary synchronous terminals. When RESET is used, the next BTAM type of operation will be a read or write initial.

The READB parameter is applicable only to the 3270 Information Display System and is used by the application programmer to read the entire contents of the 3270 buffer. Data transmission starts at buffer location 0 and continues until the contents of the entire buffer have been read. All character and attribute sequences (including nulls)

appear in the input data stream in the same order as they occur in the 3270 buffer.

Note: Because of relatively long transmission times required to transmit the entire contents of a remote 3270 Information Display Station buffer, it is recommended that the READB parameter be used mainly for test and diagnostic purposes and that the COPY parameter be used, where possible, in all other cases. Excessive use of the READB parameter may cause degradation of performance on the line.

The COPY parameter is applicable only to the remote 3270 Information Display System and is used by the application programmer to copy the format and data contained in the buffer of another terminal attached to the same 3271 Control Unit. The physical address of the device to be copied is provided as the first and only character in the output data area (TIOADBA); TIOATDL must be set to 1. The Copy Control Character (CCC), which controls and defines the copy function to be performed, is supplied through the CTLCHAR operand. The COPY parameter cannot be included with a WRITE, ERASE, or ERASEAUP parameter in the same macro instruction.

The ERASEAUP parameter is applicable only to the 3270 Information Display System and is used by the application programmer to issue an "erase all unprotected" command. The following functions are performed in response to this command:

1. All unprotected fields are cleared to nulls (X'00').
2. The modified data tags in each unprotected field are reset to zero.
3. The cursor is positioned to the first unprotected field.
4. The keyboard is restored.

The ERASEAUP parameter cannot be included with a WRITE, ERASE, or COPY parameter in the same macro instruction. Note that no data stream is supplied for this command.

The CBUFF parameter is applicable only to the 2980 General Banking Terminal System and is used by the application programmer to place a message in the common buffer of the 2972 Terminal Control Unit. The 2972 associated with the current Terminal Control Table terminal entry (TCTTE) receives the output message.

Note: The output message is translated according to the 2980 model being described by the current TCTTE. If more than one model of the 2980 is attached to a 2972 Terminal Unit, the contents of the common buffer are intelligible only for the 2980 model for which the message was translated. Since shift characters are added to the message by CICS during translation, the message length is dependent upon the contents of the message. The maximum message length is 23 characters, including shift characters.

The PASSBK parameter is applicable only to the 2980 General Banking Terminal System and is used by the application programmer to cause output to be printed on a banking passbook. If a passbook is not present, printing does not occur and an error message is sent to the terminal operator.

The TRANSPARENT parameter is applicable only to the System/7 and is used by the application programmer to indicate that the data is not to be translated on either a READ or WRITE. For further information concerning System/7 programming considerations, see the section "Application Programming Considerations".

The PSEUDOBIN parameter is applicable only to the System/7 and is used by the application programmer to indicate that the data is to be translated on both a READ and WRITE. Translation is from System/7 pseudo-binary representation to hexadecimal representation on a READ, and from hexadecimal representation to System/7 pseudo-binary representation on a WRITE. For further information concerning System/7 programming considerations, see the section "Application Programming Considerations".

The NOTRANSLATE operand is applicable only to the 3735 Programmable Buffered Terminal, and is used by the application programmer to prevent translation of FDP records which are to be transmitted to a 3735 using ASCII transmission code. For further information, see the section "Application Programming Considerations".

The LINEADR operand is used to specify that writing is to begin on a specific line of a 2260 or 2265 screen. It is the responsibility of the application programmer to provide the hexadecimal equivalent of a line number in the range 1-12 (F0-FB) for the 2260 or 1-15 (F0-FE) for the 2265. He can accomplish this in either of two ways: (1) by including the LINEADR=number operand in the DFHTC macro instruction, or (2) by coding a single instruction, prior to issuing the DFHTC macro instruction, that places the line number in the TIOALAC field of the current TIOA. If the latter method is used, the LINEADR=YES operand must be included in the DFHTC macro instruction. For further information concerning the use of this operand, see the section "Application Programming Considerations".

The CTLCHAR operand is applicable only to the 3270 Information Display System. If a DFHTC TYPE=WRITE macro instruction is issued, this operand is used to provide the hexadecimal representation of the Write Control Character (WCC) which controls the requested write operation. If a DFHTC TYPE=COPY macro instruction is issued, this operand is used to provide the hexadecimal representation of the Copy Control Character (CCC) which controls and defines the copy function to be performed.

If CTLCHAR=YES is specified, the appropriate bit configuration must have been previously moved to the TIOACLCR field of the TIOA. If only the functions defined by the WCC are to be performed (that is, no data stream is to be supplied), the TIOATDL field of the TIOA must have been previously set to zero.

If the CTLCHAR operand is omitted, the following functions are assumed for the WCC and CCC.

WCC: Reset all modified data tags to zero.
Restore the keyboard.

CCC: Copy the contents of the entire buffer (including nulls).

The DEST operand is applicable only to TCAM. If a DFHTC TYPE=WRITE macro instruction is issued, the DEST operand can be used to send a message to a destination other than the source terminal. Typically this operand could be used to route messages to:

1. The master terminal (if TCAM is used)
2. A list of terminals if a TLIST macro was coded in the TCAM MCP.

The DFHTC TYPE=WRITE,DEST=symbolic name macro instruction determines the destination of the message by CICS placing the symbolic name in the four-byte TCTTE field labeled TCTTEDES for processing by the Terminal Control program. The DFHTC TYPE=WRITE,DEST=YES macro instruction allows the user to dynamically select a destination by placing the destination in the four-byte TCTTE field labeled TCTTEDES

prior to issuing the WRITE macro instruction. If DEST is not specified, the default action is to move the source terminal ID located in the TCTTETI field to the output message to provide a TCAM destination name, sending the message back to the source terminal.

The EOF=symbolic address operand is used to specify a routine in the application program which is to receive control when an end-of-file condition has been received on batch input from a 3735. The special initialization macro instruction, DFHTC EOF=symbolic address, has been provided to test for the end-of-file condition upon the initial connection to a 3735. This macro instruction must be included in the initialization section of the '3735' transaction before subsequent DFHTC macro instructions are issued.

Note: When the EOF condition occurs, the TIOATDL field of the TIOA passed to the application program contains binary zeros to indicate that the TIOA contains no valid data.

Applicable only to terminals attached to a 2848 Display Control Model 21 or 22, the READL and WRITEL parameters are used by the application programmer to control the locking and unlocking of the terminal keyboard after a read or write event. READL is applicable only to CICS/OS but may be used in CICS/DOS application programs if upward compatibility with CICS/OS is a consideration; it causes the keyboard to remain locked at the completion of data transfer. WRITEL causes the keyboard to remain locked if previously locked, or remain unlocked if previously unlocked. (WRITE always leaves the keyboard unlocked.)

If DFHTC macro instructions are issued in the following sequence, the keyboard is locked or unlocked as indicated:

	CICS/DOS	CICS/OS
READ	L	U
WRITEL	L	U
READL	L	L
READL	L	L
WRITEL	L	L
WRITEL	L	L
WRITE	U	U
WRITEL	U	U
WRITEL	U	U
READ	L	U
WRITE	U	U
READL	L	L
READ	L	U
WRITEL	L	U

Before terminal services can be requested in an application program via the DFHTC macro instruction, it is the responsibility of the application programmer to provide instructions that do the following:

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 the section "Storage Definition".)
2. Establish addressability for the TCTTE and TIOA by specifying a symbolic base address for the TCTTE and TIOA, respectively. The application programmer must obtain the base address of the TCTTE from the TCAFCAAA field of the TCA and place it at TCTTEAR. Having established addressability to the TCTTE, he must obtain the base address of the TIOA from the TCTTEDA field of the TCTTE

and place it at TIOABAR. The application programmer now has access by field name to any field in the TCTTE or TIOA.

CICS allows one or more TIOA's to be associated with a terminal at a given time. If a TIOA is obtained in an application program via the DFHSC TYPE=GETMAIN,CLASS=TERMINAL macro instruction, the address of the TIOA obtained is automatically placed in the TCASCSA field of the TCA. The application programmer must set up a base register for this TIOA and must place the address at TCASCSA into the base register.

The length of the data to be read or written into a given TIOA is found in the TIOATDL field of that TIOA. On a read operation, this two-byte binary value is placed in the TIOATDL field by Terminal Control and represents the number of bytes actually read. On a write operation, the application programmer must assign to the TIOATDL field, prior to issuing the DFHTC TYPE=WRITE macro instruction, the number of bytes to be written.

Note: All TIOA's have a twelve-byte prefix for storage accounting and terminal control and a one-byte EOB suffix. The value at TIOATDL does not include these 13 bytes.

Given an idle line, CICS always initiates a write before polling to read.

The following are examples of the coding required to (1) acquire an output storage area via the DFHSC macro instruction, (2) place the address of the storage area acquired into TCTTEDA, (3) place the length of the data to be written into TIOATDL, (4) issue a write to a 2260/2265 terminal, erasing the screen and returning the cursor to the upper left corner of the screen before writing, and (5) issue a read from a terminal, allowing Terminal Control to manage storage for the TIOA.

For Assembler language:

```
L      TCTTEAR,TCAFCAAA       ESTABLISH ADDRESSABILITY FOR TCTTE
DFHSC  TYPE=GETMAIN,          OBTAIN TIOA FOR OUTPUT DATA          *
       NUMBYTE=80,                                                 *
       CLASS=TERMINAL
L      TIOABAR,TCASCSA        ADDRESS OF TIOA
ST     TIOAEAR,TCTTEDA        PLACE OUTPUT ADDRESS IN TCTTE
MVC    TIOADBA(80),DATA       PLACE DATA IN TIOA
MVC    TIOATDL,=H'80'         PLACE DATA LENGTH IN TIOATDL
         .
         .
         .
DFHTC  TYPE=(WRITE,ERASE,     ISSUE WRITE TO 2260/2265 TERMINAL    *
       READ,WAIT)             ERASE BEFORE WRITE, THEN READ
L      TIOAEAR,TCTTEDA        ESTABLISH ADDRESSABILITY FOR TIOA
```

For ANS COBOL:

```
MOVE TCAFCAAA TO TCTTEAR.     NOTE EST ADDRESSABILITY FOR TCTTE.
DFHSC TYPE=GETMAIN,           OBTAIN TICA FOR OUTPUT DATA          *
      NUMBYTE=80,                                                  *
      CLASS=TERMINAL
MOVE TCASCSA TC TIOABAR.      NOTE ADDRESS OF TIOA.
MOVE TIOABAR TO TCTTEDA.      NOTE PLACE ADDR OF TIOA IN TCTTE.
MOVE DATA TO TIOADBA.         NOTE PLACE DATA IN TIOA.
MOVE 80 TO TIOATDL.           NOTE PLACE DATA LENGTH IN TIOATDL.
         .
         .
         .
```

```
DFHTC TYPE=(WRITE,ERASE,      ISSUE WRITE TO 2260/2265 TERMINAL   *
      READ,WAIT)              ERASE BEFORE WRITE, THEN READ
MOVE TCTTEDA TO TICAEAR       NOTE EST ADDRESSABILITY FOR TIOA.
```

For PL/I:

```
TCTTEAR=TCAFCAAA;             /*EST ADDRESSABILITY FOR TCTTE*/
DFHSC TYPE=GETMAIN,           OBTAIN TICA FOR OUTPUT DATA         *
      NUMBYTE=80,                                                 *
      CLASS=TERMINAL
TIOAEAR=TCASCSA;              /*ADDRESS OF TIOA*/
TCTTEDA=TICAEAR;              /*PLACE ADDR OF TICA IN TCTTE*/
TIOADBA=DATA;                 /*PLACE DATA IN TIOA*/
TIOATDL=80;                   /*PLACE DATA LENGTH IN TIOATDL*/
      .
      .
      .
DFHTC TYPE=(WRITE,ERASE,      ISSUE WRITE TO 2260/2265 TERMINAL   *
      READ,WAIT)              ERASE BEFORE WRITE, THEN READ
TIOAEAR=TCTTEDA;              /*EST ADDRESSABILITY FOR TIOA*/
```

WRITE DATA TO A TERMINAL (WRITE)

The application programmer can request that data be written to a terminal by issuing the

```
DFHTC TYPE=WRITE
```

macro instruction. Before issuing this macro instruction, he has the responsibility to (1) place the address of the TIOA to be written into the TCTTEDA field of the TCTTE, and (2) place the length of the data to be written into the TICATDL field of the TIOA. (It is assumed that he has also symbolically defined the CSA, TCA, and TCTTE and has established addressability for the TCTTE.)

When the write is completed by Terminal Control, the TIOA is released to the dynamic storage pool (unless SAVE is specified) since it is understood that the application program has no further use for it. Any subsequent reference to this TIOA by the application program is logically in error and will produce unpredictable results.

A TIOA can be reused by the application program after a write if the request to write data to a terminal is made via the

```
DFHTC TYPE=(WRITE,SAVE,WAIT)
```

macro instruction. In this case the TIOA is not released by Terminal Control. The WAIT parameter is needed to ensure that the write of the TIOA is complete before the area is reused.

Note: To ensure a dump of the TIOA following a Terminal Control write, the application programmer must issue a SAVE and WAIT with the DFHTC TYPE=WRITE macro instruction that precedes the DFHDC macro instruction.

The application programmer can specify both a write and read operation in a single request by issuing the

```
DFHTC TYPE=(WRITE,READ)
```

macro instruction. When this instruction is executed, Terminal Control writes the TIOA whose address is at TCTTEDA, waits for that write to be completed (an implied wait), and then issues a read from the terminal

into the area just used for writing. Since the SAVE parameter was not specified, one TIOA can be used repeatedly. However, a new TIOA is obtained for the read operation and its address placed in TCTTEDA when certain devices are involved or when certain conditions exist. For example:

1. 2260 terminals (local and remote)
2. Local 3270 terminals
3. PSEUDOBIN is psecified with READ, WRITE
4. If the TIOA length for the WRITE instruction is less than that specified in the DFHTCT TYPE=LINE,TIOAL=length specification (binary synchronous terminals) or in the DFHTCT TYPE=LINE, INAREAL=length specification (all other terminals)
5. Certain error conditions
6. Using a 3270 terminal in 2260 compatibility mode
7. Using terminals with TCAM (CICS/OS only)

Thus the user shculd always reload TIOABAR from TCTTEDA following the (WRITE,READ) macro instruction. A typical use for the DFHTC TYPE=(WRITE,READ) macro instruction is a conversational environment where the application program writes a question to the terminal, waits for a response, and then reads the response.

Note: In the case of a terminal connected to the 7770 Audio Response Unit, a read request that does not include the WRITE parameter causes the "ready" message previously defined in the Terminal Control Table to be written to the terminal before the read operation occurs.

If both a write and read operation are specified in a single request by issuing the

```
DFHTC TYPE=(WRITE,READ,SAVE)
```

macro instruction, the TIOA used for writing is saved; a new TIOA is then dynamically acquired by Terminal Control for the read. The saved area remains chained off the TCTTE for the terminal involved and can be reused at a later time for either writing or reading. If this TIOA is reused later, the application programmer must place the address of the TIOA into the TCTTEDA field of the TCTTE prior to issuing the request to use the area.

The manner in which the address of a TIOA is "remembered" is determined by the application programmer. For example, he can store the address in the TWA, or he can rely on the area being accounted for in the TIOA storage accounting chain off the TCTTE.

Upon completion of a WRITE, READ, SAVE, the application programmer must place the value contained at TCTTEDA into TIOABAR to establish addressability for the newly acquired TIOA.

Note: A WRITE, READ, SAVE may not be usable for the application in which the initial TIOA is small, as determined by the user in the Terminal Control Table line entry (TCTLE) during system initialization for this line, and in which subsequent TIOA's acquired dynamically by CICS are of larger or varying size. There is no problem if the user always works with TIOA's of the same size.

If a write to a 2260/2265 terminal is specified by issuing the

```
DFHTC TYPE=(WRITE,ERASE)
```

macro instruction, the screen is erased and the cursor is returned to the upper left ccrner of the screen before writing occurs. If the ERASE parameter is omitted, writing begins wherever the cursor is located at the time the write is issued.

Note: The ERASE parameter may be used in conjunction with either the WRITE or WRITEL parameters; it may not be used separately. To simply erase the screen, the application programmer might (1) place at TCTTEIA the address of storage that contains only a start symbol, and (2) issue a DFHTC TYPE=(WRITE,ERASE) macro instruction.

The applicatiCn programmer can request the positioning of frames for a 2760 Optical Image Unit by issuing the

```
DFHTC TYPE=(WRITE,OIU)
```

macro instruction. See the System Programmer's Reference Manual for an example of an applicatiCn program executed as a 2760 transaction.

When TCAM is used, the applicatiCn prcgrammer issues the

```
DFHTC TYPE=WRITE,                                                    *
      DEST=symbolic name, YES
```

macro instruction. See the previous discussion of the DEST operand near the beginning of the section "Terminal Services".

READ DATA FROM A TERMINAL (READ)

The application programmer can request that data be read from a terminal by issuing the

```
DFHTC TYPE=READ
```

macro instruction. Before issuing this macro instructicn, he can place the address of the TIOA into the TCTTE.

If a TIOA is nct provided by the application program, Terminal Control attempts to use existing storage if a TIOA is attached to the TCTTE, or, if a TIOA is not attached, Terminal Control acquires a new TIOA. If the length of the existing TIOA or length of the TIOA provided by the applicatiCn program is not adequate, or if other conditions exist that make the TIOA unusable, the application program must always place the value contained at TCTTEDA into TIOABAR following completion cf the read to ensure addressability to the correct TIOA.

A new TICA is acquired by Terminal Contrcl for the read when the

```
DFHTC TYPE=(READ,SAVE)
```

macro instruction is issued. All TIOA's currently chained off the TCTTE are retained and may be subsequently reused; a new TIOA is dynamically acquired for this read (according to the length specified in the TCTLE) and is added to the chain.

Upon completicn of a READ, SAVE, the application programmer must place the value contained at TCTTEDA into TIOABAR to establish addressability for the newly acquired TIOA. The number of bytes read is provided by CICS at TIOATDL.

A read and write operatiCn can be specified in a single request, as discussed in the previous topic, "Write Data to a Terminal".

SYNCHRONIZE TERMINAL INPUT/OUTPUT FOR A TRANSACTION (WAIT)

In a transaction where more than one terminal operation is to be performed, the application programmer must ensure that the current terminal operation is complete before another begins. He can accomplish this by issuing the

```
DFHTC TYPE=WAIT
```

macro instruction, where the WAIT parameter is coded separately, as shown, or in combination with READ and/or WRITE. A PUT can be coded in place of a WRITE,WAIT; a GET can be coded in place of a READ,WAIT. A wait should be issued for each read request to ensure that the data has been transferred into the TIOA.

A wait causes the execution of a task (transaction) to be temporarily suspended. Indicators are set in the TCT and control is returned to CICS. The task resumes processing when the write and/or read is 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 it is written, even though the transaction and its associated storage may have already been deleted from the system.

CONVERSE WITH A TERMINAL (CONVERSE)

The application programmer can request a conversational mode of communication with the terminal by issuing the

```
DFHTC TYPE=CONVERSE
```

macro instruction, where CONVERSE (or CONV) is the same as WRITE, READ, WAIT. The execution of this instruction is always in the sequence: WRITE, implied wait, READ, WAIT. In the case of 2260/2265 terminals, writing begins wherever the cursor is located at the time this macro instruction is issued.

PAGE DATA TO A TERMINAL (PAGE)

The application programmer can request a conversational mode of communication with a 2260/2265 terminal by issuing the

```
DFHTC TYPE=PAGE
```

macro instruction, where PAGE is the same as ERASE, WRITE, READ, WAIT.

FILE SERVICES

File Management provides the capability, through File Control, to read a record from an existing data set (file) on a direct access device, update an existing record in a data set, and add a new record to a data set. Facilities supported by File Control include indirect access, browsing, "duplicates" data sets, and segmented records. Note that while File Services supports the user's data base, Transient Data Services supports sequential data sets.

Access methods supported by File Control are the Direct Access Method (DAM) and the Indexed Sequential Access Method (ISAM). DAM can be used for fixed- or variable-length records, for blocked or unblocked records, and for undefined records. If the user creates DAM data sets and describes them to CICS through the File Control Table (FCT), application programs can access those data sets on a logical

record level with File Control providing the blocking/deblocking service.

Optional access to the Data Language/I (DL/I) facility of the IBM Information Management System (IMS/360) is also provided under CICS/OS. See the section "Requesting Data Language/I Services" for information concerning the use of DL/I in a CICS application program.

All storage needed for data set operations is acquired by File Control in accordance with the data set descriptions previously supplied by the user in the FCT. The application programmer need only be concerned with the logical record and not with the other characteristics of the data set.

An application program always operates on data in one of two main storage areas: (1) a File Input/Output Area (FIOA) or (2) a File Work Area (FWA). A FIOA is required to handle records that are read-only and unsegmented or unblocked. A FWA is required to handle records that are new, segmented, blocked, or to be updated. In addition, a FWA is always used in a browse operation.

Requests for file services are communicated to File Control via CICS macro instructions. File Control then executes, at the priority of the requesting program, under control of the requesting program's TCA, saving and restoring registers from this TCA. After the requested file service has been provided (or attempted), control is returned to the next executable instruction in the requesting program. Upon return to the requesting program, tests can be made and control routed to various user-written exception handling routines based on the outcome of the requested file service.

The File Management macro instruction (DFHFC) is used to request any of the following services:

1. Randomly retrieve data from a data set.
2. Randomly update or add data to a data set.
3. Obtain a main storage area to create a new record.
4. Release main storage area.
5. Check the response to a request for file services.
6. Initiate a browse operation.
7. Sequentially retrieve data from a data set (browse).
8. Terminate a browse operation.
9. Reset the starting location of a browse operation.
10. Release an update request.

The following operands can be included in the DFHFC macro instruction:

```
DFHFC TYPE=GET,                                          *
      DATASET=symbolic name,                             *
      RDIDADR=symbolic address,                          *
      SEGSET=symbolic name,YES,ALL,                      *
      INDEX=symbolic name,YES,                           *
      TYPOPER=UPDATE,                                    *
      RETMETH=RELREC,KEY,                                *
      NORESP=symbolic address,                           *
      DSIDER=symbolic address,                           *
      SEGIDER=symbolic address,                          *
      NOTFND=symbolic address,                           *
      INVREQ=symbolic address,                           *
      IOERROR=symbolic address,                          *
      DUPDS=symbolic address,                            *
      NOTOPEN=symbolic address
```

```
DFHFC TYPE=PUT,                                                 *
      RDIDADR=symbolic address,                                 *
      SEGSET=YES,                                               *
      TYPOPER=NEWREC,UPDATE,                                    *
      NORESP=symbolic address,                                  *
      DUPREC=symbolic address,                                  *
      INVREQ=symbolic address,                                  *
      IOERROR=symbolic address,                                 *
      NOSPACE=symbolic address,                                 *
      NOTOPEN=symbolic address

DFHFC TYPE=GETAREA,                                             *
      DATASET=symbolic name,                                    *
      INITIMG=value,YES,                                        *
      DSIDER=symbolic address,                                  *
      NORESP=symbolic address,                                  *
      INVREQ=symbolic address,                                  *
      NOTOPEN=symbolic address

DFHFC TYPE=RELEASE,                                             *
      INVREQ=symbolic address

DFHFC TYPE=SETL,                                                *
      DATASET=symbolic name,                                    *
      RDIDADR=symbolic address,                                 *
      SEGSET=symbolic name,YES,ALL,                             *
      RETMETH=RELREC,KEY,                                       *
      NORESP=symbolic address,                                  *
      DSIDER=symbolic address,                                  *
      SEGIDER=symbolic address,                                 *
      INVREQ=symbolic address,                                  *
      NOTOPEN=symbolic address

DFHFC TYPE=GETNEXT,                                             *
      SEGSET=symbolic name,YES,ALL,                             *
      NORESP=symbolic address,                                  *
      SEGIDER=symbolic address,                                 *
      INVREQ=symbolic address,                                  *
      IOERROR=symbolic address,                                 *
      NOTOPEN=symbolic address,                                 *
      ENDFILE=symbolic address

DFHFC TYPE=ESETL,                                               *
      INVREQ=symbolic address

DFHFC TYPE=RESETL,                                              *
      SEGSET=symbolic name,YES,ALL,                             *
      NORESP=symbolic address,                                  *
      SEGIDER=symbolic address

DFHFC TYPE=CHECK,                                               *
      NORESP=symbolic address,                                  *
      DSIDER=symbolic address,                                  *
      SEGIDER=symbolic address,                                 *
      NOTFND=symbolic address,                                  *
      DUPREC=symbolic address,                                  *
      INVREQ=symbolic address,                                  *
      IOERROR=symbolic address,                                 *
      DUPDS=symbolic address,                                   *
      NOSPACE=symbolic address,                                 *
      NOTOPEN=symbolic address,                                 *
      ENDFILE=symbolic address
```

RANDOMLY RETRIEVE DATA FROM A DATA SET (GET)

The application programmer can randomly retrieve data from a data set (file) by issuing the

```
DFHFC TYPE=GET,                                          *
      DATASET=symbolic name,                             *
      RDIDADR=symbolic address,                          *
      SEGSET=symbolic name,YES,ALL,                      *
      INDEX=symbolic name,YES,                           *
      TYPOPER=UPDATE,                                    *
      RETMETH=RELREC,KEY,                                *
      NORESP=symbolic address,                           *
      DSIDER=symbolic address,                           *
      SEGIDER=symbolic address,                          *
      NOTFND=symbolic address,                           *
      INVREQ=symbolic address,                           *
      IOERROR=symbolic address,                          *
      DUPDS=symbolic address,                            *
      NOTOPEN=symbolic address
```

macro instruction. This macro instruction is used for random read-only (inquiry) or update operations. The requested data record is returned in a File Input/Output Area (FIOA) for read-only operations with unsegmented, unblocked records; the data record is returned in a File Work Area (FWA) for update operations or for read-only operations with segmented or blocked records.

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

1. Acquires the main storage area required to read a record.
2. Reads the requested data.
3. Locates the requested logical record .

In addition, CICS can perform the following services, depending on the operands that are included in the DFHFC TYPE=GET macro instruction:

1. Retrieve a record indirectly.
2. Segment a record for inquiry (read only) and return the requested segments in a work area.
3. Acquire a File Work Area of the same length as the requested record when the record is to be updated or when records are blocked or segmented.
4. Unpack a segmented record into a work area of the same length as the requested record.

Before file services can be requested in an application program via the DFHFC TYPE=GET macro instruction, the user must have previously defined in the File Control Table (FCT) all data sets referenced by the DATASET and INDEX keyword parameters and all segment sets referenced by the SEGSET keyword parameter. Instructions must have been provided in the application program that symbolically define the FIOA and/or FWA by (1) copying the appropriate CICS control section definitions (DFHFIOA and DFHFWADS) provided by CICS, and (2) providing his own storage definitions for the user's section of the FIOA and/or FWA. If ISAM data is being retrieved under CICS/OS, a 16-byte filler must be defined prior to the user's data definition.

The application programmer must specify the parameters he requires to retrieve data from a data set. He can do this in either of two ways: (1) by including the parameters in operands of the DFHFC TYPE=GET macro instruction, or (2) by coding instructions, prior to issuing the DFHFC TYPE=GET macro instruction, that dynamically move these parameters to fields in the TCA. If the parameters are included in

operands of the DFHFC TYPE=GET macro instruction, the applicable keywords are DATASET, RIDFLD, SEGSET, INDEX, TYPOPER, and RETMETH.

After file services have been requested in an application program, addressability must be established for the required FIOA and/or FWA. The address of the area involved, provided by CICS at TCAFCAA, must be placed at FIOABAR and/or FWACBAR. The user may issue a DFHSC TYPE=FREEMAIN or a DFHFC TYPE=RELEASE macro instruction to free the FIOA or FWA, otherwise CICS will free the area upon task termination.

If the application programmer desires to check the response to his request to retrieve data from a data set, he must specify the entry labels (symbolic addresses) he requires to access user-written exception handling routines. He can do this in any of three ways: (1) by including the entry labels in operands of the DFHFC TYPE=GET macro instruction, (2) by coding instructions immediately following the DFHFC TYPE=GET macro instruction that examine the response code provided by CICS at TCAFCTR (TCAFCRC if the language is ANS COBOL) and transfer control to the appropriate routine, or (3) by including the entry labels in the DFHFC TYPE=CHECK macro instruction (which must immediately follow the DFHFC TYPE=GET macro instruction). In any case, the applicable keywords are NORESP, DSIDER, SEGIDER, NOTFND, INVREQ, IOERROR, DUPDS, and NOTOPEN.

A discussion of the operands that can be included in the DFHFC TYPE=GET macro instruction follows. (The keywords used to access user-written exception handling routines are discussed in the section "Test Response to a Request for File Services".)

DATASET: This operand is used to specify the symbolic name of the primary data set to be accessed. The symbolic name must have been previously defined in the File Control Table (FCT). If the operation involves indirect accessing, the symbolic data set name specified by this operand represents the primary (target) data set from which a record is to be retrieved. This operand can be omitted if the application programmer has previously placed the symbolic name in the TCAFCDI field of the TCA.

RIDFLD: This operand is used to specify the symbolic address of the user's Record Identification field that contains the key of the record to be retrieved, as required by ISAM, or the block reference field, as required by DAM. This operand can be omitted if the application programmer has previously placed the address of the field in the TCAFCRI field of the TCA. For further details, see the section "Data Base Considerations".

Note: There must be a unique Record Identification field for each data set that is to be concurrently updated by a single application program. Because CICS application programs must have the quality of quasi-reentrance, it is highly recommended that the Record Identification field not reside in the application program.

SEGSET: The SEGSET=name operand is used to specify the symbolic name of the segment set to be retrieved. The symbolic name must have been previously defined in the associated Segment Control section of the File Control Table (FCT).

SEGSET=YES is used when reading a segmented record if the application programmer has previously placed the symbolic name of the segment set in the TCAFCSI field of the TCA.

SEGSET=ALL is used when reading a segmented record if the entire logical record is desired in an unpacked and aligned format. SEGSET=ALL is automatically used by CICS when updating a segmented record. The entire logical record is unpacked and returned to the application program.

If the SEGSET operand is omitted, and the GET is from a segmented data set, the logical record is returned in its packed format.

INDEX: The INDEX=name operand specifies the symbolic name of the highest level index data set used in an Indirect Access hierarchy of ISAM and/or DAM data sets. (This index data set is the first data set accessed in the hierarchy.) The symbolic name must have been previously defined in the FCT. INDEX=YES must be coded if the application programmer has previously placed the symbolic name in the TCAFCAI field of the TCA. If the index data set is a DAM data set, it cannot be blocked.

TYPOPER: The TYPOPER=UPDATE operand is used when a record is to be obtained for updating. To free the area used, a DFHFC TYPE=RELEASE or DFHFC TYPE=PUT must be issued. If the TYPOPER=UPDATE operand is omitted, a read-only operation is assumed. If the record being updated is from a blocked DAM data set, the RETMETH operand must also be specified. If the application program is to update more than one data set concurrently, a separate Record Identification field (RDIDADR) must be specified for each update request.

RETMETH: The RETMETH operand applies only to blocked DAM data sets and is used to specify the argument type (deblocking method) for the deblocking of the data sets. The RETMETH=RELREC operand specifies that retrieval is to occur by relative record, where the first record in a block is record zero. The RETMETH=KEY operand specifies that retrieval is to occur by key. The RETMETH operand must be specified if TYPOPER=UPDATE is present.

If the RETMETH keyword is omitted and a request to read a blocked DAM data set is issued, the entire physical record (block) is returned to the requesting program in the FIOA.

The user's block reference field, required by DAM, contains the criteria for the deblocking of DAM data sets. For further details, see the section "Data Base Considerations".

Note: If the record being retrieved is "undefined", it is the user's responsibility to determine the length of the record.

The following are examples of the coding required to do a random read-only (inquiry) operation on a record of the master data set, assuming blocked or segmented records.

For Assembler language:

```
        COPY  DFHTCADS              COPY TCA SYMBOLIC STRG DEFN
KEY     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   KEY,ACCTNO            MOVE RECORD IDENT TO KEY FIELD
READREC DFHFC TYPE=GET,             GET RECORD FROM MASTER DATA SET     *
              DATASET=MASTERA,                                          *
              RDIDADR=KEY
        L     FWACBAR,TCAFCAA       ESTABLISH ADDRESSABILITY FOR FWA
```

For ANS COBOL:

```
    02  FWACBAR PICTURE S9(8) USAGE IS COMPUTATIONAL.
        .                           NOTE DEFINE BASE REGISTER FOR FWA.
        .
        .
01  DFHTCADS COPY DFHTCADS.        NOTE COPY SYMBOLIC STRG DEFN FOR TCA.
    02  KEYF PICTURE X(8).         NOTE DEFINE KEY FIELD IN TWA.
        .
        .
        .
01  DFHFWADS COPY DFHFWADS.        NOTE COPY SYMBOLIC STRG DEFN FOR FWA.
    02  RECORD PICTURE 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.
```

For PL/I:

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

The following are examples of the coding required to randomly retrieve a record for update on the master data set.

For Assembler language:

```
        COPY  DFHTCADS              COPY TCA SYMBOLIC STRG DEFN
KEY     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   KEY,ACCTNO            MOVE RECORD IDENT TO KEY FIELD
READUPD DFHFC TYPE=GET,             GET RECORD FROM MASTER DATA SET     *
              DATASET=MASTERA,      FOR UPDATE                          *
              RDIDADR=KEY,                                              *
              TYPOPER=UPDATE
        L     FWACBAR,TCAFCAA       ESTABLISH ADDRESSABILITY FOR FWA
```

For ANS COBOL:

```
    02  FWACBAR PICTURE S9(8) USAGE IS COMPUTATIONAL.
        .                           NOTE DEFINE BASE REGISTER FOR FWA.
        .
        .
01  DFHTCADS COPY DFHTCADS.        NOTE COPY SYMBOLIC STRG DEFN FOR TCA.
    02  KEYF PICTURE X(8).         NOTE DEFINE KEY FIELD IN TWA.
        .
        .
        .
01  DFHFWADS COPY DFHFWADS.        NOTE COPY SYMBOLIC STRG DEFN FOR FWA.
    02  RECORD PICTURE 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 FIELD.
READREC.
        DFHFC TYPE=GET,             GET RECORD FROM MASTER DATA SET     *
              DATASET=MASTERA,                                          *
              RDIDADR=KEYF
    MOVE TCAFCAA TO FWACBAR.        NOTE ESTABLISH FWA ADDRESSABILITY.
```

For PL/I:

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

The following are examples of the coding required to randomly retrieve a record for update where the key for the desired record is unknown. A cross-index data set containing the master key is available, making it possible to access the record indirectly.

For Assembler language:

```
        COPY  DFHTCADS              COPY TCA SYMBOLIC STRG DEFN
KEY     DS    CL25                  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   KEY,INDEXA            MOVE INDEX IDENT TO KEY FIELD
READING DFHFC TYPE=GET,             GET RECORD FROM MASTER DATA SET     *
              DATASET=MASTERA,      BY FIRST ACCESSING A CROSS-INDEX    *
              RIDADR=KEY,           DATA SET NAMED INDIRECT             *
              TYPOPER=UPDATE,                                           *
              INDEX=INDIRECT
        L     FWACBAR,TCAFCAA       ESTABLISH ADDRESSABILITY FOR FWA
```

For ANS COBOL:

```
    02  FWACBAR PICTURE S9(8) USAGE IS COMPUTATIONAL.
        .                           NOTE DEFINE BASE REGISTER.
        .
        .
01  DFHTCADS COPY DFHTCADS         NOTE COPY SYMBOLIC STRG DEFN FOR TCA.
    02  KEY PICTURE X(25).         NOTE DEFINE KEY FIELD IN TWA.
        .
        .
        .
01  DFHFWADS COPY DFHFWADS.        NOTE COPY SYMBOLIC STRG DEFN FOR FWA.
    02  RECORD PICTURE X(350).     NOTE DEFINE RECORD LAYOUT IN FWA.
        .
        .
        .
PROCEDURE DIVISION.
    MOVE CSACDTA TO TCACBAR.        NOTE ESTABLISH TCA ADDRESSABILITY.
        .
        .
        .
    MOVE PARTNAME TO KEY.           NOTE MOVE INDEX IDENT TO KEY.
READREC.
       DFHFC TYPE=GET,              GET RECORD FROM MASTER DATA SET     *
             DATASET=MASTERA,       BY FIRST ACCESSING A CROSS-INDEX    *
             RIDADR=KEY,            DATA SET NAMED INDEXAB              *
             TYPOPER=UPDATE,                                            *
             INDEX=INDEXAB
    MOVE TCAFCAA TO FWACBAR.        NOTE ESTABLISH FWA ADDRESSABILITY.
```

For PL/I:

```
%INCLUDE DFHTCADS;                    /*COPY SYMBOLIC STRG DEFN FOR TCA*/
   02 KEY CHAR(25);                   /*DEFINE KEY FIELD IN TWA*/
%INCLUDE DFHFWADS;                    /*COPY SYMBOLIC STRG DEFN FOR FWA*/
   02 RECORD CHAR(350);               /*DEFINE RECORD LAYOUT IN FWA*/
        .
        .
        .
KEY=PARTNAME;                         /*ASSIGN RECORD INDENT TO KEY FIELD*/
READREC:
      DFHFC TYPE=GET,                 GET RECORD FROM MASTER DATA SET     *
            DATASET=MASTERA,          BY FIRST ACCESSING A CROSS-INDEX    *
            RDIDADR=KEY,              DATA SET NAMED INDEXAB              *
            TYPOPER=UPDATE,                                               *
            INDEX=INDEXAB
FWACBAR=TCAFCAA;                      /*ESTABLISH ADDRESSABILITY FOR FWA*/
```

RANDOMLY UPDATE OR ADD DATA TO A DATA SET (PUT)

The application programmer can randomly update or add data to a data set (file) by issuing the

```
      DFHFC TYPE=PUT,                                                     *
            RDIDADR=symbolic address,                                     *
            SEGSET=YES,                                                   *
            TYPOPER=NEWREC,UPDATE,                                        *
            NORESP=symbolic address,                                      *
            DUPREC=symbolic address,                                      *
            INVREQ=symbolic address,                                      *
            IOERROR=symbolic address,                                     *
            NOSPACE=symbolic address,                                     *
            NOTOPEN=symbolic address
```

macro instruction. This macro instruction is used to (1) update an existing record, which has been previously retrieved via the DFHFC TYPE=GET,TYPOPER=UPDATE macro instruction, or (2) add a new record to an existing data set. Note that a DFHFC TYPE=PUT macro instruction must never be issued without first issuing a DFHFC TYPE=GET or DFHFC TYPE=GETAREA macro instruction, or unpredictable results will occur.

A File Work Area (FWA) is used to contain the record or segments to be written or updated. The first 16 bytes of this work area are the CICS control section followed by the actual record or segments.

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

1. Writes updated or new records on user-defined data sets
2. Acquires or locates the main storage and control blocks required to write the record
3. Releases all data set storage associated with the request to write
4. Packs a segmented record, depending on the data set organization and the operands included in the DFHFC TYPE=PUT macro instruction

Before file services can be requested in an application program via the DFHFC TYPE=PUT macro instruction, the user must have previously defined in the File Control Table (FCT) all data sets referenced by the DATASET keyword parameter and all segment sets referenced by the SEGSET keyword parameter. The application programmer must have provided instructions that do the following:

1. Symbolically define the FWA by (1) copying the appropriate storage definition (DFHFWADS) provided by CICS, or (2) providing his own storage definition for 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 the TCA at TCAFCAA. This address is provided by CICS in response to a previous DFHFC TYPE=GET or DFHFC TYPE=GETAREA request.

The application programmer must specify the parameters he requires to PUT data to a data set. He can do this in either of two ways: (1) by including the parameters in operands of the DFHFC TYPE=PUT macro instruction, or (2) by coding instructions, prior to issuing the DFHFC TYPE=PUT macro instruction, that dynamically move these parameters to fields in the TCA. If the parameters are included in operands of the DFHFC TYPE=PUT macro instruction, the applicable keywords are RDIDADR, SEGSET, and TYPOPER.

If the records being written to a data set are undefined, the application programmer must place the length of the record being written in the TCA at TCAFCURL.

A discussion of the operands that can be included in the DFHFC TYPE=PUT macro instruction follows. (The keywords used to access user-written exception handling routines are discussed in the section "Test Response to a Request for File Services".)

RDIDADR: This operand is used to specify the symbolic address of the user's data field that contains the key, as required by ISAM, or the block reference field, as required by DAM, of the record to be written. This operand can be omitted if the application programmer has previously placed the symbolic address in the TCAFCRI field of the TCA. Note that this operand must not reference a field in the FWA as the FWA might be freed before the actual write occurs.

SEGSET: The SEGSET=YES operand is used when a data set containing segmented records is to be added to or updated. If this operand is omitted, File Control does not perform its normal packing operation on segmented records.

TYPOPER: The TYPOPER=NEWREC operand must be used when adding a new record to an existing data set. If this operand is omitted, the default is TYPOPER=UPDATE in which case the DFHFC TYPE=GET,TYPOPER=UPDATE macro instruction must precede the DFHFC TYPE=PUT request.

If the application programmer desires to check the response to his request to retrieve data from a data set, he must specify the entry labels he requires to access user-written error handling routines. He can do this in any of three ways: (1) by including the entry labels in operands of the DFHFC TYPE=PUT macro instruction, (2) by coding instructions immediately following the DFHFC TYPE=PUT macro instruction that examine the response code provided by CICS at TCAFCTR (TCAFCRC if the language is ANS COBOL) and transfer control to the appropriate routine, or (3) by including the entry labels in the DFHFC TYPE=CHECK macro instruction (which usually immediately follows the DFHFC TYPE=PUT macro instruction). In any case, the applicable keywords are NORESP, DUPREC, INVREQ, IOERROR, NOSPACE, and NOTOPEN.

The following are examples of the coding required to randomly retrieve a record for updating and then return that record to the data set.

For Assembler language:

```
        COPY  DFHTCADS              COPY TCA SYMBOLIC STRG DEFN
KEY     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=KEY,                                          *
              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 UPIATED RECORD        *
              RDIDADR=KEY
```

For ANS COBOL:

```
    02  FWACBAR PICTURE S9(8).     USAGE IS COMPUTATIONAL.
        .                           NOTE DEFINE BASE REGISTER FOR FWA.
        .
        .
01  DFHTCADS COPY DFHTCADS.        NOTE COPY SYMBOLIC STRG DEFN FOR TCA.
    02  KEY PICTURE X(8).          NOTE DEFINE KEY FIELD IN TWA.
        .
        .
        .
01  DFHFWADS COPY DFHFWADS.        NOTE COPY SYMBOLIC STRG DEFN FOR FWA.
    02  RECORD PICTURE 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=KEY,                                           *
             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=KEY
```

For PL/I:

```
%INCLUDE DFHTCADS;                  /*COPY SYMBOLIC STRG DEFN FOR TCA*/
    02 KEY CHAR(8);                 /*DEFINE KEY FIELD IN TWA*/
%INCLUDE DFHFWADS;                  /*COPY SYMBOLIC STRG DEFN FOR FWA*/
    02 RECORD CHAR(350);            /*DEFINE RECORD LAYOUT IN FWA*/
```

```
          .
          .
          .
READUP:
          DFHFC TYPE=GET,              READ RECORD FOR UPDATE              *
                DATASET=MASTERB,                                          *
                RDIDADR=KEY,                                              *
                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 UPIATED RECORD            *
                RDIDADR=KEY
```

OBTAIN A FILE WORK AREA (GETAREA)

The application programmer can obtain an area of main storage to create a new record for a data set by issuing the

```
DFHFC TYPE=GETAREA,                                                       *
      DATASET=symbolic name,                                              *
      INITIMG=value,YES,                                                  *
      DSIDER=symbolic address,                                            *
      NORESP=symbolic address,                                            *
      INVREQ=symbolic address,                                            *
      NOTOPEN=symbolic address
```

macro instruction. The new main storage area is a File Work Area (FWA) and can only be obtained through a DFHFC TYPE=GETAREA request. (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 instruction:

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.

Before the DFHFC TYPE=GETAREA is used in an application program, the user must have previously defined in the File Control Table (FCT) all data sets referenced by the DATASET keyword parameter. The application programmer must have provided instructions that do the following:

1. Symbolically define the FWA by (1) copying the appropriate storage definition (DFHFWADS) provided by CICS, or (2) providing his own storage definition for the FWA.
2. 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 at FWACBAR.)

The application programmer must specify the parameters he requires to obtain a FWA. He can do this in either of two ways: (1) by including the parameters in operands of the DFHFC TYPE=GETAREA macro instruction, or (2) by coding instructions, prior to issuing the DFHFC TYPE=GETAREA macro instruction, that dynamically move these parameters to fields in the TCA. If the parameters are included in operands of the DFHFC TYPE=GETAREA macro instruction, the application keywords are DATASET and INITIMG.

If the application programmer desires to check the response to his request to obtain a FWA, he must specify the entry labels (symbolic addresses) he requires to access user-written error handling routines. He can do this in any of three ways: (1) by including the entry labels in operands of the DFHFC TYPE=GETAREA macro instruction, (2) by coding instructions immediately following the DFHFC TYPE=GETAREA macro instruction that examine the response code provided by CICS at TCAFCTR (TCAFCRC if the language is ANS COBOL) and transfer control to the appropriate routine, or (3) by including the entry labels in the DFHFC TYPE=CHECK macro instruction (which usually immediately follows the DFHFC TYPE=GETAREA macro instruction). In any case, the applicable keywords are DSIDER, NORESP, INVREQ, and NOTOPEN.

A discussion of the operands that can be included in the DFHFC TYPE=GETAREA macro instruction follows. (The keywords used to access user-written exception handling routines are discussed in the section "Test Response to a Request for File Services".)

DATASET: This operand is used to specify the symbolic name of the data set (file) to be accessed. The symbolic name must have been previously defined in the File Control Table (FCT). This operand can be omitted if the application programmer has previously placed the symbolic name in the TCAFCDI field of the TCA.

INITIMG: The INITIMG=value operand is used, at the option of the application programmer, to specify a one-byte hexadecimal initialization value for the FWA acquired by File Control. INITIMG=YES must be used if the application programmer has previously placed the initialization value in the TCASCIB field of the TCA.

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

The following are examples of the coding required to obtain a FWA, build a new record in the FWA, and then write the record to a data set.

For Assembler language:

```
        COPY  DFHTCADS             COPY TCA SYMBOLIC STRG DEFN
KEY     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,        OBTAIN A 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              *
              TYPCPER=NEWREC,                                        *
              RCILADR=KEY
```

For ANS COBOL:

```
   02 FWACBAR PICTURE S9(8) USAGE IS COMPUTATIONAL.
      .                              NOTE DEFINE BASE REGISTER FOR FWA.
      .
      .
01 DFHTCADS COPY DFHTCADS.            NOTE COPY SYMBOLIC STRG DEFN FOR TCA.
   02 KEY PICTURE X(8).              NOTE DEFINE KEY FIELD IN TWA.
      .
      .
      .
01 DFHFWADS COPY DFHFWADS.            NOTE COPY SYMBOLIC STRG DEFN FOR FWA.
   02 RECORD PICTURE 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.
        DFHFC TYPE=PUT,              WRITE THE NEW RECORD                *
              TYPOPER=NEWREC,                                           *
              RDIDADR=KEY
```

For PL/I:

```
%INCLUDE DFHTCADS;                   /*COPY SYMBOLIC STRG DEFN FOR TCA*/
   02 KEY 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,          OBTAIN A 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=KEY
```

RELEASE FILE STORAGE (RELEASE)

The application programmer can release the storage areas acquired for File Control operations by issuing the

```
       DFHFC TYPE=RELEASE,                                              *
             INVREQ=symbolic address
```

macro instruction. This macro instruction is primarily used when (1) a record has been retrieved for update, (2) it is determined that the update should not occur, and (3) it is desired to release all encumbrances associated with the update operation (that is, FWA, FIOA, exclusive control).

Note: In the case of response codes X'01' (DSIDER), X'04' (SEGIDER), X'03' (INVREQ), X'0C' (NOTOPEN), CICS does not acquire an FIOA; therefore TCAFCAA does not contain an FIOA address.

To release the storage occupied by a FWA or FIOA that was returned to a read, either a DFHFC TYPE=RELEASE or DFHSC TYPE=FREEMAIN macro instruction should be issued. However, if a "read for update" request results in an error, the FIOA is returned to the user; a DFHFC TYPE=RELEASE macro instruction should then be issued to release any exclusive control emcumbrances.

The DFHFC TYPE=RELEASE macro instruction must not be specified if the DFHFC TYPE=PUT macro instruction is used to write an updated record back to a data set.

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

1. Releases a FWA and/or FIOA.
2. Releases exclusive control of a record retrieved for update (if applicable).

Before the DFHFC TYPE=RELEASE macro instruction is executed, the application programmer must ensure that the address of the FWA to be released has been placed in the TCA at TCAFCAA. The FIOA (if any) associated with it is also released. In addition, the correct record identification must be present in the Record Identification field specified in the RDIDADR operand of the DFHFC TYPE=GET macro instruction.

The FWA and FIOA are automatically released at termination of the task, if not released earlier in response to to this macro instruction.

If the application programmer desires to check the response to his request to release a FWA or FIOA, he must specify the entry label (symbolic address) he requires to access the user-written exception handling routine. He can do this in any of three ways: (1) by including the INVREQ operand in the DFHFC TYPE=RELEASE macro instruction, (2) by coding an instruction immediately following the DFHFC TYPE=RELEASE macro instruction that examines the response code provided by CICS at TCAFCTR (TCAFCRC if the language is ANS COBOL) and transfers conrol to the appropriate routine, or (3) by including the INVREQ operand in the DFHFC TYPE=CHECK macro instruction (which usually immediately follows the DFHFC TYPE=RELEASE macro instruction). In any case, the applicable keyword is INVREQ.

For a discussion of the INVREQ keyword, see the section "Test Response to a Request for File Services".

The following are examples of the coding required to request the release of a FWA.

For Assembler language:

```
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
DFHFC TYPE=RELEASE           IN TCA AND ISSUE RELEASE REQUEST
```

For ANS COBOL:

```
   02 FWACBAR PICTURE S9(8) USAGE IS COMPUTATIONAL.
      .                                NOTE DEFINE BASE REGISTER FOR FWA.
      .
      .
      .
01 DFHTCADS COPY DFHTCADS.           NOTE COPY SYMBOLIC STRG DEFN FOR FWA.
   02 RECORD PICTURE 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.
FLSEREC.
        DFHFC TYPE=RELEASE           ISSUE RELEASE REQUEST
```

For 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*/
      .
      .
      .
TCACBAR=CSACDTA;                     /*ESTABLISH ADDRESSABILITY FOR TCA*/
TCAFCAA=FWACBAR;                     /*ADDRESS OF FWA TO BE RELEASED*/
FLSEREC:
        DFHFC TYPE=RELEASE           ISSUE RELEASE REQUEST
```

INITIATE SEQUENTIAL RETRIEVAL (SETL)

The application programmer initiates a sequential retrieval operation on a data set by issuing the

```
      DFHFC TYPE=SETL,                                              *
            DATASET=symbolic name,                                  *
            RDIDADR=symbolic address,                               *
            SEGSET=symbolic name,YES,ALL,                           *
            RETMETH=RELREC,KEY,                                     *
            NORESP=symbolic address,                                *
            DSIDER=symbolic address,                                *
            SEGIDER=symbolic address,                               *
            INVREQ=symbolic address,                                *
            NOTOPEN=symbolic address
```

macro instruction. This macro instruction is used only to initiate a sequential retrieval operation and must be issued before any GETNEXT macro instruction. It is used to establish the starting position within the data set where the browse operation is to begin.

Records are always returned to the application program in a File Work Area (FWA). The FWA returned by CICS following a SETL request is unique for the duration of that particular sequential operation. Should the application program issue another SETL request, for the same or another data set, a different FWA will be created by CICS.

Thus it is possible for a single application program to be concurrently browsing the same data set at several different locations.

Note that during a browse operation on a segmented data set, the original FWA (that is, the one allocated by the SETL request) may be replaced with a different FWA if a segment set specified in a GETNEXT request requires a larger FWA than the segment set specified in the SETL request. In this situation, the application programmer should not rely on the same FWA being returned from a GETNEXT request as was specified when the GETNEXT request was issued. The address of the appropriate FWA is always located in the TCA field labeled TCAFCAA upon return from a GETNEXT request.

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

1. Acquires the main storage I/O areas and work areas to be associated with this browse operation.
2. Preserves the segment set name (if any) as the default segment set to be used if none is specified in subsequent GETNEXT requests.
3. Returns the FWA address in the TCA field labeled TCAFCAA.

Before the SETL macro instruction can be used, the user must have previously defined in the File Control Table (FCT) the data set referenced by the DATASET operand and all segment sets referenced by the SEGSET operand. The application programmer should have also provided instructions which do the following:

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

A discussion of the operands that can be used with the DFHFC TYPE=SETL macro instruction follows. (The keywords used to specify user-written exception handling routines are discussed in the section "Test Response to a Request for File Services". These keywords include NORESP, DESIDER, SEGIDER, INVREQ, and NOTOPEN.)

DATASET: This operand is used to specify the symbolic name of the data set on which sequential retrieval is to be initiated. The symbolic name must have been previously defined in the File Control Table (FCT). This operand can be omitted if the application programmer has previously placed the symbolic data set name in the TCA field labeled TCAFCDI.

RIDADR: This operand specifies the symbolic address of the user's Record Identification field which contains the specific or generic (partial) key as required by ISAM, or the block reference as required by DAM. This operand can be omitted if the application programmer has previously placed the address of the field in the TCAFCRI field of the TCA. A generic key is one in which the user supplies only the significant characters of a desired group of keys, padding the remainder of the key field with blanks or binary zeros.

For an ISAM data set, the browse operation begins at the first record with a key equal to or higher than the key provided in the user's Record Identification field. For example, a generic key specification of "D6420000" would cause sequential processing to begin

at the first record with a key containing D642xxxx, regardless of the characters represented by the x's. (A key field of all binary zeros would therefore cause sequential processing to begin at the first logical record of the data set.)

For a DAM data set, the user's Record Identification field must contain a specific block reference (for example, TTR, MBBCCHHR, etc.) which conforms to the acceptable addressing method defined for that data set. (For further details, refer to the section "Data Base Considerations".) Processing begins with the specified block and continues with each subsequent block until the browse operation is terminated. If the DAM data set contains blocked records, processing begins at the first logical record of the first block and continues with each subsequent logical record.

The information supplied by the user in the Record Identification field is preserved by CICS for use when GETNEXT requests are issued. The Record Identification field is used by CICS during subsequent GETNEXT operations and should not be released by the application programmer. CICS places the identification of each record into this field as the record is retrieved in response to a GETNEXT request.

This feedback, placed into the Record Identification field by CICS, is always in a form which completely identifies each record. (Refer to the section "Data Base Considerations" for further information concerning the Record Identification field.) For example, assume a browse operation is to start with the first logical record of a blocked, keyed DAM data set. Before issuing the DFHFC TYPE=SETL macro instruction, the user 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 instruction, CICS places the complete logical record identification into the Record Identification field. After the first GETNEXT, the Record Identification field might contain:

```
C0C001BLOCK1REC1
```

where "C00001" represents the TTR value, "BLOCK1" represents the block key, and "REC1" represents the record key.

SEGSET: This operand is used to specify the symbolic name of the default segment set to be retrieved during a browse operation involving segmented records. This segment set is used automatically by CICS if the user fails to specify a segment set name on subsequent GETNEXT service requests. The segment set identified by a SETL macro instruction is always used as the default segment set throughout a browse operation unless altered by a RESETL macro instruction. The symbolic name must have been previously defined in the associated Segment Control section of the File Control Table (FCT).

SEGSET=YES is used if the application programmer has dynamically placed the symbolic name of the segment set in the TCA field labeled TCAFCSI prior to issuing the DFHFC TYPE=SETL macro instruction.

SEGSET=ALL is used if the application programmer wishes all segments of a record returned in an unpacked and aligned format.

If the SEGSET operand is omitted, and the data set contains segmented records, the logical record is returned in its packed format.

RETMETH: Applicable only to blocked BDAM data sets, the RETMETH operand is used to specify the format of the logical record identification that is placed in the user's Record Identification field by CICS each

time the next logical record is retrieved in a browse operation. If RETMETH=RELREC is specified, the one-byte binary relative record number is provided. If RETMETH=KEY is specified, the logical record key is provided; however, the records must have embedded keys.

For example, if a user is browsing a blocked BDAM data set (non-keyed) and the second logical record from the second physical block on the third relative track was just read in response to a GETNEXT request, the Record Identification Field would contain:

X'00020201'

upon return to the user, where "0002" represents the track, "02" represents the block, and "01" represents the logical record within the block.

The following is an example of the coding required to initiate a browse operation.

For Assembler language:

```
FWACBAR EQU   7                  ASSIGN BASE REGISTER FOR FWA
        COPY  DFHFWADS           DEFINE CONTROL SECTION OF FWA
RECORD  DS    0CL350             RECORD LAYOUT
        .
        .
        .
        CSECT
        .
        .
        .
START   DFHFC TYPE=SETL,         INITIATE BROWSE                    *
              DATASET=MASTER,                                       *
              RDIDADR=KEY,                                          *
              NOTOPEN=ERROR      CHECK FOR ERRORS
        L     FWACBAR,TCAFCAA
        .
        .
        .
KEY     DS    0CL8               INITIAL KEY DESIGNATION
        DC    CL5'JONES'         PARTIAL KEY
        DC    XL3'00'            PADDING
ERROR   DS    0H                 ENTRY TO ERROR ROUTINE
        .
        .
        .
```

For ANS COBOL:

```
   02 FWACBAR PICTURE S9(8) USAGE IS COMPUTATIONAL.
      .                           NOTE DEFINE BASE REGISTER FOR FWA.
      .
      .
01 DFHTCADS COPY DFHTCADS.        NOTE COPY SYMBOLIC STRG DEFN FOR TCA.
   02 KEY PICTURE X(8).          NOTE DEFINE KEY FIELD IN TWA.
01 DFHFWADS COPY DFHFWADS.       NOTE COPY SYMBOLIC STRG DEFN FOR FWA.
   02 RECORD PICTURE X(350).     NOTE DEFINE RECORD LAYOUT IN FWA.
      .
      .
      .
PROCEDURE DIVISION.
   MOVE CSACDTA TO TCACBAR.      NOTE ESTABLISH TCA ADDRESSABILITY.
```

```
      .
      .
      .
   MOVE 'JONES' TO KEY.
START.
        DFHFC TYPE=SETL,          INITIATE BROWSE                    *
              DATASET=MASTER,                                        *
              RIDADR=KEY,                                            *
              NOTOPEN=ERROR       CHECK FOR ERRORS
   MOVE TCAFCAA TO FWACBAR.
      .
      .
      .
ERROR.
      .
      .
      .
```

For PL/I:

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

RETRIEVE NEXT SEQUENTIAL RECORD (GETNEXT)

Once the application programmer has issued a DFHFC TYPE=SETL macro instruction to initiate a browse operation, he may request the next (or first) sequential record by issuing the

```
       DFHFC TYPE=GETNEXT,                                           *
             SEGSET=symbolic name,YES,ALL,                           *
             NORESP=symbolic address,                                *
             SEGIDER=symbolic address,                               *
             INVREQ=symbolic address,                                *
             IOERROR=symbolic address,                               *
             NOTOPEN=symbolic address,                               *
             ENDFILE=symbolic address
```

macro instruction. When the first GETNEXT request is issued following a SETL request for an ISAM data set, CICS acquires the first logical record with a key equal to or higher than the key presented by a

previous SETL; for a DAM data set, CICS acquires the first logical record specified by the user. When initiating a browse operation on a DAM data set, the user must provide a specific record reference. Each subsequent GETNEXT request, whether for an ISAM or DAM data set, causes CICS to acquire the next logical record in sequence.

Before issuing the DFHFC TYPE=GETNEXT macro instruction, the application programmer must place the address of the FWA associated with the particular operation in the TCA field labeled 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.

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

1. Retrieves the next sequential record and places it in the FWA specified by the user at TCAFCAA.
2. Places the record identification (key, block identification, etc.) of the record just retrieved into the users Record Identification field which was specified in the DFHFC TYPE=SETL request. (Refer to the discussion of Record Identification field feedback under the RDIDADR operand.) If the user wishes to issue a random "read for update" on the record just returned, he need only specify the address of the Record Identification field in his GET request.

In addition, CICS can perform the following services, depending on the operands included in the DFHFC TYPE=GETNEXT macro instruction.

1. Present the user with the segments as specified in the GETNEXT request.
2. Present the user with the segments as specified in the SETL request if no segment set is specified with the GETNEXT request.
3. If the FWA is not large enough to process a segment set specified in the GETNEXT request, dispose of the old FWA and acquire a new one large enough to process the new request.

A discussion of the operands that can be included in the DFHFC TYPE=GETNEXT macro instruction follows. (The keywords used to access user-written exception handling routines are discussed in the section "Test Response to a Request for File Services".)

SEGSET: This operand is used to specify the symbolic name of the segment set which is to be retrieved from the next sequential record. If this operand is not included in the DFHFC TYPE=GETNEXT macro instruction, CICS will use the default segment set name that may have been specified in the DFHFC TYPE=SETL macro instruction.

If this operand is omitted on a GETNEXT operation and if SEGSET was specified in the DFHFC TYPE=SETL macro instruction, the eight-character default segment identification, as specified in the SETL macro instruction, is returned at TCAFCSI upon normal completion of the GETNEXT.

SEGSET=YES is used if the application programmer has dynamically placed the segment set name in the TCA field labeled TCAFCSI prior to issuing the DFHFC TYPE=GETNEXT macro instruction.

SEGSET=ALL is specified if the application programmer wishes all segments returned in an unpacked and aligned format.

The following is an example of the coding necessary to retrieve the next sequential record in a browse operation using segmented records.

For Assembler language:

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

For ANS COBOL:

```
   02 FWACBAR PICTURE S9(8) USAGE IS COMPUTATIONAL.
      .
      .                            NOTE DEFINE BASE REGISTER FOR FWA.
      .
01 DFHTCADS COPY DFHTCADS.         NOTE COPY SYMBOLIC STRG DEFN FOR TCA.
   02 KEY PICTURE S9(18) USAGE IS COMPUTATIONAL.
                                    NOTE DEFINE KEY FIELD IN TWA.
      .
      .
      .
01 DFHFWADS COPY DFHFWADS.         NOTE COPY SYMBOLIC STRG DEFN FOR FWA.
   02 RECORD PICTURE X(350).       NOTE DEFINE RECORD LAYOUT IN FWA.
      .
      .
      .
PROCEDURE DIVISION.
   MOVE CSACDTA TO TCACBAR.        NOTE ESTABLISH TCA ADDRESSABILITY.
      .
      .
      .
   MOVE 0 TO KEY.                  NOTE START AT BEGINNING OF DATA SET.
      .
      .
      .
      DFHFC TYPE=SETL              INITIALIZE BROWSE                *
```

```
            DATASET=MASTER,                                          *
            SEGSET=A,           SET DEFAULT SEGMENT SET              *
            RIDADR=KEY
   MOVE TCAFCAA TO FWACBAR.     NOTE ESTABLISH FWA ADDRESSABILITY.
        .
        .
        .
   MOVE FWACBAR TO TCAFCAA.
        DFHFC TYPE=GETNEXT      GET NEXT SEQUENTIAL RECORD.
        .
        .
        .
   MOVE FWACBAR TO TCAFCAA.
        DFHFC TYPE=GETNEXT,     GET NEXT RECORD                      *
              SEGSET=B          WITH SEGMENT B
        .
        .
        .
```

For PL/I:

```
%INCLUDE DFHTCADS;              /*COPY SYMBCLIC STRG DEFN FOR TCA*/
   02 KEY BINARY FIXED(8,0);    /*DEFINE KEY FIELD IN TWA*/
        .
        .
        .
%INCLUDE DFHFWADS;              /*COPY SYMBOLIC STRG DEFN FOR FWA*/
   02 RECORD CHAR(350);         /*DEFINE RECORD LAYOUT IN FWA*/
        .
        .
        .
KEY=0;                          /*START AT BEGINNING OF DATA SET*/
        .
        .
        .
        DFHFC TYPE=SETL,        INITIALIZE BROWSE                    *
              DATASET=MASTER,                                        *
              SEGSET=A,         SET DEFAULT SEGMENT SET              *
              RIDADR=KEY
FWACBAR=TCAFCAA;                /*ESTABLISH FWA ADDRESSABILITY*/
        .
        .
        .
TCAFCAA=FWACBAR;
        DFHFC TYPE=GETNEXT      GET NEXT SEQUENTIAL RECORD
        .
        .
        .
TCAFCAA=FWACBAR;
        DFHFC TYPE=GETNEXT,     GET NEXT RECORD                      *
              SEGSET=B          WITH SEGMENT B
        .
        .
        .
```

TERMINATE SEQUENTIAL RETRIEVAL (ESETL)

The application programmer may terminate a browse operation by issuing the

```
       DFHFC TYPE=ESETL,                                             *
             INVREQ=symbolic address
```

macro instruction. Before the macro is issued, the programmer must ensure that the TCA field labeled TCAFCAA contains the address of the File Work Area (FWA) associated with the browse operation he wishes to terminate. In response to an ESETL request, CICS will release all I/O and work areas associated with the browse operation.

The following is an example of the coding necessary to terminate two concurrent browse operations.

For Assembler language:

```
        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
```

For ANS COBOL:

```
   02 FWACBAR PICTURE S9(8) USAGE IS COMPUTATIONAL.
        .
        .                           NOTE DEFINE BASE REGISTER FOR FWA.
        .
01 DFHTCADS COPY DFHTCADS.          NOTE COPY SYMBOLIC STRG DEFN FOR TCA.
   02 FWACELL1 PICTURE S9(8) USAGE IS COMPUTATIONAL.
   02 FWACELL2 PICTURE S9(8) USAGE IS COMPUTATIONAL.
01 DFHFWADS COPY DFHFWADS.          NOTE COPY SYMBOLIC STRG DEFN FOR FWA.
   02 RECORD PICTURE 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.
```

For PL/I:

```
%INCLUDE DFHTCADS;                  /*COPY SYMBCLIC 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 SEQUENTIAL RETRIEVAL (RESETL)

Once a browse operation has been initiated with a SETL request, the application programmer may, at any time prior to issuing the ESETL request, reset the search argument to some record other than the next sequential record. He can accomplish this by issuing the

```
      DFHFC TYPE=RESETL,                                          *
            SEGSET=symbolic name,YES,ALL,                         *
            NORESP=symbolic address,                              *
            SEGIDER=symbolic address
```

macro instruction. Prior to issuing the request, the application programmer should place the address of the appropriate FWA into the TCA field labeled TCAFCAA and place the new record identification in the Record Identification field specified through the RDIDADR operand in the original SETL request.

The use of the RESETL macro instruction 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, the TCA field labeled TCAFCAA contains the address of a new FWA which 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 the least possible overhead.

SEGSET: This operand allows the user to replace the default segment set identification specified at SETL. If this operand is omitted, the SEGSET specified in the last SETL or RESETL for this browse operation is used.

SEGSET=YES is used if the application programmer has dynamically placed the symbolic name of the segment set in the TCA field labeled TCAFCSI prior to issuing the DFHFC TYPE=RESETL macro instruction.

SEGSET=ALL is used if the application programmer wishes all segments of a record returned in an unpacked and aligned format.

The following is an example of the coding necessary to reset the search argument and the default segment set for a browse operation.

For Assembler language:

```
        COPY   DFHTCADS                 COPY TCA SYMBOLIC STRG DEFN
KEY     DS     D                        DEFINE KEY FIELD IN TWA
FWACBAR EQU    7                        ASSIGN FWA BASE REGISTER
        COPY   DFHFWADS                 COPY FWA DSECT
RECORD1 DS     0CL350                   DEFINE RECORD WITH SEGSET A
        .
        .
```

```
         .
         .
         ORG   RECORD1
RECORD2  DS    0CL250                  DEFINE RECORD WITH SEGSET B
         .
         .
         .
         CSECT
         MVC   KEY(8),=8X'00'          INITIALIZE KEY FIELD
         DFHFC TYPE=SETL,              ISSUE INITIAL SETL MACRO        *
               DATASET=MASTER,         FOR DATASET "MASTER"            *
               RDIDADR=KEY,            INITIAL SEARCH ARG=0            *
               SEGSET=A                FOR SEGSET=A
         L     FWACBAR,TCAFCAA         ESTABLISH ADDRESSABILITY TO FWA
         .
         .
         .
         ST    FWACBAR,TCAFCAA         STORE FWA ADDR IN TCA
         MVC   KEY(8),=CL8'SMITH'      ESTABLISH NEW SEARCH ARGUMENT
         DFHFC TYPE=RESETL,            ISSUE RESETL MACRO              *
               SEGSET=B                NEW SEGSET ID
         L     FWACBAR,TCAFCAA         ESTABLISH ADDRESSABILITY TO FWA
```

For ANS COBOL:

```
   02 FWACBAR PICTURE S9(8) USAGE IS COMPUTATIONAL.
         .
         .                             NOTE DEFINE BASE REGISTER FOR FWA.
         .
01 DFHTCADS COPY DFHTCADS.              NOTE COPY SYMBOLIC STRG DEFN FOR TCA.
   02 KEY PICTURE S9(18) USAGE IS COMPUTATIONAL.
                                       NOTE DEFINE KEY FIELD IN TWA.
   02 FILLER REDEFINES KEY.
      03 KEYC PICTURE X(8).
         .
         .
         .
01 DFHFWADS COPY DFHFWADS.              NOTE COPY SYMBOLIC STRG DEFN FOR FWA.
   02 RECORD1 PICTURE X(350).           NOTE DEFINE RECORD WITH SEGSET A.
         .
         .
         .
01 DFHFWA REDEFINES DFHFWADS.           NOTE CREATE STRG DEFN FOR FWA.
   02 CICSPART PICTURE X(*).            NOTE PLACE LENGTH OF FWA HERE.
   02 RECORD2 PICTURE X(250).           NOTE DEFINE RECORD WITH SEGSET B.
         .
         .
         .
   MOVE 0 TO KEY.
   DFHFC TYPE=SETL,                    ISSUE INITIAL SETL MACRO INSTR  *
               DATASET=MASTER,         FOR DATASET "MASTER"            *
               RDIDADR=KEY,            INITIAL SEARCH ARG=0            *
               SEGSET=A                FOR SEGSET=A
   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 INSTRUCTION  *
               SEGSET=B                NEW SEGSET ID
   MOVE TCAFCAA TO FWACBAR.            NOTE ESTABLISH ADDRESSABILITY TO FWA.
```

For PL/I:

```
%INCLUDE DFHTCADS;                    /*COPY SYMBOLIC STRG DEFN FOR TCA*/
   02 KEY BINARY FIXED(8,0);          /*DEFINE KEY AS BINARY*/
DECLARE 01 DFHXTCA BASED(TCACBAR),
   02 FILL CHAR(*),                   /*PLACE LENGTH OF TCA HERE*/
   02 KEYC CHAR(8);                   /*DEFINE KEY AS CHARACTER*/
      .
      .
      .
%INCLUDE DFHFWADS;                    /*COPY SYMBOLIC STRG DEFN FOR FWA*/
   02 RECORD1 CHAR(350);              /*DEFINE RECORD WITH SEGSET A*/
DECLARE 01 DFHXFWA BASED(FWACBAR),
   02 FILL CHAR(*),                   /*PLACE LENGTH OF FWA HERE*/
   02 RECORD2 CHAR(250);              /*DEFINE RECORD WITH SEGSET B*/
      .
      .
      .
KEY=0;                                /*SET KEY VALUE TO ZERO*/
        DFHFC TYPE=SETL,              ISSUE INITIAL SETL MACRO INSTR      *
              DATASET=MASTER,         FOR DATA SET "MASTER"               *
              RELADR=KEY,             INITIAL SEARCH ARG EQUALS ZERO      *
              SEGSET=A                FOR SEGSET A
FWACBAR=TCAFCAA;                      /*ESTABLISH ADDRESSABILITY FOR FWA*/
      .
      .
      .
TCAFCAA=FWACBAR;                      /*STORE FWA ADDR IN TCA*/
KEYC='SMITH';                         /*ESTABLISH NEW SEARCH ARGUMENT*/
        DFHFC TYPE=RESETL,            ISSUE RESETL MACRO INSTRUCTION      *
              SEGSET=B                NEW SEGSET ID
FWACBAR=TCAFCAA;                      /*ESTABLISH ADDRESSABILITY TO FWA*/
```

TEST RESPONSE TO A REQUEST FOR FILE SERVICES (CHECK)

One of the ways the application programmer can test the response to a request for file services is by issuing the

```
      DFHFC TYPE=CHECK,                                                  *
            NOBESP=symbolic address,                                     *
            DSIDER=symbolic address,                                     *
            SEGIDER=symbolic address,                                    *
            NOTFND=symbolic address,                                     *
            DUPREC=symbolic address,                                     *
            INVREQ=symbolic address,                                     *
            IOERROR=symbolic address,                                    *
            DUPDS=symbolic address,                                      *
            NOSPACE=symbolic address,                                    *
            NOTOPEN=symbolic address,                                    *
            ENDFILE=symbolic address
```

macro instruction, which provides for the testing of response codes and the routing of control to the appropriate user-written exception handling routines. This macro instruction provides an exception handling facility that can be used in the manner of a subroutine.

CICS automatically places the appropriate response code in the TCA at TCAFCTR (TCAFCRC if the language is ANS COBOL) after completion of the file service requested. The application programmer must specify the entry labels (symbolic addresses) he requires to access the appropriate exception handling routines previously supplied by the user.

If the application programmer does not use the DFHFC TYPE=CHECK macro instruction, he can specify the entry labels in either of two other ways: (1) by including the entry labels in operands of any other DFHFC macro instruction, or (2) by coding instructions immediately following the DFHFC macro instruction that examine the response code provided by CICS at TCAFCTR (TCAFCRC if the language is ANS COBOL) and transfer control to the appropriate routine.

The response codes are as follows:

CONDITION	ASSEMBLER	ANS COBOL	PL/I
NORESP	X'00'	12-0-1-8-9	00000000
DSIDER	X'01'	12-1-9	00000001
SEGIDER	X'04'	12-4-9	00000100
INVREQ	X'08'	12-8-9	00001000
DUPDS	X'0A'	12-2-8-9	00001010
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	X'81'	12-0-1	10000001
DUPREC	X'82'	12-0-2	10000010
NOSPACE	X'83'	12-0-3	10000011

If the DFHFC TYPE=CHECK macro instruction is used by the application programmer, it normally follows another DFHFC macro instruction. The applicable keywords are NORESP, DSIDER, SEGIDER, NOTFND, DUPREC, INVREQ, IOERROR, DUPDS, NOSPACE, NOTOPEN, and ENDFILE.

Note: When an exception condition occurs (for example, NOTFND, IOERROR, or DUPREC), the FIOA is retained; the FIOA contains the address of the FCT data set entry that produced the exception condition. The FIOA address is returned to the user at TCAFCAA. Before issuing other File Control requests, the user should free the storage occupied by the FIOA through use of the DFHFC TYPE=RELEASE macro instruction.

If the application programmer does not check for a particular response to his service request, and if that exception condition occurs, program flow proceeds to the next sequential instruction.

A discussion of the operands that can be used to test the response to a request for file services follows.

NORESP: Specifies the entry label of the user-written routine to which control is to be passed in the event no errors occur on a file operation. NORESP signifies "normal response" rather than "no response".

DSIDER: Specifies the entry label of the user-written routine to which control is to be passed if the data set specified at TCAFCDI cannot be located in the File Control Table. DSIDER signifies "data set identification error".

SEGIDER: Specifies the entry label of the user-written routine to which control is to be passed if the segment set specified at TCAFCSI cannot be located in the File Control Table. SEGIDER signifies "segment set identification error".

NOTFND: Specifies the entry label of the user-written routine to which control is to be passed in the event of an unsuccessful retrieval of a record based on the search argument provided (key or block reference). NOTFND signifies a "record not found" situation.

DUPREC: Specifies the entry label of the user-written routine to which control is to be passed in the event an attempt is made to add a record to the data set in which one already exists with that key. DUPREC signifies "duplicate record".

INVREQ: Specifies the entry label of the user-written routine to which control is to be passed in the event a file operation is attempted that is not provided for (or allowed) according to the data set entry specifications in the FCT. INVREQ signifies "invalid request". The address of the appropriate File Control Table entry is returned at TCAFCAA.

IOERROR: Specifies the entry label of the user-written routine to which control is to be passed in the event an unusual event occurs during a file operation. When an I/O event error code is not covered by one of the CICS error classes (for example, NOSPACE, NOTFND), it is considered an I/O error. The user's routine may check the actual error codes in the FIOA (FCFIOBEX in the case of DAM or BDAM, FCFIOEX in the case of ISAM), the address of which is returned in the TCA field labeled TCAFCAA. Since these error codes are access method and operating system dependent, the user should be aware that checking these codes in his application programs would have a limiting effect on migrating those application programs from CICS/DOS to CICS/OS, if this were ever desired.

DUPDS: Specifies the entry label of the user-written routine to which control is to be passed in the event the record just retrieved on an indirect access is from the duplicate data set rather than from the prime (master) data set. The duplicate record is processed by the user-written routine rather than allowing the record to be processed by main line code as a prime data set record. DUPDS signifies "duplicate data set".

NOSPACE: Specifies the entry label of the user-written routine to which control is to be passed in the event no direct access space is available for adding records to a data set. When this condition occurs, the original user record is returned in a File Work Area (FWA) the address of which is at TCAFCAA. The main storage location of this FWA may be different from that for the FWA acquired in response to the DFHFC TYPE=PUT macro instruction (which was issued to add the record). This error code is not applicable when adding records to DAM non-keyed data sets.

NOTOPEN: Specifies the entry label of the user-written routine to which control is to be passed in the event the requested data set is not open. This error condition can occur after any file service request except RELEASE, ESETL, and RESETL because data base data sets can be dynamically closed at any time without regard to outstanding activity on the data set.

ENDFILE: Specifies the entry label of the user-written routine to which control is to be passed in the event an end-of-file condition

is detected during the sequential retrieval (browse) of records from a data set. This condition occurs only after a GETNEXT request.

The following are examples of the coding required to examine the response code provided by CICS at TCAFCTR (TCAFCRC if the language is ANS COBOL) and transfer control to the appropriate user-written error handling routine.

For Assembler language:

```
        DFHFC TYPE=GET,                                          *
              DATASET=MASTER,                                    *
              RDIDADR=KEY
        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
```

For ANS COBOL:

```
        DFHFC TYPE=GET,                                          *
              DATASET=MASTER,                                    *
              RDIDADR=KEY
        IF TCAFCRC=' ' THEN GO TO GOOD.
        IF TCAFCRC=' ' THEN GO TO ERROR.
        IF TCAFCRC=' ' THEN GO TO ERROR.
        .
        .
        .
GOOD.
        .
        .
        .
ERROR.
        DFHPC TYPE=ABEND
```

where the value specified within single quotes is a multipunch code for the required hexadecimal value. For example, a hexadecimal 00 has a multipunch code of 12-0-1-8-9.

For PL/I:

```
        DFHFC TYPE=GET,                                          *
              DATASET=MASTER,                                    *
              RDIDADR=KEY
        IF TCAFCTR='00C00000'B THEN GO TO GOOD;
        IF TCAFCTR='1CCC0000'B THEN GO TO ERROR;
        IF TCAFCTR='00001000'B THEN GO TO ERROR;
        .
        .
        .
```

```
GOOD:
        .
        .
        .
ERROR:
        DFHPC TYPE=ABEND
```

TRANSIENT DATA SERVICES

Transient Data Management provides, through Transient Data Control, a generalized queuing facility where 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.

Intrapartition destinations are queues of data on direct access devices develcped for input to cne or more programs running asynchronously (ccncurrently) as separate tasks; they are internal to the CICS partition/region. Data directed to or from these internal destinations is called intrapartition data and may consist only of variable-length records. Intrapartition destinations can be associated with (1) a terminal (to accomplish message switching or to route data to a terminal other than the originating terminal), (2) an output data set, or (3) an application program under the control of CICS.

The intrapartition queue is reusable. An option permits the user to indicate, by symbolic destinaticn, whether Transient Data space management is tc ccntrol the reuse of tracks associated with a particular destination identification (DESTID), or whether the releasing of track space is to be contrclled through the Transient Data PURGE macro facility. Note that if Transient Data space management is not used, intrapartition queues continue to grow, irrespective of whether the data has been read, until the user purges them.

Extrapartition destinaticns are queues (data sets) that are external to the CICS partition/region, residing on tape or direct access devices. Data directed to or from these external destinations is called extrapartition data and may consist of sequential records that are fixed or variable length, blocked or unblocked. The record format specification is described in the Destination Control Table in the System Programmer's Reference Manual.

Intrapartition and extrapartition destinations can be used as indirect destinaticns which are symbolic references to still other destinations. This facility provides some flexibility in program maintenance in that an installation can be changed, giving a destination a new symbolic name, withcut recompiling existing programs. These programs can be allowed to route data to the previously existing symbolic name; however, the previously existing symbolic name is now an indirect destination that refers to the new symbolic name.

Requests fcr transient data services are communicated to Transient Data Control via CICS macro instructions. Transient Data Control then executes as a service program, at the priority of the requesting program, under control of the requesting program's TCA, saving and restoring registers from this TCA. After the requested transient data service has been provided (or attempted), control is returned to the next executable instruction in the requesting program. Upon return tc the requesting program, tests can be made and control routed to various user-written error handling routines based on the outcome of the requested transient data service.

The Transient Data Management macro instruction (DFHTD) is used to request any of the following services:

1. Acquire data from a predefined symbolic source which references a data set, a program, or a terminal.
2. Direct data to a predefined symbolic destination which references a data set, a program, or a terminal.
3. Control the processing of extrapartition data sets.
4. Check the response to a request for transient data services.

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 symbolic destination CSTL (terminal log) and/or to symbolic destination CSMT (master terminal).

Destinations must have been previously established in the Destination Control Table (DCT) for all user and CICS destinations. Lack of a destination definition results in the loss of data sent to these destinations.

For intrapartition destinations, CICS provides the option of automatic task initiation. Automatic task initiation is accomplished by setting a nonzero trigger level for a particular destination. When the number of entries (PUT's from one or more programs) in the queue (destination) reaches a specified level, the transaction is automatically initiated and a program given control to process the data in that queue. The program that has been automatically initiated must issue repetitive GET's to deplete the queue.

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

If an automatically initiated task does not deplete the queue, access to the queue is not prevented. If the task is normally or abnormally terminated before the queue is emptied, and if the destination is a terminal, the same task is reinitiated regardless of the trigger level. However, if the destination is a data set (file), 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.

The following operands can be included in the DFHTD macro instruction:

```
DFHTD TYPE=PUT,                                          *
      DESTID=symbolic name,                              *
      IDADDR=symbolic address,                           *
      NORESP=symbolic address,                           *
      IDERROR=symbolic address,                          *
      IOERROR=symbolic address,                          *
      NOTOPEN=symbolic address,                          *
      NOSPACE=symbolic address
```

```
DFHTD TYPE=GET,                                                *
      DESTID=symbolic name,                                    *
      NORESP=symbolic address,                                 *
      QUEZERO=symbolic address,                                *
      IDERROR=symbolic address,                                *
      IOERROR=symbolic address,                                *
      NOTOPEN=symbolic address

DFHTD TYPE=FEOV,                                               *
      DESTID=symbolic name,                                    *
      NORESP=symbolic address,                                 *
      IDERROR=symbolic address,                                *
      NOTOPEN=symbolic address

DFHTD TYPE=PURGE,                                              *
      DESTID=symbolic name,                                    *
      IDERROR=symbolic address,                                *
      NORESP=symbolic address

DFHTD TYPE=CHECK,                                              *
      NORESP=symbolic address,                                 *
      QUEZERO=symbolic address,                                *
      IDERROR=symbolic address,                                *
      IOERROR=symbolic address,                                *
      NOTOPEN=symbolic address,                                *
      NOSPACE=symbolic address
```

DISPOSE OF DATA (PUT)

The application programmer can direct transient data to a predefined symbolic destination by issuing the

```
DFHTD TYPE=PUT,                                                *
      DESTID=symbolic name,                                    *
      TDADDR=symbolic address,                                 *
      NORESP=symbolic address,                                 *
      IDERROR=symbolic address,                                *
      IOERROR=symbolic address,                                *
      NOTOPEN=symbolic address,                                *
      NOSPACE=symbolic address
```

macro instruction. 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 the data is intrapartition, a copy of the TDOA symbolic storage definition (DFHTDOA) should be included and all references to the output area should be made via a register (TDOABAR) which points to the beginning of the area.

If the data is variable length, whether intrapartition or extrapartition, the first four bytes of the data (LLbb) contain the data length, 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 recommended to be a two-byte field of binary zeros.

The application programmer must specify the parameters he requires to dispose of transient data. He can do this in either of two ways: (1) by including the parameters in operands of the DFHTD TYPE=PUT macro instruction, or (2) by coding instructions, prior to issuing the DFHTD TYPE=PUT macro instruction, that dynamically move these parameters to fields in the TCA. If the parameters are included in operands of the DFHTD TYPE=PUT macro instruction, the applicable keywords are DESTID and TDADDR.

A discussion of the operands that can be included in the DFHTD TYPE=PUT macro instruction follows. (The keywords used to access user-written exception handling routines are discussed in the section "Test Response to a Request for Transient Data Services".)

DESTID: Specifies the symbolic destination name (which is the name of an entry in the DCT) to which the data is to be routed and queued. The destination name can be coded in the macro instruction or dynamically loaded in the TCA at location TCATDDI.

TDADDR: Specifies the address of the data to be written. This can be provided by coding the symbolic name of the area in the macro instruction or by dynamically loading the address of the area in the TCA at location TCATDAA. Transient Data Control does not release this area after the output of the data. The address points to the first four bytes of the output area which, for variable length records and intrapartition data, must contain the length of the record. This length includes both the length of the data and the length field (of the form LLbb).

The following are examples of the coding required to write data to a predefined symbolic destination.

For Assembler language:

```
TDOABAR     EQU     7
            COPY    DFHTDOA
DATA        DS      CL10
            .
            .
            .
            MVC     TDOAVRL,LENGTH
            MVC     DATA,MESSAGE
            MVC     TCATDDI,=C'CSML'
            DFHTD TYPE=PUT,                                    *
                  TDADDR=TDOAVRL
            .
            .
            .
```

For ANS COBOL:

```
            .
            .
            .
       02   TDOABAR PICTURE S9(8) USAGE IS COMPUTATIONAL.
            .
            .
            .
            MOVE LENGTH TO TDOAVRL.
            MOVE MESSAGE TO DATA.
            MOVE 'CSML' TO TCATDDI.
            DFHTD TYPE=PUT,                                    *
                  TDADDR=TDOAVRL
            .
            .
            .
```

For PL/I:

```
%INCLUDE DFHTDOA;
    2  DATA CHAR(10);
          .
          .
          .
TDOAVRL=LENGTH;
DATA=MESSAGE;
TCATDDI='CSML';
       DFHTD TYPE=PUT,                                          *
             TDADDR=TDOAVRL
          .
          .
          .
```

ACQUIRE QUEUED DATA (GET)

The application programmer can acquire transient data from a predefined symbolic source by coding the

```
       DFHTD TYPE=GET,                                          *
             DESTID=symbolic name,                              *
             NORESP=symbolic address,                           *
             QUEZERO=symbolic address,                          *
             IOERROR=symbolic address,                          *
             NOTOPEN=symbolic address
```

macro instruction. The address of the data acquired is returned in the TCA at TCATDAA.

If the data is extrapartition, the address points to the first word of the data area. For variable-length records, the first four bytes of the data contain the length (LLbb) as specified for variable-length data sets.

If the data is intrapartition, the address of the data acquired points to a CICS input area defined by DFHTDIA. The field TDIAIRL contains the length (data length plus the length of the length field). In the case of either intrapartition or extrapartition data, the data must be moved to be used in any other input/output operation.

If the user issues a subsequent DFHTD TYPE=GET macro, the Transient Data I/O area from the previous GET will be reused, thus data to be saved should be moved to a user area.

Note: The application programmer should not attempt to free storage acquired by the Transient Data Control program in response to a DFHTD TYPE=GET macro instruction. 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 application programmer must specify the parameters he requires to acquire transient data. He can do this in either of two ways: (1) by including the parameters in operands of the DFHTD TYPE=GET macro instruction, or (2) by coding instructions, prior to issuing the DFHTD TYPE=GET macro instruction, that dynamically move these parameters to fields in the TCA. If the parameters are included in operands of the DFHTD TYPE=GET macro instruction, the applicable keyword is DESTID.

A discussion of the DESTID operand follows. (The keywords used to access user-written exception handling routines are discussed in the section "Test Response to a Request for Transient Data Services".)

DESTID: Specifies the symbolic destination name (the name of an entry in the DCT) from which data is to be retrieved. The name can be specified in the macro instruction or by dynamically loading it in the TCA at location TCATDDI.

The following are examples of the coding required to read a record from an intrapartition data set.

For Assembler language:

```
TDIABAR     EQU     7
            COPY    DFHTDIA
            .
            .
            .
            MVC     TCATDDI,=C'CSML'
            DFHTD   TYPE=GET
            L       TDIABAR,TCATDAA
```

For ANS COBOL:

```
            02  TDIABAR PICTURE S9(8) USAGE IS COMPUTATIONAL.
            .
            .
            .
        01  DFHTDIA COPY DFHTDIA.
            .
            .
            .
        MOVE 'CSML' TO TCATDDI.
            DFHTD TYPE=GET
        MOVE TCATDAA TO TDIABAR.
            .
            .
            .
```

For PL/I:

```
        %INCLUDE DFHTDIA;
            2   DUMMY CHAR(1);
            .
            .
            .
        TCATDDI='CSML';
        DFHTD TYPE=GET
        TDIABAR=TCATDAA;
            .
            .
            .
```

In the above examples, if the record is to be read from an extrapartition data set, the address passed to the user at TCATDAA is the address of the actual data. However, since the DFHTDIA symbolic storage definition is being used, the address must be adjusted to point to the CICS control area preceding the actual data. Therefore, immediately following the instruction that moves the contents of TCATDAA

to TDIABAR, another instruction must be added. The following examples which apply to CICS/OS are applicable TO CICS/DOS if '40' is replaced by '8'.

The examples are for variable-length records where the first byte of the data is actually the LLbb field. Therefore, if the retrieved record is fixed format, the value in the examples must be 12 and 44.

Note: This DSECT is intended to be used for intrapartition data. The values are subject to change in future versions of CICS. No DSECTs are provided for extrapartition data.

For Assembler language:

```
SH TDIABAR,=H'40'
```

For ANS COBOL:

```
SUBTRACT 40 FROM TDIABAR.
```

For PL/I:

```
DCL TDIABAA FIXED BIN(30) BASED(TDIABAB);
TDIABAB=ADDR(TDIABAR);
TDIABAA=TDIABAA - 40;
```

If the extrapartition data set is blocked, alignment requirements are the user's responsibility. The DFHTDIA DSECT assumes doubleword alignment for the start of the LLbb field in variable records, or for the start of the data if fixed records are processed.

CONTROL THE PROCESSING OF EXTRAPARTITION DATA SETS (FEOV)

The application programmer can create a "forced end of volume" situation on an extrapartition magnetic tape data set (file) by issuing the

```
DFHTD TYPE=FEOV,                                              *
      DESTID=symbolic name,                                   *
      NORESP=symbolic address,                                *
      IDERROR=symbolic address,                               *
      NOTOPEN=symbolic address
```

macro instruction. This macro instruction is used to cause the rewinding and unloading of a magnetic tape reel; the next tape reel must then be reloaded.

DESTID: Specifies the symbolic address of the destination against which "forced end of volume" is to be applied.

Note: This facility must be used with caution since CICS operation is halted until the new tape reel has been reloaded.

For a discussion of the NORESP, IDERROR, and NOTOPEN operands, see the section "Test Response to a Request for Transient Data Services."

The following are examples of the coding required to create a "forced end of volume" situation on an extrapartition magnetic tape data set.

For Assembler language:

```
         .
         .
         .
         MVC    TCATDDI,=C'CSML'
         DFHTD TYPE=FEOV
         .
         .
         .
```

For ANS COBOL:

```
        .
        .
        .
        MOVE 'CSML' TO TCATDDI.
        DFHTD TYPE=FEOV
        .
        .
        .
```

For PL/I:

```
        .
        .
        .
        TCATDDI='CSML';
        DFHTD TYPE=FEOV
        .
        .
        .
```

PURGE TRANSIENT DATA (PURGE)

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 the

```
DFHTD TYPE=PURGE,                                              *
      DESTID=symbolic name,                                    *
      IDERROR=symbolic address,                                *
      NORESP=symbolic address
```

macro instruction, which causes all storage associated with the destination to be freed (erased and deallocated).

For destinations designated as non-reusable in the Destination Control Table, the DFHTD TYPE=PURGE macro instruction must be used to free storage associated with the destination. Otherwise, the storage remains allocated to the destination, and the data associated with the destination continues to grow until the allocated storage is entirely used or until the storage is freed via this macro instruction.

A discussion of the DESTID operand follows. For a discussion of the IDERROR operand, see the section "Test Response to a Request for Transient Data Services".

DESTID: Specifies the symbolic destination name (the name of an entry in the Destination Control Table) associated with the transient data to be purged. The destination name can be coded in the macro instruction or dynamically loaded in the TCA at location TCATDDI.

TEST RESPONSE TO A REQUEST FOR TRANSIENT DATA SERVICES (CHECK)

One of the ways the application programmer can test the response to a request for transient data services is by issuing the

```
DFHTD TYPE=CHECK,                                              *
      NORESP=symbolic address,                                 *
      QUEZERO=symbolic address,                                *
      IDERROR=symbolic address,                                *
```

```
IOERROR=symbolic address,                                    *
NOTOPEN=symbolic address,                                    *
NOSPACE=symbolic address
```

macro instruction, which provides for the testing of response codes and the routing of control to the appropriate user-written exception handling routines. This macro instruction provides an exception handling facility that can be used in the manner of a subroutine.

CICS automatically places the appropriate response code in the TCA at TCATDTR (TCATDRC if the language is ANS COBOL) after completion of the transient data service requested. The application programmer must specify the entry labels (symbolic addresses) he requires to access the appropriate exception handling routine previously supplied by the user.

If the application programmer does not use the DFHTD TYPE=CHECK macro instruction, he can specify the entry labels in either of two other ways: (1) by including the entry labels in operands of any other DFHTD macro instruction, or (2) by coding instructions immediately following the DFHTD macro instruction that examine the response code provided by CICS at TCATDTR (TCATDRC if the language is ANS COBOL) and transfer control to the appropriate routine.

The response codes are as follows:

CONDITION	ASSEMBLER	ANS COBOL	PL/I
NORESP	X'00'	12-0-1-8-9	00000000
QUEZERO	X'01'	12-1-9	00000001
IDERROR	X'02'	12-2-9	00000010
IOERROR	X'04'	12-4-9	00000100
NOTOPEN	X'08'	12-8-9	00001000
NOSPACE	X'10'	12-11-1-8-9	00010000

If the DFHTD TYPE=CHECK macro instruction is used by the application programmer, it should usually immediately follow another DFHTD macro instruction. The applicable keywords are NORESP, QUEZERO, IDERROR, IOERROR, NOTOPEN, and NOSPACE.

If the application programmer does not check for a particular response to his service request, and if that exception condition occurs, program flow proceeds to the next sequential instruction.

The operands that can be used to test the response to a request for transient data services are as follows.

NORESP: Specifies the entry label of the user-written routine to which control is to be passed in the event no errors occur during a data set (file) operation. NORESP signifies "normal response" rather than "no response".

QUEZERO: Specifies the entry label of the user-written routine to which control is to be passed when the destination (queue) accessed by a GET is found to be empty. This response applies to both intrapartition and extrapartition input queues.

IDERROR: Specifies the entry label of the user-written routine to which control is to be passed in the event the symbolic destination identification referenced by a GET, PUT, or FEOV cannot be found.

ICERROR: Specifies the entry label of the user-written routine to which control is to be passed in the event an input/output error occurs on a data record and the data record in error is skipped. Transient Data returns an ICERROR indication as long as the queue (destination) can be read, after which a QUEZERO response is returned; queue processing may then be restarted.

NOTOPEN: Specifies the entry label of the user-written routine to which control is to be passed in the event a destination is closed.

NOSPACE: Specifies the entry label of the user-written routine to which control is to be passed when it is determined that no more space exists on a particular intrapartition queue or that the write request cannot be serviced. If the NOSPACE response is received, more data should not be written to this queue as it could be lost.

The following are examples of the coding required to examine the response code provided by CICS at TCATDTR (TCATDRC if the language is ANS COBOL) and transfer control to the appropriate user-written exception handling routine.

For Assembler language:

```
        DFHTD TYPE=GET,                                          *
              DESTID=CSML
        CLI   TCATDTR,X'00'
        BE    GOOD
        DFHPC TYPE=ABEND
GOOD    DS    0H
        .
        .
        .
```

For ANS COBOL:

```
        DFHTD TYPE=GET,                                          *
              DESTID=CSML
        IF TCATDRC=' ' THEN GO TO GOOD.
        DFHPC TYPE=ABEND
GOOD.
        .
        .
        .
```

where the value specified within single quotes is a multipunch code for the required hexadecimal value. For example, a hexadecimal 00 has a multipunch code of 12-0-1-8-9.

For PL/I:

```
        DFHTD TYPE=GET,                                          *
              DESTID=CSML
        IF TCATDTR='CCCC0000'B THEN GO TO GOOD;
        DFHPC TYPE=ABEND
GOOD:
        .
        .
        .
```

TEMPORARY STORAGE SERVICES

Temporary Storage Management provides the facility, through Temporary Storage Contrcl, that enables user-written applicatiocn programs to store temporary data in main storage or on auxiliary storage (DASD). Temporary data is stored, retrieved, and released using a unique symbolic name (up to eight characters) assigned to the data by the originating task.

Data stored in temporary storage can remain intact beyond the time that the originating task is terminated. That is, the originating task may be terminated and its transaction storage released; however, the data stored in temporary storage is still available under the symbolic name with which it was stored. The data remains intact until it is released by the user (the originating task or any other task). Temporary data can be accessed any number of times until it is released.

When temporary data is released, the space it occupies becomes reusable. If the data was stored in main storage, the area is released and returned to the available dynamic area. If the data was stored cn auxiliary storage, the physical block becomes available and can be reused for other data.

Temporary data can be retrieved by the originating task or any other task using the unique user-supplied name. To avoid conflicts due to duplicate names, a naming convention must be devised by the user; for example, by appending the operator identificatiocn, terminal identification, or transaction identification as a prefix or suffix to the user-supplied symbolic name.

When retrieving data, Temporary Storage Control always searches for the data in main storage before it searches in auxiliary storage.

Except in the CICS/DOS-ENTRY system, main storage is used by Temporary Storage Control to store small amounts of data (up to 256 bytes) for short periods of time. For example, 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-reentrant (serially reusable between entry and exit points of the program). If a request is made to store more than 256 bytes of data in main storage, the request automatically defaults to auxiliary storage.

Auxiliary storage is used by Temporary Storage Control to contain data greater than 256 bytes in length and/or data that is to be kept for extended periods of time. Auxiliary temporary storage can also be used when reusable storage space is required.

Possible uses of auxiliary storage by Temporary Storage Control include:

1. Video paging. A task could retrieve a large master record from DASD, format it into several screen images, store the screen images on Temporary Storage auxiliary storage, and then ask the terminal operator which "page" (screen image) is desired. The user 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, etc.
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 the coded input, the data collection task could output its "incomplete" data to Temporary Storage and terminate the task. The terminal is now free to enter a completely 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 which will be used for output on a preprinted form.

The Temporary Storage Management macro instruction (DFHTS) is used to request any of the following services:

1. Acquire data from a symbolic source in main or auxiliary storage.
2. Send data to symbolic storage in main or auxiliary storage.
3. Release data from main or auxiliary storage.
4. Check the response to a request for temporary storage services.

The following operands can be included in the DFHTS macro instruction:

```
DFHTS TYPE=PUT,                                          *
      DATAID=name,                                       *
      TSDADDR=symbolic address,                          *
      STORFAC=AUXILIARY,MAIN,                            *
      NORESP=symbolic address,                           *
      INVREQ=symbolic address

DFHTS TYPE=GET,                                          *
      DATAID=name,                                       *
      TSDADDR=symbolic address,YES,                      *
      RELEASE=YES,NO,                                    *
      NORESP=symbolic address,                           *
      IDERROR=symbolic address,                          *
      IOERROR=symbolic address

DFHTS TYPE=RELEASE,                                      *
      DATAID=name,                                       *
      NORESP=symbolic address,                           *
      IDERROR=symbolic address

DFHTS TYPE=CHECK,                                        *
      NORESP=symbolic address,                           *
      IDERROR=symbolic address,                          *
      IOERROR=symbolic address,                          *
      INVREQ=symbolic address
```

STORE TEMPORARY DATA (PUT)

The application programmer can send temporary data to a symbolic source in main or auxiliary storage by issuing the

```
DFHTS TYPE=PUT,                                          *
      DATAID=name,                                       *
      TSDADDR=symbolic address,                          *
      STORFAC=AUXILIARY,MAIN,                            *
      NORESP=symbolic address,                           *
      INVREQ=symbolic address
```

macro instruction. The temporary data must have the standard variable-length format, with the data length specified in the first four bytes (LLbb) followed by the data. 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 recommended to be a two-byte field of binary zeros.

The application programmer must specify the parameters he requires to store temporary data. He can do this in either of two ways: (1) by including the parameters in operands of the DFHTS TYPE=PUT macro instruction, or (2) by coding instructions, prior to issuing the DFHTS TYPE=PUT macro instruction, that dynamically move these parameters to fields in the TCA. If the parameters are included in operands of the DFHTS TYPE=PUT macro instruction, the applicable keywords are DATAID, TSDADDR, and STORFAC.

A discussion of the operands that can be included in the DFHTS TYPE=PUT macro instruction follows. (The keywords used to access user-written exception handling routines are discussed in the section "Test Response to a Request for Temporary Storage Services".)

DATAID: Specifies the unique alphameric name (one to eight characters) to be assigned to the temporary data to be stored. This operand can be omitted if the application programmer has previously placed the unique alphameric name in the TCATSDI field of the TCA.

TSDADDR: Specifies the symbolic address (halfword aligned) of the temporary data to be stored. This operand can be omitted if the application programmer has previously placed the address in the TCATSDA field of the TCA. The data area is not released by the temporary data PUT.

STORFAC: Specifies whether the temporary data is to be stored on auxiliary storage (AUXILIARY) or in main storage (MAIN). The default is STORFAC=AUXILIARY. Any data greater than 256 bytes in length is automatically placed on auxiliary storage regardless of the user's request.

The following are examples of the coding required to write a record to temporary storage.

For Assembler language:

```
TSIOABAR   EQU   7
           COPY  DFHTSIOA
DATA       DS    CL10
           .
           .
           .
           MVC   TSIOAVRL,LENGTH
           MVC   DATA,MESSAGE
           DFHTS TYPE=PUT,                                        *
                 DATAID=UNIQNME,                                  *
                 TSDADDR=TSIOAVRL
           .
           .
           .
```

For ANS COBOL:

```
           02 TSIOABAR PICTURE S9(8) USAGE IS COMPUTATIONAL.
       01  DFHTSIOA COPY DFHTSIOA.
           02 DATA PICTURE X(10).
              .
              .
              .
           MOVE LENGTH TO TSIOAVRL.
           MOVE MESSAGE TO DATA.
       DFHTS TYPE=PUT,                                              *
             DATAID=UNIQNME,                                        *
             TSDADDR=TSIOAVRL
              .
              .
              .
```

For PL/I:

```
       %INCLUDE DFHTSIOA;
           2 DATA CHAR(10);
              .
              .
              .
       TSIOAVRL=LENGTH;
       DATA=MESSAGE;
       DFHTS TYPE=PUT,                                              *
             DATAID=UNIQNME,                                        *
             TSDADDR=TSIOAVRL
              .
              .
```

RETRIEVE TEMPORARY DATA (GET)

The application programmer can retrieve temporary data from a symbclic source in main or auxiliary storage by issuing the

```
       DFHTS TYPE=GET,                                              *
             DATAID=name,                                           *
             TSDADDR=symbclic address,YES,                          *
             RELEASE=YES,NO,                                        *
             NORESP=symbolic address,                               *
             IDERROR=symbclic address,                              *
             IOERROR=symbclic address
```

macro instruction. Data retrieved from temporary storage is placed in either a user-provided storage area or an area (transaction storage) acquired for the user by Temporary Storage Control.

The application programmer must specify the parameters he requires to retrieve temporary data. He can do this in either of two ways: (1) by including the parameters in operands of the DFHTS TYPE=GET macro instruction, or (2) by coding instructions, prior to issuing the DFHTS TYPE=GET macro instruction, that dynamically move these parameters to fields in the TCA. If the parameters are included in operands of the DFHTS TYPE=GET macro instruction, the applicable keywords are DATAID, TSDADDR, and RELEASE.

A discussion of the operands that can be included in the DFHTS TYPE=GET macro instruction follows. (The keywords used to access user-written exception handling routines are discussed in the section "Test Response to a Request for Temporary Storage Services".)

DATAID: Specifies the name assigned to the temporary data at the time it was placed in temporary storage. This operand can be omitted if the application programmer has previously placed the name in the TCATSDI field of the TCA.

TSDADDR: Specifies the symbolic name of the user-provided storage area into which the temporary data is to be read (or moved). TSDADDR=YES must be coded if the application programmer has previously placed this symbolic address in the TCA at TCATSDA. If this operand is omitted, Temporary Storage Control obtains a storage area, moves or reads temporary data into the area, and returns the address of the area to the user in the TCA at TCATSDA.

RELEASE: Specifies whether the data is to be released following this acquisition. The default is RELEASE=NO.

The following are examples of the coding required to read a record from temporary storage. In these examples, the data is moved to the area defined by the user in the TSDADDR operand. If the TSDADDR operand is omitted, the data is moved into a storage area obtained by Temporary Storage Control, and the address of the storage area is returned to the user at TCATSDA.

For Assembler language:

```
ISIOABAR   EQU     7
           COPY    DFHTSIOA
           .
           .
           .
       DFHTS TYPE=GET,                                          *
             DATAID=UNIQNME,                                    *
             TSDADDR=TSIOAVRL
           .
           .
           .
```

For ANS COBOL:

```
           02  TSIOABAR PICTURE S9(8) USAGE IS COMPUTATIONAL.
           .
           .
           .
       01  DFHTSIOA COPY DFHTSIOA.
           .
           .
           .
       DFHTS TYPE=GET,                                          *
             DATAID=UNIQNME,                                    *
             TSDADDR=TSIOAVRL
           .
           .
           .
```

For PL/I:

```
%INCLUDE DFHTSIOA;
    2 DATA CHAR(10);
    .
    .
    .
DFHTS TYPE=GET,                                          *
      DATAID=UNIQNME,                                    *
      TSDADDR=TSIOAVRL
    .
    .
    .
```

RELEASE TEMPORARY DATA (RELEASE)

The application programmer can release temporary data from main or auxiliary storage by issuing the

```
DFHTS TYPE=RELEASE,                                      *
      DATAID=name,                                       *
      NORESP=symbolic address,                           *
      IDERROR=symbolic address
```

macro instruction. If the data was stored in main storage, the area is freed and returned to the available dynamic area. If the data was stored in auxiliary storage, the space is made available for other data.

Temporary data should be released at the earliest possible time to avoid suspended tasks.

A discussion of the DATAID=name operand of the DFHTS TYPE=RELEASE macro instruction follows. (The keywords used to access user-written exception handling routines are discussed in the section "Test Response to a Request for Temporary Storage Services".)

DATAID: Specifies the name assigned to the data to be released from temporary storage. This operand can be omitted if the application programmer has previously placed the name in the TCATSDI field of the TCA.

The following are examples of the coding required to release a record from temporary storage.

For Assembler language:

```
MVC   TCATSDI,=C'UNIQNME'
DFHTS TYPE=RELEASE
```

For ANS COBOL:

```
MOVE 'UNIQNME' TO TCATSDI.
DFHTS TYPE=RELEASE
```

For PL/I:

```
TCATSDI='UNIQNME';
DFHTS TYPE=RELEASE
```

TEST RESPONSE TO A REQUEST FOR TEMPORARY STORAGE SERVICES (CHECK)

One of the ways the application programmer can test the response to a request for temporary storage services is by issuing the

```
DFHTS TYPE=CHECK,                                              *
      NORESP=symbolic address,                                 *
      IDERROR=symbolic address,                                *
      IOERROR=symbolic address,                                *
      INVREQ=symbolic address
```

macro instruction, which provides for the testing of response codes and the routing of control to the appropriate user-written exception handling routines. This macro instruction provides an exception handling facility that can be used in the manner of a subroutine.

CICS automatically places the appropriate response code in the TCA at TCATSTR (TCATSRC if the language is ANS COBOL) after completion of the temporary storage service requested. The application programmer must specify the entry labels (symbolic addresses) he requires to access the appropriate exception handling routine previously supplied by the user.

The response codes are as follows:

CONDITION	ASSEMBLER	ANS COBOL	PL/I
NORESP	X'00'	12-0-1-8-9	00000000
IDERROR	X'02'	12-2-9	00000010
IOERROR	X'04'	12-4-9	00000100
INVREQ	X'20'	11-0-1-8-9	00100000

If the application programmer does not use the DFHTS TYPE=CHECK macro instruction, he can specify the entry labels (symbolic addresses) in either of two other ways: (1) by including the entry labels in operands of any other DFHTS macro instruction, or (2) by coding instructions immediately following the DFHTS macro instruction that examine the response code provided by CICS at TCATSTR (TCATSRC if the language is ANS COBOL) and transfer control to the appropriate routine.

If the DFHTS TYPE=CHECK macro instruction is used by the application programmer, it should usually immediately follow another DFHTS macro instruction. The applicable keywords are NORESP, IDERROR, IOERROR, and INVREQ.

If the application programmer does not check for a particular response to his service request, and if that exception condition occurs, program flow proceeds to the next sequential instruction.

The operands that can be used to test the response to a request for temporary storage services are as follows.

NORESP: Specifies the entry label of the user-written routine to which control is to be passed in the event no errors occur during a Temporary Storage GET, PUT, or RELEASE. NORESP signifies "normal response" rather than "no response".

IDERROR: Specifies the entry label of the user-written routine to which control is to be passed in the event the symbolic destination identification referenced by a GET or RELEASE cannot be found in either main storage or auxiliary storage.

IOERROR: Specifies the entry label of the user-written routine to which control is to be passed in the event an input/output error occurs during a GET operation on auxiliary storage.

INVREQ: Specifies the entry label of the user-written routine to which control is to be passed in the event (1) a PUT is requested for data whose length is equal to zero or is greater than the block size of the auxiliary data set, or (2) the request is otherwise determined to be invalid.

The following are examples of the coding required to examine the response code provided by CICS at TCATSTR (TCATSRC if the language is ANS COBOL) and transfer control to the appropriate user-written exception handling routine.

For Assembler language:

```
        DFHTS TYPE=GET,                                          *
              DATAID=UNIQNME,                                    *
              TSDADDR=YES
        CLI   TCATSTR,X'00'
        BE    GOOD
        DFHPC TYPE=ABEND
GOOD    DS    0H
        .
        .
        .
```

For ANS COBOL:

```
        DFHTS TYPE=GET,                                          *
              DATAID=UNIQNME,                                    *
              TSDADDR=YES
        IF TCATSRC=' ' THEN GO TO GOOD.
        DFHPC TYPE=ABEND
GOOD.
        .
        .
        .
```

where the value specified within single quotes is a multipunch code for the required hexadecimal value. For example, a hexadecimal 00 has a multipunch code of 12-0-1-8-9.

For PL/I:

```
        DFHTS TYPE=GET,                                          *
              DATAID=UNIQNME,                                    *
              TSDADDR=YES
        IF TCATSTR='CCC00000'B THEN GO TO GOOD;
        DFHPC TYPE=ABEND
GOOD:
        .
        .
        .
```

TIME SERVICES

Time Management provides the capability, primarily through Interval Control and Task Control, to control various task functions based on the time of day or on intervals of time. Time services include:

1. Establish the partition/region exit time interval when CICS voluntarily relinquishes control to the operating system.
2. Provide system stall detection and corrective action (optional) based on the expiration of a user-provided time interval, in conjunction with other symptoms of a system stall condition.
3. Provide runaway task detection and corrective action capabilities (optional) based on the expiration of a user-provided time interval with an executing application program apparently in a logical loop.
4. Provide time of day in binary or packed decimal representation.
5. Provide task synchronization based on time-dependent events.
6. Provide automatic time-ordered task initiation with associated data retention and recovery support.

The services enumerated in items 1-3 are CICS system services and require no action on the part of the application programmer. The services enumerated in items 4-6 are available to the application programmer through use of the Interval Control macro instruction (DFHIC).

The following operands can be included in the DFHIC macro instruction:

```
DFHIC TYPE=GETIME,                                          *
      FORM=BINARY,PACKED,                                   *
      TIMADR=symbolic address,YES,                          *
      NORESP=symbolic address,                              *
      INVREQ=symbolic address

DFHIC TYPE=WAIT,                                            *
      INTRVAL=numeric value,YES,                            *
      TIME=numeric value,YES,                               *
      REQID=name,YES,                                       *
      NORESP=symbolic address,                              *
      INVREQ=symbolic address,                              *
      EXPIRD=symbolic address

DFHIC TYPE=POST,                                            *
      INTRVAL=numeric value,YES,                            *
      TIME=numeric value,YES,                               *
      REQID=name,YES,                                       *
      NORESP=symbolic address,                              *
      INVREQ=symbolic address,                              *
      EXPIRD=symbolic address

DFHIC TYPE=INITIATE,                                        *
      INTRVAL=numeric value,YES,                            *
      TIME=numeric value,YES,                               *
      REQID=name,YES,                                       *
      TRANSID=name,                                         *
      TRMIDNT=name,YES,                                     *
      NORESP=symbolic address,                              *
      INVREQ=symbolic address,                              *
      IRNIDER=symbolic address,                             *
      TRMIDER=symbolic address

DFHIC TYPE=PUT,                                             *
      INTRVAL=numeric value,YES,                            *
      TIME=numeric value,YES,                               *
```

```
        REQID=name,YES,                                          *
        TRANSID=name,                                            *
        TRMIDNT=name,YES,                                        *
        ICDADDR=symbolic address,YES,                            *
        NORESP=symbolic address,                                 *
        INVREQ=symbolic address,                                 *
        TRNIDER=symbolic address,                                *
        TRMIDER=symbolic address,                                *
        IOERROR=symbolic address

DFHIC   TYPE=GET,                                                *
        ICDADDR=symbolic address,YES,                            *
        NORESP=symbolic address,                                 *
        INVREQ=symbolic address,                                 *
        ENDDATA=symbolic address,                                *
        NOTFND=symbolic address,                                 *
        IOERROR=symbolic address

DFHIC   TYPE=RETRY,                                              *
        NORESP=symbolic address,                                 *
        INVREQ=symbolic address,                                 *
        NOTFND=symbolic address,                                 *
        IOERROR=symbolic address

DFHIC   TYPE=CANCEL,                                             *
        REQID=name,YES,                                          *
        NORESP=symbolic address,                                 *
        INVREQ=symbolic address,                                 *
        NOTFND=symbolic address

DFHIC   TYPE=CHECK,                                              *
        NORESP=symbolic address,                                 *
        INVREQ=symbolic address,                                 *
        EXPIRD=symbolic address,                                 *
        TRNIDER=symbolic address,                                *
        TRMIDER=symbolic address,                                *
        IOERROR=symbolic address,                                *
        NOTFND=symbolic address,                                 *
        ENDDATA=symbolic address
```

In the course of normal operation, CICS maintains the current time of day within the Common System Area (CSA); in binary form at CSACTODB, and in packed decimal form at CSATODP. These values are updated during task dispatching to reflect the time of day maintained by the operating system. The accuracy of these values depends upon the task mix and frequency of task switching occurences.

Since the time of day maintained by the operating system can be changed either by the operating system (for example, OS resetting the clock to zero at midnight) or by the console operator, CICS must recognize the situation where a "negative" change in the time of day has occurred, and must adjust expiration times maintained by CICS accordingly.

If the optional time adjustment feature of CICS Time Management is not included in CICS, any change to the operating system time of day involving midnight is represented by CICS as a value larger than the previous value (for example, 1:00 a.m. is represented as 2500 hours). If the optional time adjustment feature is included in CICS, and if either the time-ordered task synchronization feature or automatic task initiation feature of CICS Time Management is also included, any change to the operating system time of day is automatically reflected in the expiration times maintained by CICS.

In the case of CICS/OS, when the operating system time of day is set to zero at midnight (and the time adjustment feature has been included in CICS), CICS/OS adjusts the expiration times of day it maintains and then resets its time of day to zero. In the case of both CICS/OS and CICS/DOS, when the operating system time of day is changed by the console operator to a value less than the previous value, CICS adjusts the expiration times it maintains to reflect the negative value and then resets its time of day to the time of day maintained by the operating system. The optional time adjustment feature thus makes it possible for CICS to be operated on a continuous round-the-clock basis.

TIME-OF-DAY SERVICES (GETIME)

In the course of normal operation, CICS maintains the current time of day in two forms within the Common System Area (CSA); in binary form at CSACTODB; and in packed decimal form at CSATODP. These values are updated periodically during task dispatching, their accuracy being dependent upon the task mix and frequency of task switching occurrences.

Tasks can obtain a more current time of day by issuing the

```
DFHIC TYPE=GETIME,                                          *
      FORM=BINARY,PACKED,                                   *
      TIMADR=symbolic address,YES,                          *
      NORESP=symbolic address,                              *
      INVREQ=symbolic address
```

macro instruction. This macro instruction causes one or 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 user-specified location. A discussion of the operands that can be included in the DFHIC TYPE=GETIME macro instruction follows. (The keywords used to access user-written exception handling routines are discussed in the section "Test Response to a Request for Time Services".)

FORM: This optional operand is used to indicate which representation of time of day is desired. The default is FORM=BINARY.

FORM=PACKED is used to indicate that the packed decimal representation of the time of day is desired. 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 use of this operand causes both the packed and binary representations of the time of day to be updated and retained in the CSA.

Note: As a performance consideration, it should be taken into account that lengthy conversion routines are executed each time the FORM=PACKED operand is used.

FORM=BINARY is used when the binary representation of time of day is desired. The binary representation is expressed as a four-byte positive value in hundredths of a second. The use of this operand causes only the binary representation of time of day to be updated and retained in the CSA.

TIMADR: This optional operand is used when the requested time of day is to be returned to a user-defined four-byte location. The application programmer can accomplish this in either of two ways: (1) by including the TIMADR=symbolic address operand in the DFHIC TYPE=GETIME macro instruction, or (2) by coding a single instruction, prior to issuing

For ANS COBOL:

```
        02 TSIOABAR PICTURE S9(8) USAGE IS COMPUTATIONAL.
    01  DFHTSICA COPY DFHTSIOA.
        02 DATA PICTURE X(10).
           .
           .
           .
        MOVE LENGTH TO TSIOAVRL.
        MOVE MESSAGE TO DATA.
    DFHTS TYPE=PUT,                                              *
          DATAID=UNIQNME,                                        *
          TSDADDR=TSIOAVRL
           .
           .
           .
```

For PL/I:

```
    %INCLUDE DFHTSIOA;
        2 DATA CHAR(10);
           .
           .
           .
    TSIOAVRL=LENGTH;
    DATA=MESSAGE;
    DFHTS TYPE=PUT,                                              *
          DATAID=UNIQNME,                                        *
          TSDADDR=TSIOAVRL
           .
           .
```

RETRIEVE TEMPORARY DATA (GET)

The application programmer can retrieve temporary data from a symbclic source in main or auxiliary storage by issuing the

```
    DFHTS TYPE=GET,                                              *
          DATAID=name,                                           *
          TSDADDR=symbclic address,YES,                          *
          RELEASE=YES,NO,                                        *
          NORESP=symbolic address,                               *
          IDERROR=symbclic address,                              *
          IOERROR=symbclic address
```

macro instruction. Data retrieved from temporary storage is placed in either a user-provided storage area or an area (transaction storage) acquired for the user by Temporary Storage Control.

The application programmer must specify the parameters he requires to retrieve temporary data. He can do this in either of two ways: (1) by including the parameters in operands of the DFHTS TYPE=GET macro instruction, or (2) by coding instructions, prior to issuing the DFHTS TYPE=GET macro instruction, that dynamically move these parameters to fields in the TCA. If the parameters are included in operands of the DFHTS TYPE=GET macro instruction, the applicable keywords are DATAID, TSDADDR, and RELEASE.

A discussion of the operands that can be included in the DFHTS TYPE=GET macro instruction follows. (The keywords used to access user-written exception handling routines are discussed in the section "Test Response to a Request for Temporary Storage Services".)

DATAID: Specifies the name assigned to the temporary data at the time it was placed in temporary storage. This operand can be omitted if the application programmer has previously placed the name in the TCATSDI field of the TCA.

TSIADDR: Specifies the symbolic name of the user-provided storage area into which the temporary data is to be read (or moved). TSDADDR=YES must be coded if the application programmer has previously placed this symbolic address in the TCA at TCATSDA. If this operand is omitted, Temporary Storage Control obtains a storage area, moves or reads temporary data into the area, and returns the address of the area to the user in the TCA at TCATSDA.

RELEASE: Specifies whether the data is to be released following this acquisition. The default is RELEASE=NO.

The following are examples of the coding required to read a record from temporary storage. In these examples, the data is moved to the area defined by the user in the TSDADDR operand. If the TSDADDR operand is omitted, the data is moved into a storage area obtained by Temporary Storage Control, and the address of the storage area is returned to the user at TCATSDA.

For Assembler language:

```
TSIOABAR   EQU     7
           COPY    DFHTSIOA
           .
           .
           .
       DFHTS TYPE=GET,                                          *
             DATAID=UNIQNME,                                    *
             TSDADDR=TSIOAVRL
           .
           .
           .
```

For ANS COBOL:

```
           02  TSIOABAR PICTURE S9(8) USAGE IS COMPUTATIONAL.
           .
           .
           .
       01  DFHTSIOA COPY DFHTSIOA.
           .
           .
           .
       DFHTS TYPE=GET,                                          *
             DATAID=UNIQNME,                                    *
             TSDADDR=TSIOAVRL
           .
           .
           .
```

For PL/I:

```
%INCLUDE DFHTSIOA;
    2 DATA CHAR(10);
    .
    .
    .
DFHTS TYPE=GET,                                          *
      DATAID=UNIQNME,                                    *
      TSDADDR=TSIOAVRL
    .
    .
    .
```

RELEASE TEMPORARY DATA (RELEASE)

The application programmer can release temporary data from main or auxiliary storage by issuing the

```
DFHTS TYPE=RELEASE,                                      *
      DATAID=name,                                       *
      NORESP=symbolic address,                           *
      IDERROR=symbolic address
```

macro instruction. If the data was stored in main storage, the area is freed and returned to the available dynamic area. If the data was stored in auxiliary storage, the space is made available for other data.

Temporary data should be released at the earliest possible time to avoid suspended tasks.

A discussion of the DATAID=name operand of the DFHTS TYPE=RELEASE macro instruction follows. (The keywords used to access user-written exception handling routines are discussed in the section "Test Response to a Request for Temporary Storage Services".)

DATAID: Specifies the name assigned to the data to be released from temporary storage. This operand can be omitted if the application programmer has previously placed the name in the TCATSDI field of the TCA.

The following are examples of the coding required to release a record from temporary storage.

For Assembler language:

```
MVC   TCATSDI,=C'UNIQNME'
DFHTS TYPE=RELEASE
```

For ANS COBOL:

```
MOVE 'UNIQNME' TO TCATSDI.
DFHTS TYPE=RELEASE
```

For PL/I:

```
TCATSDI='UNIQNME';
DFHTS TYPE=RELEASE
```

TEST RESPONSE TO A REQUEST FOR TEMPORARY STORAGE SERVICES (CHECK)

One of the ways the application programmer can test the response to a request for temporary storage services is by issuing the

```
DFHTS TYPE=CHECK,                                              *
      NORESP=symbolic address,                                 *
      IDERROR=symbolic address,                                *
      IOERROR=symbolic address,                                *
      INVREQ=symbolic address
```

macro instruction, which provides for the testing of response codes and the routing of control to the appropriate user-written exception handling routines. This macro instruction provides an exception handling facility that can be used in the manner of a subroutine.

CICS automatically places the appropriate response code in the TCA at TCATSIR (TCATSRC if the language is ANS COBOL) after completion of the temporary storage service requested. The application programmer must specify the entry labels (symbolic addresses) he requires to access the appropriate exception handling routine previously supplied by the user.

The response codes are as follows:

CONDITION	ASSEMBLER	ANS COBOL	PL/I
NORESP	X'00'	12-0-1-8-9	00000000
IDERROR	X'02'	12-2-9	00000010
IOERROR	X'04'	12-4-9	00000100
INVREQ	X'20'	11-0-1-8-9	00100000

If the application programmer does not use the DFHTS TYPE=CHECK macro instruction, he can specify the entry labels (symbolic addresses) in either of two other ways: (1) by including the entry labels in operands of any other DFHTS macro instruction, or (2) by coding instructions immediately following the DFHTS macro instruction that examine the response code provided by CICS at TCATSIR (TCATSRC if the language is ANS COBOL) and transfer control to the appropriate routine.

If the DFHTS TYPE=CHECK macro instruction is used by the application programmer, it should usually immediately follow another DFHTS macro instruction. The applicable keywords are NORESP, IDERROR, IOERROR, and INVREQ.

If the application programmer does not check for a particular response to his service request, and if that exception condition occurs, program flow proceeds to the next sequential instruction.

The operands that can be used to test the response to a request for temporary storage services are as follows.

NORESP: Specifies the entry label of the user-written routine to which control is to be passed in the event no errors occur during a Temporary Storage GET, PUT, or RELEASE. NORESP signifies "normal response" rather than "no response".

IDERROR: Specifies the entry label of the user-written routine to which control is to be passed in the event the symbolic destination identification referenced by a GET or RELEASE cannot be found in either main storage or auxiliary storage.

IOERROR: Specifies the entry label of the user-written routine to which control is to be passed in the event an input/output error occurs during a GET operation on auxiliary storage.

INVREQ: Specifies the entry label of the user-written routine to which control is to be passed in the event (1) a PUT is requested for data whose length is equal to zero or is greater than the block size of the auxiliary data set, or (2) the request is otherwise determined to be invalid.

The following are examples of the coding required to examine the response code provided by CICS at TCATSTR (TCATSRC if the language is ANS COBOL) and transfer control to the appropriate user-written exception handling routine.

For Assembler language:

```
        DFHTS TYPE=GET,                                          *
              DATAID=UNIQNME,                                    *
              TSDADDR=YES
        CLI   TCATSTR,X'00'
        BE    GOOD
        DFHPC TYPE=ABEND
GOOD    DS    0H
        .
        .
        .
```

For ANS COBOL:

```
        DFHTS TYPE=GET,                                          *
              DATAID=UNIQNME,                                    *
              TSDADDR=YES
        IF TCATSRC=' ' THEN GO TO GOOD.
        DFHPC TYPE=ABEND
GOOD.
        .
        .
        .
```

where the value specified within single quotes is a multipunch code for the required hexadecimal value. For example, a hexadecimal 00 has a multipunch code of 12-0-1-8-9.

For PL/I:

```
        DFHTS TYPE=GET,                                          *
              DATAID=UNIQNME,                                    *
              TSDADDR=YES
        IF TCATSTR='CCC00000'B THEN GO TO GOOD;
        DFHPC TYPE=ABEND
GOOD:
        .
        .
        .
```

TIME SERVICES

Time Management provides the capability, primarily through Interval Control and Task Control, to control various task functions based on the time of day or on intervals of time. Time services include:

1. Establish the partition/region exit time interval when CICS voluntarily relinquishes control to the operating system.
2. Provide system stall detection and corrective action (optional) based on the expiration of a user-provided time interval, in conjunction with other symptoms of a system stall condition.
3. Provide runaway task detection and corrective action capabilities (optional) based on the expiration of a user-provided time interval with an executing application program apparently in a logical loop.
4. Provide time of day in binary or packed decimal representation.
5. Provide task synchronization based on time-dependent events.
6. Provide automatic time-ordered task initiation with associated data retention and recovery support.

The services enumerated in items 1-3 are CICS system services and require no action on the part of the application programmer. The services enumerated in items 4-6 are available to the application programmer through use of the Interval Control macro instruction (DFHIC).

The following operands can be included in the DFHIC macro instruction:

```
DFHIC TYPE=GETIME,                                          *
      FORM=BINARY,PACKED,                                   *
      TIMADR=symbolic address,YES,                          *
      NORESP=symbolic address,                              *
      INVREQ=symbolic address

DFHIC TYPE=WAIT,                                            *
      INTRVAL=numeric value,YES,                            *
      TIME=numeric value,YES,                               *
      REQID=name,YES,                                       *
      NORESP=symbolic address,                              *
      INVREQ=symbolic address,                              *
      EXPIRD=symbolic address

DFHIC TYPE=POST,                                            *
      INTRVAL=numeric value,YES,                            *
      TIME=numeric value,YES,                               *
      REQID=name,YES,                                       *
      NORESP=symbolic address,                              *
      INVREQ=symbolic address,                              *
      EXPIRD=symbolic address

DFHIC TYPE=INITIATE,                                        *
      INTRVAL=numeric value,YES,                            *
      TIME=numeric value,YES,                               *
      REQID=name,YES,                                       *
      TRANSID=name,                                         *
      TRMIDNT=name,YES,                                     *
      NORESP=symbolic address,                              *
      INVREQ=symbolic address,                              *
      IRNIDER=symbolic address,                             *
      TRMIDER=symbolic address

DFHIC TYPE=PUT,                                             *
      INTRVAL=numeric value,YES,                            *
      TIME=numeric value,YES,                               *
```

```
        REQID=name,YES,                                          *
        TRANSID=name,                                            *
        TRMIDNT=name,YES,                                        *
        ICDADDR=symbolic address,YES,                            *
        NORESP=symbolic address,                                 *
        INVREQ=symbolic address,                                 *
        TRNIDER=symbolic address,                                *
        TRMIDER=symbolic address,                                *
        IOERROR=symbolic address

DFHIC   TYPE=GET,                                                *
        ICDADDR=symbolic address,YES,                            *
        NORESP=symbolic address,                                 *
        INVREQ=symbolic address,                                 *
        ENDDATA=symbolic address,                                *
        NOTFND=symbolic address,                                 *
        IOERROR=symbolic address

DFHIC   TYPE=RETRY,                                              *
        NORESP=symbolic address,                                 *
        INVREQ=symbolic address,                                 *
        NOTFND=symbolic address,                                 *
        IOERROR=symbolic address

DFHIC   TYPE=CANCEL,                                             *
        REQID=name,YES,                                          *
        NORESP=symbolic address,                                 *
        INVREQ=symbolic address,                                 *
        NOTFND=symbolic address

DFHIC   TYPE=CHECK,                                              *
        NORESP=symbolic address,                                 *
        INVREQ=symbolic address,                                 *
        EXPIRD=symbolic address,                                 *
        TRNIDER=symbolic address,                                *
        TRMIDER=symbolic address,                                *
        IOERROR=symbolic address,                                *
        NOTFND=symbolic address,                                 *
        ENDDATA=symbolic address
```

In the course of normal operation, CICS maintains the current time of day within the Common System Area (CSA); in binary form at CSACTODB, and in packed decimal form at CSATODP. These values are updated during task dispatching to reflect the time of day maintained by the operating system. The accuracy of these values depends upon the task mix and frequency of task switching occurences.

Since the time of day maintained by the operating system can be changed either by the operating system (for example, OS resetting the clock to zero at midnight) or by the console operator, CICS must recognize the situation where a "negative" change in the time of day has occurred, and must adjust expiration times maintained by CICS accordingly.

If the optional time adjustment feature of CICS Time Management is not included in CICS, any change to the operating system time of day involving midnight is represented by CICS as a value larger than the previous value (for example, 1:00 a.m. is represented as 2500 hours). If the optional time adjustment feature is included in CICS, and if either the time-ordered task synchronization feature or automatic task initiation feature of CICS Time Management is also included, any change to the operating system time of day is automatically reflected in the expiration times maintained by CICS.

In the case of CICS/OS, when the operating system time of day is set to zero at midnight (and the time adjustment feature has been included in CICS), CICS/OS adjusts the expiration times of day it maintains and then resets its time of day to zero. In the case of both CICS/OS and CICS/DOS, when the operating system time of day is changed by the console operator to a value less than the previous value, CICS adjusts the expiration times it maintains to reflect the negative value and then resets its time of day to the time of day maintained by the operating system. The optional time adjustment feature thus makes it possible for CICS to be operated on a continuous round-the-clock basis.

TIME-OF-DAY SERVICES (GETIME)

In the course of normal operation, CICS maintains the current time of day in two forms within the Common System Area (CSA); in binary form at CSACTODB; and in packed decimal form at CSATODP. These values are updated periodically during task dispatching, their accuracy being dependent upon the task mix and frequency of task switching occurrences.

Tasks can obtain a more current time of day by issuing the

```
DFHIC TYPE=GETIME,                                          *
      FORM=BINARY,PACKED,                                   *
      TIMADR=symbolic address,YES,                          *
      NORESP=symbolic address,                              *
      INVREQ=symbolic address
```

macro instruction. This macro instruction causes one or 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 user-specified location. A discussion of the operands that can be included in the DFHIC TYPE=GETIME macro instruction follows. (The keywords used to access user-written exception handling routines are discussed in the section "Test Response to a Request for Time Services".)

FORM: This optional operand is used to indicate which representation of time of day is desired. The default is FORM=BINARY.

FORM=PACKED is used to indicate that the packed decimal representation of the time of day is desired. The packed decimal representation is expressed as a four-byte positive signed value of the form 0HHMMSSt+ where the seconds are truncated to tenths of a second. The use of this operand causes both the packed and binary representations of the time of day to be updated and retained in the CSA.

Note: As a performance consideration, it should be taken into account that lengthy conversion routines are executed each time the FORM=PACKED operand is used.

FORM=BINARY is used when the binary representation of time of day is desired. The binary representation is expressed as a four-byte positive value in hundredths of a second. The use of this operand causes only the binary representation of time of day to be updated and retained in the CSA.

TIMADR: This optional operand is used when the requested time of day is to be returned to a user-defined four-byte location. The application programmer can accomplish this in either of two ways: (1) by including the TIMADR=symbolic address operand in the DFHIC TYPE=GETIME macro instruction, or (2) by coding a single instruction, prior to issuing

the DFHIC TYPE=GETIME macro instruction, that dynamically moves the address to the TCAICDA field of the TCA. If the latter is used, the TIMADR=YES operand must also be included in the DFHIC TYPE=GETIME macro instruction. If this operand is omitted, only the appropriate fields in the CSA are updated.

The following is an example of the coding required to request the time of day:

```
DFHIC TYPE=GETIME,              REQUEST CURRENT TIME-OF-DAY     *
      FORM=PACKED,              PACKED DECIMAL FORM             *
      TIMADR=CLOCK              SYMBOLIC ADDRESS FOR RESPONSE
```

The following are examples of the coding required to dynamically request the time of day.

For Assembler language:

```
      MVC TCAICDA,=A(CLOCK)     MOVE ADDR FOR RESPONSE TO TCA
      .
      .
      .
      DFHIC TYPE=GETIME,        REQUEST CURRENT TIME-OF-DAY     *
            FORM=PACKED,        PACKED DECIMAL FORM             *
            TIMADR=YES          RESPONSE ADDRESS GIVEN
```

For ANS COBOL:

```
      MOVE CLOCKADR TO TCAICDA. NOTE MOVE ADDR FOR RESP TO TCA.
      .
      .
      .
      DFHIC TYPE=GETIME,        REQUEST CURRENT TIME-OF-DAY     *
            FORM=PACKED,        PACKED DECIMAL FORM             *
            TIMADR=YES          RESPONSE ADDRESS GIVEN
```

For PL/I:

```
      TCAICDA=ADDR(CLOCK);      /*MOVE ADDR FOR RESP TO TCA*/
      .
      .
      .
      DFHIC TYPE=GETIME,        REQUEST CURRENT TIME-OF-DAY     *
            FORM=PACKED,        PACKED DECIMAL FORM             *
            TIMADR=YES          RESPONSE ADDRESS GIVEN
```

TIME-ORDERED TASK SYNCHRONIZATION (WAIT, POST)

The task synchronization feature of CICS Time Management provides the capability to either delay the processing of a task until a specified time occurs or to signal a processing 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 "Time-Ordered Request Cancellation" later in this section.)

Delay the Processing of a Task (WAIT)

The application programmer can request that the processing of a task be suspended until a given time expires by issuing the

```
DFHIC TYPE=WAIT,                                              *
      INTRVAL=numeric value,YES,                              *
      TIME=numeric value,YES,                                 *
      REQID=name,YES,                                         *
      NORESP=symbolic address,                                *
      INVREQ=symbolic address,                                *
      EXPIRD=symbolic address
```

macro instruction. This macro instruction causes the task to temporarily suspend its own processing, and to resume control at a specified time of day or after a specified interval of time has elapsed. It supersedes and cancels any previously initiated DFHIC TYPE=POST request for the task.

The application programmer must specify the parameters required in either of two ways: (1) by including the parameters in operands of the DFHIC TYPE=WAIT macro instruction, or (2) by coding instructions, prior to issuing the DFHIC TYPE=WAIT macro instruction, that dynamically move these parameters to fields in the TCA. If the parameters are included in operands of the DFHIC TYPE=WAIT macro instruction, the applicable keywords are INTRVAL, TIME, and REQID. (The keywords used to access user-written exception handling routines are discussed in the section "Test Response to a Request for Time Services".)

The numeric value specified in either the INTRVAL operand or TIME operand is used by CICS to calculate the time of day the requested time service is to be provided. If the calculated time of day is the same as the current clock time, or up to and including six hours preceding the current clock time, the specified time is considered to have elapsed (occurred) and the requested service is provided immediately. If the calculated time of day is in advance of the current clock time, the requested service is provided when the specified time occurs. If the calculated time of day precedes the current clock time by more than six hours, the requested service is provided the next day at the specified time.

Note: Users of CICS/OS must be aware that the current clock time is reset to zero each day to represent midnight. CICS makes no attempt to calculate a time of day based on a clock time less than zero.

INTRVAL: This operand is used to specify the interval of time a task is to be suspended in response to a DFHIC TYPE=WAIT request. The interval of time is specified as a numeric value 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 DFHIC TYPE=WAIT macro instruction is executed to calculate the time of day (clock time) at which the posting is to occur. The minimum value that may be specified is one second.

The numeric value can be specified in the DFHIC TYPE=WAIT macro instruction, or it can be dynamically moved to the TCAICRT field of the TCA prior to issuing the DFHIC TYPE=WAIT macro instruction. In the latter case, the INTRVAL=YES operand must be included in the macro instruction.

The INTRVAL operand and TIME operand are mutually exclusive and may not be used in the same macro instruction.

TIME: This operand is used to specify the time of day at which the processing of a task is to begin. The time of day is expressed as a numeric value 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.

The numeric value can be specified in the DFHIC TYPE=WAIT macro instruction, or it can be dynamically moved to the TCAICRT field of the TCA prior to issuing the DFHIC TYPE=WAIT macro instruction. In the latter case, the TIME=YES operand must be included in the macro instruction.

The TIME operand and INTRVAL operand are mutually exclusive and may not be used in the same macro instruction.

REQID: Each time-ordered request has a unique Request Identification assigned to it. Its purpose is to provide a means of symbolically identifying the request and any data associated with it. Unless otherwise instructed, CICS generates a unique Request Identification.

The optional REQID operand allows the user to supply the unique Request Identification as part of the DFHIC TYPE=WAIT service request in either of two ways: (1) by specifying a maximum of eight characters in the REQID operand, or (2) by dynamically moving an eight-byte Request Identification to the TCAICQID field prior to issuing the DFHIC TYPE=WAIT macro instruction. In the latter case, the REQID=YES operand must be included in the macro instruction.

The REQID operand should be used when a task issues the DFHIC TYPE=WAIT macro instruction, if the application programmer wishes to provide another task with the capability of cancelling the unexpired WAIT request. (See the discussion of the DFHIC TYPE=CANCEL macro instruction.)

The following is an example of the coding required to temporarily suspend the processing of a task for a specified period of time:

```
      DFHIC TYPE=WAIT,                DELAY TASK PROCESSING,          *
            INTRVAL=500,              WAIT 5 MINUTES 0 SECONDS        *
            REQID=GXLBZQMR            UNIQUE REQUEST ID
```

The following are examples of the coding required to dynamically request the suspension of a task until a specified time of day.

For Assembler language:

```
      MVC TCAICRT,=PL4'124500'        MOVE 12:45 TO TCA
      MVC TCAICQID,UNIQCODE           UNIQUE REQUEST ID TO TCA
      .
      .
      .
      DFHIC TYPE=WAIT,                DELAY TASK PROCESSING           *
            TIME=YES,                 EXPIRATION TIME GIVEN           *
            REQID=YES                 UNIQUE ID GIVEN
```

For ANS COBOL:

```
      MOVE 124500 TO TCAICRT.         NOTE MOVE 12:45 TO TCA
      MOVE UNIQCODE TO TCAICQID.      NOTE UNIQUE REQUEST ID TO TCA.
      .
      .
      .
      DFHIC TYPE=WAIT,                DELAY TASK PROCESSING±          *
            TIME=YES,                 EXPIRATION TIME GIVEN           *
            REQID=YES                 UNIQUE ID GIVEN
```

For PL/I:

```
TCAICRT=124500;                 /*MOVE 12:45 TO TCA*/
TCAICQID=UNIQCODE;              /*UNIQUE REQUEST ID TO TCA*/
.
.
.
DFHIC TYPE=WAIT,                DELAY TASK PROCESSING          *
      TIME=YES,                 EXPIRATION TIME GIVEN          *
      REQID=YES                 UNIQUE ID GIVEN
```

Signal the Expiration of a Specified Time (POST)

The application programmer can request that CICS indicate to a processing task when a given time has expired by issuing the

```
DFHIC TYPE=POST,                                               *
      INTRVAL=numeric value,YES,                               *
      TIME=numeric value,YES,                                  *
      REQID=name,YES,                                          *
      NORESP=symbolic address,                                 *
      INVREQ=symbolic address,                                 *
      EXPIRD=symbolic address
```

macro instruction. In response to this macro instruction, CICS sets a series of bits in a Timer Event Control Area available to the user for testing. The address of the Timer Event Control Area is returned to the requesting task in the TCAICTEC field after issuing the DFHIC TYPE=POST macro instruction.

The Timer Event Control Area provided by CICS is a four-byte storage area initialized to binary zeros at the time the DFHIC TYPE=POST macro instruction is issued. When CICS determines that the specified time has expired, byte 0 is set to a hexadecimal 40 and byte 2 is set to a hexadecimal 80 (the other bytes are set to zero). 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 referenced in a DFHKC TYPE=WAIT macro instruction. (See the discussion of task synchronization in the section "Task Services".)

The Timer Event Control Area provided to the user is not released or altered (except as described above) until the first of any of the following events occur:

1. The task issues a subsequent DFHIC TYPE=WAIT, DFHIC TYPE=POST, DFHIC TYPE=INITIATE or DFHIC TYPE=PUT macro request.
2. The task issues a DFHIC TYPE=CANCEL macro request on behalf of its own previously issued DFHIC TYPE=POST request (this releases the storage area occupied by the Timer Event Control Area).
3. The task terminates, normally or abnormally.

A task can only have 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 made 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 macro instruction (see the discussion of task synchronization in the section "Task Services") or indirectly by requesting a CICS service which in turn initiates a task service on behalf of the task.

The application programmer must specify the parameters required in either of two ways: (1) by including the parameters in operands of the DFHIC TYPE=POST macro instruction, or (2) by coding instructions, prior to issuing the DFHIC TYPE=POST macro instruction, that dynamically move these parameters to fields in the TCA. If the parameters are included in operands of the DFHIC TYPE=POST macro instruction, the applicable keywords are INTRVAL, TIME, and REQID. (The keywords used to access user-written exception handling routines are discussed in the section "Test Response to a Request for Time Services".)

The numeric value specified in either the INTRVAL operand or TIME operand is used by CICS to calculate the time of day the requested time service is to be provided. If the calculated time of day is the same as the current clock time, or up to and including six hours preceding the current clock time, the specified time is considered to have elapsed (occurred) and the requested service is provided immediately. If the calculated time of day is in advance of the current clock time, the requested service is provided when the specified time occurs. If the calculated time of day precedes the current clock time by more than six hours, the requested service is provided the next day at the specified time.

Note: Users of CICS/OS must be aware that the current clock time is reset to zero each day to represent midnight. CICS makes no attempt to calculate a time of day based on a clock time less than zero.

INTRVAL: This operand is used to specify the interval of time that is to elapse in response to a DFHIC TYPE=POST request. The interval of time is specified as a numeric value 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 DFHIC TYPE=POST macro instruction is executed to calculate the time of day (clock time) at which the task is to be resumed. The minimum value that may be specified is one second.

The numeric value can be specified in the DFHIC TYPE=POST macro instruction, or it can be dynamically moved to the TCAICRT field of the TCA prior to issuing the DFHIC TYPE=POST macro instruction. In the latter case, the INTRVAL=YES operand must be included in the macro instruction.

The INTRVAL operand and TIME operand are mutually exclusive and may not be used in the same macro instruction.

TIME: This operand is used to specify the time of day at which the posting action in response to a DFHIC TYPE=POST request is to occur. The time of day is expressed as a numeric value 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.

The numeric value can be specified in the DFHIC TYPE=POST macro instruction, or it can be dynamically moved to the TCAICRT field of the TCA prior to issuing the DFHIC TYPE=POST macro instruction. In the latter case, the TIME=YES operand must be included in the macro instruction.

The TIME operand and INTRVAL operand are mutually exclusive and may not be used in the same macro instruction.

REQID: Each time-ordered request has a unique Request Identification assigned to it. Its purpose is to provide a means of symbolically identifying the request and any data associated with it. Unless otherwise instructed, CICS generates a unique Request Identification.

The optional REQID operand allows the user to supply the unique Request Identification as part of the DFHIC TYPE=POST service request in either of two ways: (1) by specifying a maximum of eight characters in the REQID operand, or (2) by dynamically moving an eight-byte Request Identification to the TCAICQID field prior to issuing the DFHIC TYPE=POST macro instruction. In the latter case, the REQID=YES operand must be included in the macro instruction.

If the REQID operand is omitted from the DFHIC TYPE=POST macro instruction, the unique Request Identification generated by CICS is returned to the user in the TCAICQID field.

The following is an example of the coding required to request that CICS signal the task when a specified interval of time has elapsed:

```
DFHIC TYPE=POST,                SIGNAL WHEN INTERVAL PASSES     *
      INTRVAL=30                INTERVAL IS 30 SECONDS
```

The following are examples of the coding required to dynamically request that CICS signal the task when the specified time of day occurs.

For Assembler language:

```
MVC TCAICRT,PACKTIME            STORE CALCULATED EXPIR TIME
.
.
.
DFHIC TYPE=POST,                SIGNAL WHEN TIME OCCURS         *
      TIME=YES                  EXPIRATION TIME GIVEN
MVC UNIQCODE,TCAICQID           SAVE CICS UNIQUE REQUEST ID
```

For ANS COBOL:

```
MOVE PACKTIME TO TCAICRT.       NOTE STORE CALC EXPIR TIME.
.
.
.
DFHIC TYPE=POST,                SIGNAL WHEN TIME OCCURS         *
      TIME=YES                  EXPIRATION TIME GIVEN
MOVE TCAICQID TO UNIQCODE.      SAVE CICS UNIQUE REQUEST ID
```

For PL/I:

```
TCAICRT=PACKTIME;               /*STORE CALCULATED EXPIR TIME*/
.
.
.
DFHIC TYPE=POST,                SIGNAL WHEN TIME OCCURS         *
      TIME=YES                  EXPIRATION TIME GIVEN
UNIQCODE=TCAICQID;              SAVE CICS UNIQUE REQUEST ID
```

AUTOMATIC TIME-ORDERED TASK INITIATION (INITIATE, PUT)

This feature of Time Management allows a task to initiate another task at some future time and, optionally, to pass data to that task. The automatic task initiation services available through DFHIC macro instructions include:

1. Request that a task be initiated at some future time.
2. Request that data be stored for a task which is to be initiated at some future time.

Task Initiation Without Data (INITIATE)

The application programmer can request that another task be initiated at some future time by issuing the:

```
DFHIC TYPE=INITIATE,                                          *
      INTRVAL=numeric value,YES,                              *
      TIME=numeric value,YES,                                 *
      REQID=name,YES,                                         *
      TRANSID=name,                                           *
      TRMIDNT=name,YES,                                       *
      NORESP=symbolic address,                                *
      INVREQ=symbolic address,                                *
      TRNIDER=symbolic address,                               *
      TRMIDER=symbolic address
```

macro instruction. Through the use of this macro instruction the application programmer provides the symbolic Transaction Identification of the task to be initiated at some future time and other information pertaining to the task. CICS queues the request until the specified time occurs. Then, as soon as all necessary resources are available (for example, a terminal), the task is initiated. Only one task is initiated if multiple DFHIC TYPE=INITIATE requests (all for the same transaction and terminal) expire at the same time or prior to terminal availability. The DFHIC TYPE=INITIATE macro instruction is used when no data is to be passed to the future task. It supersedes and cancels any previously initiated DFHIC TYPE=POST request for the task.

The application programmer must specify the parameters required in either of two ways: (1) by including the parameters in operands of the DFHIC TYPE=INITIATE macro instruction, or (2) by coding instructions, prior to issuing the DFHIC TYPE=INITIATE macro instruction, that dynamically move these parameters to fields in the TCA. If the parameters are included in operands of the DFHIC TYPE=INITIATE macro instruction, the applicable keywords are INTRVAL, TIME, REQID, TRANSID, and TRMIDNT. (The keywords used to access user-written exception handling routines are discussed in the section "Test Response to a Request for Time Services".)

The numeric value specified in either the INTRVAL operand or TIME operand is used by CICS to calculate the time of day the requested time service is to be provided. If the calculated time of day is the same as the current clock time, or up to and including six hours preceding the current clock time, the specified time is considered to have elapsed (occurred) and the requested service is provided immediately. If the calculated time of day is in advance of the current clock time, the requested service is provided when the specified time occurs. If the calculated time of day precedes the current clock time by more than six hours, the requested service is provided the next day at the specified time.

Note: Users of CICS/OS must be aware that the current clock time is reset to zero each day to represent midnight. CICS makes no

attempt to calculate a time of day based on a clock time less than zero.

INTRVAL: This operand is used to specify the interval of time after which the task is to be automatically initiated in response to a DFHIC=TYPE=INITIATE request. The interval of time is specified as a numeric value 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 DFHIC TYPE=INITIATE macro instruction is executed to calculate the time of day (clock time) at which the task is to be automatically initiated. The minimum value that may be specified is one second.

The numeric value can be specified in the DFHIC TYPE=INITIATE macro instruction, or it can be dynamically moved to the TCAICRT field of the TCA prior to issuing the DFHIC TYPE=INITIATE macro instruction. In the latter case, the INTRVAL=YES operand must be included in the macro instruction.

The INTRVAL operand and TIME operand are mutually exclusive and may not be used in the same macro instruction.

TIME: This operand is used to specify the time of day at which the task is to be automatically initiated in response to a DFHIC TYPE=INITIATE request. The time of day is expressed as a numeric value 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.

The numeric value can be specified in the DFHIC TYPE=INITIATE macro instruction, or it can be dynamically moved to the TCAICRT field of the TCA prior to issuing the DFHIC TYPE=INITIATE macro instruction. In the latter case, the TIME=YES operand must be included in the macro instruction.

The TIME operand and INTRVAL operand are mutually exclusive and may not be used in the same macro instruction.

REQID: Each time-ordered request has a unique Request Identification assigned to it. Its purpose is to provide a means of symbolically identifying the request. Unless otherwise instructed, CICS generates a unique Request Identification.

The optional REQID operand allows the user to supply the unique Request Identification as part of the DFHIC TYPE=INITIATE request in either of two ways: (1) by specifying a maximum of eight characters in the REQID operand or (2) by dynamically moving an eight-byte Request Identification to the TCAICQID field prior to issuing the DFHIC TYPE=INITIATE macro instruction. In the latter case, the REQID=YES operand must be included in the macro instruction.

If the REQID operand is omitted from the DFHIC TYPE=INITIATE macro instruction, the Unique Request identification provided by CICS is returned to the user in the TCAICQID field.

TRANSID: This operand is used to supply the symbolic Transaction Identification of the future task. This operand can be omitted provided the application programmer has placed the symbolic Transaction Identification in the TCAICTI field prior to issuing the DFHIC TYPE=INITIATE macro instruction. CICS validates the symbolic

Transaction Identification through a scan of the Program Control Table at the time of the initial macro request, providing a response code at TCAICTR (TCAICRC if the language is ANS COBOL) without servicing the request if it fails tc locate a matching Transaction Identification.

TRMIDNT: This operand is used when the future task must communicate with a terminal. The symbolic Terminal Identification can be included in the DFHIC TYPE=INITIATE macro instruction, or can be dynamically moved to the TCAICTID prior to issuing the DFHIC TYPE=INITIATE macro instruction. In the latter case, the TRMIDNT=YES operand must be included in the macro instruction. CICS validates the symbolic Terminal Identification through a scan of the Terminal Control Table at the time of the initial macro request, providing a response code at TCAICTR (TCAICRC if the language is ANS COBOL) without servicing the request if it fails to locate a matching Terminal Identification. The TRMIDNT operand is omitted from the DFHIC TYPE=INITIATE macro instruction if no association with a terminal is required.

The following is an example of the coding required to request automatic initiation of a task not associated with a terminal without passing data to the task:

```
DFHIC TYPE=INITIATE,          REQUEST TASK INITIATION          *
      INTRVAL=10000,          IN ONE HOUR                      *
      TRANSID=TRNL            TRANSACTION IDENTIFICATION
```

The following are examples of the coding required to dynamically request automatic initiation of a task associated with a terminal without passing data to the task.

For Assembler language:

```
MVC TCAICRT,=PL4'10000',      MOVE ONE HOUR TO TCA
MVC TCAICTI,=CL4'TRN1'        TRANSACTION ID TO TCA
MVC TCAICTID,=CL4'STA5'       TERMINAL ID TO TCA
.
.
.
DFHIC TYPE=INITIATE,          REQUEST TASK INITIATION          *
      INTRVAL=YES,            INTERVAL OF TIME GIVEN           *
      TRMIDNT=YES             TERMINAL ID GIVEN
MVC UNIQCODE,TCAICQID         SAVE CICS UNIQUE REQUEST ID
```

For ANS COBOL:

```
MOVE 10000 TO TCAICRT.        NOTE MOVE ONE HOUR TO TCA
MOVE 'TRN1' TO TCAICTI.       NOTE TRANSACTION ID TO TCA
MOVE 'STA5' TO TCAICTID.      NOTE TERMINAL ID TO TCA
.
.
.
DFHIC TYPE=INITIATE,          REQUEST TASK INITIATION          *
      INTRVAL=YES,            INTERVAL OF TIME GIVEN           *
      TRMIDNT=YES             TERMINAL ID GIVEN
MOVE TCAICQID TO UNIQCODE.    SAVE CICS UNIQUE REQUEST ID
```

For PL/I:

```
TCAICRT=10000;                /*MOVE ONE HOUR TO TCA*/
TCAICII='TRN1';               /*TRANSACTION ID TO TCA*/
TCAICTID='STA5';              /*TERMINAL ID TO TCA*/
```

```
.
.
.
DFHIC TYPE=INITIATE,            REQUEST TASK INITIATION          *
      INTRVAL=YES,              INTERVAL OF TIME GIVEN           *
      TRMIDNT=YES               TERMINAL ID GIVEN
UNIQCODE=TCAICQID;              SAVE CICS UNIQUE REQUEST ID
```

Task Initiation with Data (PUT)

Supported by CICS Temporary Storage Management, this facility allows the application programmer to pass data to another task that is to be initiated at some future time by issuing the

```
DFHIC TYPE=PUT,                                                  *
      INTRVAL=numeric value,YES,                                 *
      TIME=numeric value,YES,                                    *
      REQID=name,YES,                                            *
      TRANSID=name,                                              *
      TRMIDNT=name,YES,                                          *
      ICDADDR=symbolic address,YES,                              *
      NORESP=symbolic address,                                   *
      INVREQ=symbolic address,                                   *
      TRNIDER=symbolic address,                                  *
      TRMIDER=symbolic address,                                  *
      IOERROR=symbolic address
```

macro instruction. This macro instruction is used to provide the symbolic Transaction Identification, the location of the data to be stored, and other information applicable to the task to be initiated at some future time. CICS stores the data and queues the request until the specified interval of time has elapsed or the specified time of day has occurred. As soon as all necessary resources are available (for example, a terminal) the task is initiated.

The DFHIC TYPE=PUT macro instruction 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 the task to be initiated at some future time is associated with a terminal, the initial DFHIC TYPE=PUT request causes the task to be initiated at the specified time. Subsequent PUT's, with the same Terminal identification, Transaction Identification, and expiration time as the initial PUT, are used to store data for subsequent retrieval by the initiated task. (See the section "Retrieve Time-Ordered Data".)

If the task to be initiated at some future time 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 is passed to the initiated task. (See the section "Retrieve Time-Ordered Data".)

The application programmer must specify the parameters required in either of two ways: (1) by including the parameters in operands of the DFHIC TYPE=PUT macro instruction, or (2) by coding instructions, prior to issuing the DFHIC TYPE=PUT macro instruction, that dynamically move these parameters to fields in the TCA. If the parameters are included in operands of the DFHIC TYPE=PUT macro instruction, the applicable keywords are INTRVAL, TIME, REQID, TRANSID, TRMIDNT, and ICDADDR. (The keywords used to access user-written exception handling routines are discussed in the section "Test Response to a Request for Time Services".)

The numeric value specified in either the INTRVAL operand or TIME operand is used by CICS to calculate the time of day the requested time service is to be provided. If the calculated time of day is the same as the current clock time, or up to and including six hours preceding the current clock time, the specified time is considered to have elapsed (occurred) and the requested service is provided immediately. If the calculated time of day is in advance of the current clock time, the requested service is provided when the specified time occurs. If the calculated time of day precedes the current clock time by more than six hours, the requested service is provided the next day at the specified time.

Note: Users of CICS/OS must be aware that the current clock time is reset to zero each day to represent midnight. CICS makes no attempt to calculate a time of day based on a clock time less than zero.

INTRVAL: This operand is used to specify the interval of time after which the task is to be automatically initiated and/or data made available to the task in response to a DFHIC TYPE=PUT request. The interval of time is specified as a numeric value 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 DFHIC TYPE=PUT macro instruction is executed to calculate the time of day (clock time) at which the task is to be automatically initiated and/or data made available to the task. The minimum value that may be specified is one second.

The numeric value can be specified in the DFHIC TYPE=PUT macro instruction, or it can be dynamically moved to the TCAICRT field of the TCA prior to issuing the DFHIC TYPE=PUT macro instruction. In the latter case, the INTRVAL=YES operand must be included in the macro instruction.

The INTRVAL operand and TIME operand are mutually exclusive and may not be used in the same macro instruction.

TIME: This operand is used to specify the time of day at which the task is to be automatically initiated and/or data made available to the task in response to a DFHIC TYPE=PUT request. The time of day is expressed as a numeric value 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.

The numeric value can be specified in the DFHIC TYPE=PUT macro instruction, or it can be dynamically moved to the TCAICRT field of the TCA prior to issuing the DFHIC TYPE=PUT macro instruction. In the latter case, the TIME=YES operand must be included in the macro instruction.

The TIME operand and INTRVAL operand are mutually exclusive and may not be used in the same macro instruction.

REQID: Each time-ordered request has a unique Request Identification assigned to it. Its purpose is to provide a means of symbolically identifying the request and any data associated with it. Unless otherwise instructed, CICS generates a unique Request Identification.

The optional REQID operand allows the user to supply the unique Request Identification as part of the DFHIC TYPE=PUT service request in either of two ways: (1) by specifying a maximum of eight characters

in the REQID operand, or (2) by dynamically moving an eight-byte Request Identification to the TCAICQID field prior to issuing the DFHIC TYPE=PUT macro instruction. In the latter case, the REQID=YES operand must be included in the macro instruction.

If the REQID operand is omitted from the DFHIC TYPE=PUT macro instruction, the unique Request Identification generated by CICS is returned to the user in the TCAICQID field. The unique Request Identification becomes the symbolic name assigned to the data stored by CICS when servicing the DFHIC TYPE=PUT request.

TRANSID: This operand is used to supply the symbolic Transaction Identification of the task to be initiated at some future time. This operand can be omitted provided the application programmer has placed the symbolic Transaction Identification in the TCAICTI field prior to issuing the DFHIC TYPE=PUT macro instruction. CICS validates the symbolic Transaction Identification through a scan of the Program Control Table at the time of the initial macro request, providing a response code at TCAICTR (TCAICRC if the language is ANS COBOL) without servicing the request if it fails to locate a matching Transaction Identification.

TRMIDNT: This operand is used when the task to be initiated at some future time must communicate with a terminal. The symbolic Terminal Identification can be coded in the macro instruction, or can be dynamically moved to the TCAICTID field prior to issuing the DFHIC TYPE=PUT macro instruction. In the latter case, the TRMIDNT=YES operand must be included in the macro instruction.

CICS validates the symbolic Terminal Identification through a scan of the Terminal Control Table at the time of the initial macro request, providing a response code at TCAICTR (TCAICRC if the language is ANS COBOL) without servicing the request if it fails to locate a matching Terminal Identification. The TRMIDNT operand is omitted from the DFHIC TYPE=PUT macro instruction if no association with a terminal is required.

ICDADDR: This operand is used to supply the location of the data to be stored for the task that is to be initiated at some future time. The data must have the standard variable-length format, with the data length specified in the first four bytes (LLbb) followed by the data. 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 recommended to be a two-byte field of binary zeros. The symbolic address can be coded in the DFHIC TYPE=PUT macro instruction, or can be dynamically moved to the TCAICDA field prior to issuing the DFHIC TYPE=PUT macro instruction. In the latter case, the ICDADDR=YES operand must be included in the macro instruction.

The following is an example of the coding required to request automatic task initiation and/or request that time-ordered data be made available to a task associated with a terminal:

```
      DFHIC TYPE=PUT,                REQUEST TASK INITIATION        *
            TIME=173000,             TIME IS 5:30 PM                *
            TRANSID=TRN2,            TRANSACTION IDENTIFICATION     *
            TRMIDNT=STA3,            TERMINAL IDENTIFICATION        *
            ICDADDR=DATAFLD          DATA ADDRESS
```

The following are examples of the coding required to dynamically request automatic task initiation and/or request that time-ordered data be made available to a task associated with a terminal.

For Assembler language:

```
        MVC TCAICRT,PACKTIME            CALCULATED EXPIR TIME TO TCA
        MVC TCAICQID,UNIQCODE           UNIQUE REQUEST ID TO TCA
        MVC TCAICTI,=CL4'TRN2'          TRANSACTION ID TO TCA
        MVC TCAICTID,=CL4'STA3'         TERMINAL ID TO TCA
        MVC TCAICDA,=A(DATAFLD)         ADDRESS OF DATA TO TCA
        .
        .
        .
        DFHIC TYPE=PUT,                 REQUEST TASK INITIATION         *
              TIME=YES,                 EXPIRATION TIME GIVEN           *
              TRMIDNT=YES,              TERMINAL ID GIVEN               *
              REQID=YES,                UNIQUE REQUEST ID GIVEN         *
              ICDADDR=YES               DATA ADDRESS GIVEN
```

For ANS COBOL:

```
        MOVE PACKTIME TO TCAICRT.       NOTE CALC EXPIR TIME TO TCA
        MOVE UNIQCODE TO TCAICQID.      NOTE UNIQUE REQEST ID TO TCA.
        MOVE 'TRN2' TO TCAICTI.         NOTE TRANSACTION ID TO TCA.
        MOVE 'STA3' TO TCAICTID.        NOTE TERMINAL ID TO TCA.
        MOVE DATADDR TO TCAICDA.        NOTE ADDRESS OF DATA TO TCA.
        .
        .
        .
        DFHIC TYPE=PUT,                 REQUEST TASK INITIATION         *
              TIME=YES,                 EXPIRATION TIME GIVEN           *
              TRMIDNT=YES,              TERMINAL ID GIVEN               *
              REQID=YES,                UNIQUE REQUEST ID GIVEN         *
              ICDADDR=YES               DATA ADDRESS GIVEN
```

For PL/I:

```
        TCAICRT=PACKTIME;               /*CALC EXPIR TIME TO TCA*/
        TCAICQID=UNIQCODE;              /*UNIQUE REQUEST ID TO TCA*/
        TCAICTI='TRN2';                 /*TRANSACTION ID TO TCA*/
        TCAICTID='STA3';                /*TERMINAL ID TO TCA*/
        TCAICDA=ADDR(DATAFLD);          /*ADDRESS OF DATA TO TCA*/
        .
        .
        .
        DFHIC TYPE=PUT,                 REQUEST TASK INITIATION         *
              TIME=YES,                 EXPIRATION TIME GIVEN           *
              TRMIDNT=YES,              TERMINAL ID GIVEN               *
              REQID=YES,                UNIQUE REQUEST ID GIVEN         *
              ICDADDR=YES               DATA ADDRESS GIVEN
```

RETRIEVE TIME-ORDERED DATA (GET)

Tasks can retrieve expired time-ordered data by issuing the

```
        DFHIC TYPE=GET,                                                 *
              ICDADDR=symbolic address,YES,                             *
              NORESP=symbolic address,                                  *
              INVREQ=symbolic address,                                  *
              ENDDATA=symbolic address,                                 *
              NOTFND=symbolic address,                                  *
              IOERROR=symbolic address
```

macro instruction. This service is supported by the CICS Temporary Storage Management facility.

Only data from an expired DFHIC TYPE=PUT request can be accessed via the DFHIC TYPE=GET macro instruction. All data stored via the DFHIC TYPE=PUT macro instruction must be retrieved via the DFHIC TYPE=GET macro instruction.

Each originating DFHIC TYPE=PUT request symbolically identifies 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 request, 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 data record associated with the original request, and does so by issuing the DFHIC TYPE=GET macro instruction. An end-of-data condition occurs in response to the DFHIC TYPE=GET request when all data records have been retrieved for this task. 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).

A task associated with a terminal that is initiated as the result of an expired DFHIC TYPE=PUT request, or that is active on the terminal at the time of expiration of a DFHIC TYPE=PUT request, can access all data records associated with the other expired DFHIC TYPE=PUT macro requests, each having the same Transaction Identification, and Terminal Identification as the operating task. Therefore, a task associated with a terminal can retrieve all the expired data destined for the terminal and task by issuing consecutive DFHIC TYPE=GET requests. Each expired data record is presented to the task in expiration time sequence. The automatic task initiation request associated with each data record retrieved is canceled by the DFHIC TYPE=GET request for that data, and the storage occupied by the data is released at the same time.

CICS provides an end-of-data response at TCAICTR (TCAICRC if the language is ANS COBOL) when all expired data has been exhausted. CICS releases the storage occupied by the single data record associated with an expired DFHIC TYPE=PUT request, if that request also resulted in initiating the task and the task terminated (normally or abnormally) without retrieving the data. Subsequent expired DFHIC TYPE=PUT requests, specifying the same Terminal Identification and Transaction Identification, result in new tasks being initiated unless the data associated with the expired PUT request has been retrieved in response to a DFHIC TYPE=GET request.

The application programmer must specify the parameters required in either of two ways: (1) by including the parameter in operands of the DFHIC TYPE=PUT macro instruction, or (2) by coding the instruction, prior to issuing the DFHIC TYPE=PUT macro instruction, that dynamically moves the parameter to the field in the TCA. If the parameter is included in the operand of the DFHIC TYPE=PUT macro instruction, the applicable keyword is ICDADDR. (The keywords used to access user-written exception handling routines are discussed the section "Test Response to a Request for Time Services".)

ICDADDR: This optional operand is used to specify the location of the storage area provided by the user into which the data is to be placed. The symbolic address can be coded in the DFHIC TYPE=GET macro instruction, or it can be dynamically moved to the TCAICDA field prior to issuing the DFHIC TYPE=GET macro instruction. In the latter case,

the ICDADDR=YES operand must be included in the macro instruction. The storage area provided by the user must be large enough to contain the four-byte length field (LLbb) and data record. If this operand is omitted, a storage area is acquired by CICS that is large enough to contain the four-byte length field (LLbb) and data record. Its address is returned to the user in the TCAICDA field.

The following is an example of the coding required to request retrieval of a time-ordered data record:

```
DFHIC TYPE=GET,                    RETRIEVE TIME-ORDERED DATA      *
      ICDADDR=DATAFLD              USER-PROVIDED DATA AREA
```

The following are examples of the coding required to dynamically request retrieval of a time-ordered data record.

For Assembler language:

```
MVC TCAICDA,=A(DATAFLD)            DATA FIELD ADDR TO TCA
.
.
.
DFHIC TYPE=GET,                    RETRIEVE TIME-ORDERED DATA      *
      ICDADDR=YES                  DATA FIELD ADDRESS GIVEN
```

For ANS COBOL:

```
MOVE DATADDR TO TCAICDA.           NOTE DATA FIELD ADDR TO TCA.
.
.
.
DFHIC TYPE=GET,                    RETRIEVE TIME-ORDERED DATA      *
      ICDADDR=YES
```

For PL/I:

```
TCAICDA=ADDR(DATAFLD);             /*DATA FIELD ADDR TO TCA*/
.
.
.
DFHIC TYPE=GET,                    RETRIEVE TIME-ORDERED DATA      *
      ICDADDR=YES
```

TIME-ORDERED REQUEST CANCELLATION (CANCEL)

The application programmer can request that a previously issued time-ordered service request (DFHIC TYPE=WAIT, DFHIC TYPE=POST, DFHIC TYPE=INITIATE, or DFHIC TYPE=PUT) be canceled by issuing the

```
DFHIC TYPE=CANCEL,                                                 *
      REQID=name,YES,                                              *
      NORESP=symbolic address,                                     *
      INVREQ=symbolic address,                                     *
      NOTFND=symbolic address
```

macro instruction. The effect of the cancellation is dependent upon the presence or absence of the REQID operand in the DFHIC TYPE=CANCEL request and on the type of service request being canceled.

The keywords used to access user-written exception handling routines are discussed in the section "Test Response to a Request for Time Services".

REQID: This operand is required when identifying the unexpired time-ordered request issued by a task other than the cancelling task, and can be supplied by the application programmer in either of two ways: (1) by specifying the unique Request Identification in the REQID operand, or (2) by dynamically moving the unique Request Identification to the TCAICQID field prior to issuing the DFHIC TYPE=CANCEL macro instruction. In the latter case, the REQID=YES operand must be included in the macro instruction.

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 instruction.

When the originating task cancels a previously issued DFHIC TYPE=POST request, the REQID operand must be omitted from the cancellation request. The cancellation request can be made either before or after expiration of the original request. In either case, the storage occupied by the Timer Event Control Area is released and all reference to the original request is removed from the system.

When a task other than the originating task cancels a previously issued DFHIC TYPE=POST request, the REQID operand is required. The effect of the cancellation is the same as an early expiration of the original DFHIC TYPE=POST request. That is, the originating task's Timer Event Control Area 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 originating task (the originating task being suspended for the duration of the request). The REQID operand is required. The effect of the cancellation is the same as an early expiration of the original DFHIC TYPE=WAIT 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

Any cancellation of a previously issued DFHIC TYPE=INITIATE or DFHIC TYPE=PUT request requires that the REQID operand be included in the DFHIC TYPE=CANCEL macro instruction. 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.

INPUT/OUTPUT ERROR RETRY CAPABILITY (RETRY)

When the response to a previously issued DFHIC TYPE=GET macro instruction indicates an I/O error, the application programmer can issue the

```
DFHIC TYPE=RETRY,                                              *
      NORESP=symbolic address,                                 *
      INVREQ=symbolic address,                                 *
      NOTFND=symbolic address,                                 *
      IOERROR=symbolic address
```

macro instruction, requesting that CICS retry the retrieval operation. CICS attempts to perform the retrieval operation on the data record (whose symbolic eight-character identification is contained at TCAICQID) using the data area specified at TCAICDA. These fields are preset by CICS at the time the I/O error response was returned to the user.

The keywords used to access user-written exception handling routines are discussed in the section "Test Response to a Request for Time Services".

TEST RESPONSE TO A REQUEST FOR TIME SERVICES (CHECK)

One of the ways the application programmer can test the response to a request for time services is by issuing the

```
DFHIC TYPE=CHECK,                                              *
      NORESP=symbolic address,                                 *
      EXPIRD=symbolic address,                                 *
      IOERROR=symbolic address,                                *
      TRNIDER=symbolic address,                                *
      TRMIDER=symbolic address,                                *
      NOTFND=symbolic address,                                 *
      ENDDATA=symbolic address,                                *
      INVREQ=symbolic address
```

macro instruction. It provides for the testing of response codes and the routing of control to the appropriate user-written exception handling routines. This macro instruction provides an exception handling facility that can be used in the manner of a subroutine.

CICS automatically places the appropriate response code in the TCA at TCAICTR (TCAICRC if the language is ANS COBOL) after completion of the time service requested. The application programmer must specify the entry labels (symbolic addresses) he requires to access the appropriate exception handling routines previously supplied by the user.

If the application programmer does not use the DFHIC TYPE=CHECK macro instruction, he can specify the entry labels in either of two other ways: (1) by including the entry labels in operands of any other DFHIC macro instruction, or (2) by coding instructions immediately following the DFHIC macro instruction that examine the response code provided by CICS at TCAICTR (TCAICRC if the language is ANS COBOL) and transfer control to the appropriate routine.

The response codes are as follows:

CONDITION	ASSEMBLER	ANS COBOL	PL/I
NORESP	X'00'	12-0-1-8-9	00000000
ENDDATA	X'01'	12-1-9	00000001
IOERROR	X'04'	12-4-9	00000100
TRNIDER	X'11'	11-1-9	00010001
TRMIDER	X'12'	11-2-9	00010010
EXPIRD	X'20'	11-0-1-8-9	00100000
NOTFND	X'81'	12-11-0-1	10000001
INVREQ	X'FF'	12-11-0-7-8-9	11111111

If the TYPE=CHECK macro instruction is used by the application programmer, it should usually immediately follow another DFHIC macro instruction. The applicable keywords are NORESP, EXPIRD, IOERROR, TRNIDER, TRMIDER, NOTFND, ENDDATA, and INVREQ.

If the application programmer does not check for a particular response to his service request, and if that exception condition occurs, program flow proceeds to the next sequential instruction.

The operands that can be used to test the response to a request for time services are as follows.

NORESP: Specifies the entry label of the user-written routine to which control is to be passed in the event no errors occur. NORESP signifies "normal response" rather than "no response".

EXPIRD: Specifies the entry label of the user-written routine to which control is to be passed in the event the time specified in a DFHIC TYPE=POST or DFHIC TYPE=WAIT request had already expired at the time the request was issued.

IOERROR: Specifies the entry label of the user-written routine to which control is to be passed in the event an input/output error occurs during a DFHIC TYPE=GET or DFHIC TYPE=PUT operation on auxiliary storage. The DFHIC TYPE=RETRY macro instruction can be used in connection with the GET type of error handling routine.

TRNIDER: Specifies the entry label of the user-written routine to which control is to be passed in the event the symbolic Transaction Identification specified in the DFHIC TYPE=INITIATE or DFHIC TYPE=PUT request is not found in the Program Control Table (PCT).

TRMIDER: Specifies the entry label of the user-written routine to which control is to be passed in the event the symbolic Terminal Identification specified in the DFHIC TYPE=INITIATE or DFHIC TYPE=PUT request is not found in the Terminal Control Table (TCT).

NOTFND: Specifies the entry label of the user-written routine to which control is to be passed in the event the Request Identification specified in the DFHIC TYPE=CANCEL macro instruction 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 instruction cannot be located using the unique Request (data) Identification contained in TCAICQID at the time of this response.

For example, the "data not found" condition occurs on a retrieval operation if some prior task retrieved the data stored under that symbolic identification directly through Temporary Storage facilities and then released the data area. This condition also occurs if the Request Identification associated with the original DFHIC TYPE=PUT request fails to remain a unique identification.

ENDDATA: Specifies the entry label of the user-written routine to which control is to be passed in the event 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.

INVREQ: Specifies the entry label of the user-written routine to which control is to be passed in the event the operational CICS does not support the optional requested service. It may also indicate that an invalid type of request code was received for processing by the Interval Control program.

PROGRAMMABLE DEVICE CONSIDERATIONS

When BTAM is used by CICS for programmable 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 program then issues a write, CICS issues a read interrupt (RVI) to that line.

The programmable terminal response to a read interrupt must be "end of transmission" (EOT), except that the EOT response may be preceded by writes to exhaust the contents of output buffers so long as the input buffer size (specified previously by the user during preparation of the Terminal Control Table) is not exceeded by this data. CICS issues a read continue until it receives an EOT, or until the input message exceeds the size of the input buffer (which is an error condition).

After receiving an EOT, CICS issues a write initial (TI) or the equivalent (depending on the type of line). The programmable terminal response must be a read initial (TI) or the equivalent.

If another write is issued by the application program, CICS issues a write continue (TT) to that line. If the program issues a read after it has issued a write, CICS turns the line around; the program must have relinquished use of the line through a write reset (TR). (CICS does not recognize a read interrupt.)

To ensure that binary synchronous terminals (for example, System/360, 1130, 2780) remain coordinated, CICS processes the data collection or data transmission transaction on a line to completion before it polls the other terminals on that line.

Note: Since the CICS system service programs (Sign On/Sign Off, Master Terminal, etc.) do not insert binary synchronous control characters into the data stream, these programs cannot be used with the 2780 Data Transmission Terminal or the 2980 General Banking Terminal System.

The type of response required on the part of CICS and the user-written programmable terminal program to DFHTC macro instructions issued in a user-written application program is as follows:

APPLICATION PROGRAM	CICS (note 1)	TERMINAL PROGRAM
	READ INITIAL (TI)	WRITE INITIAL
DFHTC TYPE=READ	READ CONTINUE (TT)	WRITE CONTINUE
DFHTC TYPE=WRITE (note 2)	READ INTERRUPT (RVI)	WRITE RESET
(note 3)	READ CONTINUE (TT)	
	WRITE INITIAL (TT)	READ INITIAL
DFHTC TYPE=WRITE	WRITE CONTINUE (TT)	READ CONTINUE
DFHTC TYPE=READ (note 4)	WRITE RESET (TR)	READ CONTINUE
	READ INITIAL (TI)	WRITE INITIAL
DFHTC TYPE=RESET	WRITE RESET (TR)	

Note 1: CICS issues the equivalent of the macro instruction shown, depending upon whether the communication line is switched

or non-switched. The user-written programmable terminal program must issue the equivalent of the BTAM operation shown.

Note 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).

Note 3: The read continue is issued only if the "end of transmission" (EOT) character is not received on the read interrupt.

Note 4: Write reset is issued only for point-to-point terminals.

3735 CONSIDERATIONS

The 3735 Programmable Buffered Terminal is supported by CICS/OS and CICS/DOS-STANDARD as explained below.

3735 Transactions - 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; however, zero-length final blocks are not passed to the application program. If the block contains multiple logical records, the user's application 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 to ensure that all data has been captured before deblocking and processing that data.

The application must follow 3735 conventions and read to end-of-file before attempting to write FDP's (Form Description Programs) or data to the 3735. For this reason, the EOF=symbolic address operand must be used with each DFHTC TYPE=READ request. When the EOF branch is taken, the user may begin to write FDP's 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=symbolic address) before issuing any other Terminal Control requests in his program.

The user is responsible for formatting all special message headers for output to the 3735 (for example, SELECTRIC, POWERDOWN). If FDP's 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 instruction when he has exhausted the output to be 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.

3735 Transactions - Autocall

In automatic and time-initiated transactions, all of the considerations contained in the section "3735 Transactions - Autoanswer" apply when CICS dials a 3735 with the exception that the DFHTC EOF=symbolic address macro instruction is not used.

CICS connects the line and allows the user to indicate the direction of data transfer by means of his first Terminal Control request. The user is cautioned, however, that if his first request is a WRITE and the 3735 has data to send, the 3735 causes the line to be disconnected.

SYSTEM/7 CONSIDERATIONS

The implementation of System/7 support treats the System/7 as any other programmable terminal. Transactions are normally initiated from the System/7 by issuing a four-character transaction code as in the following example:

```
          .
          .
          .
        @XMIT   TRNID                   TRANSMIT TRANSACTION CODE
        PBER    ERROR                   BRANCH IF CONDITION ERROR CODE
        PLEX                            WAIT FOR COMPLETION
          .
          .
          .
TRNID   #IOLT   3,CHECK,/0000,TRAN,2
*       #IOLT                           GENERATE I/O LIST
*               3,                      RETURN CONTROL ON INTERRUPT
*                                       LEVEL 3
*               CHECK,                  RETURN CONTROL AT LOCATION CHECK
*               /0000,                  TRANSMIT MESSAGE IN BCD MODE
*               TRAN,                   MESSAGE LOCATED AT TRAN
*               2                       MESSAGE TWO WORDS LONG
TRAN    PEQU    *                       TRANSACTION ID
        PDC     /A6D2                   ='TR'
        PDC     /CA0E                   ='N7'
          .
          .
          .
CHECK   PEQU    *                       TEXT FOR SUCCESSFUL COMPLETION
          .
          .
          .
```

In this example, the transaction identification is transmitted in BCD mode. Pseudo-binary mode may only be used while communicating with an active CICS transaction; it can never be used to initiate the transaction. Note that the message length is given as the number of words to be transmitted and not as the number of characters.

When a transaction is initiated on a System/7, CICS services only that System/7 for the duration of the transaction; all other System/7's on that line are locked out for the duration of the transaction to provide most efficient use of the line. Therefore, it is highly recommended that CICS application programs be designed for the multipoint System/7 so that their execution is of short duration.

It is an MSP/7 standard that the first word (two characters) of every message received by the System/7 be an identification word. All identification words beginning with "@" (X'20') are reserved for future use by CICS.

When the PSEUDOBIN parameter is specified as part of an input request (for example, DFHTC TYPE=(READ,PSEUDOBIN)), the TIOA provided by the application program must be at least twice the length of the data to be read. For example, if 20 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 was specified. Therefore, on a DFHTC TYPE=(WRITE,READ,PDEUDOBIN) 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 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 then 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 specification.

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. As a result of this message transmission, a data check message may be recorded on the CICS (host) system console. This occurs if there is no program resident in the System/7 capable of supporting the Asynchronous Communication Control Adapter (ACCA) as is normally the case when the task to be initiated by CICS is to IPL the System/7. Although the BTAM error routines cause a data check message to be printed at the CICS console, CICS ignores this error and continues normal processing.

If a program capable of supporting the ACCA is resident in the System/7 at the time of this message transmission, the data check will not occur.

When a disconnect is issued to a dial-up System/7, the 'busy' bit is sometimes left on in the ACCA's interrupt status word. 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 every disconnect before message transmission is attempted. This may be accomplished by issuing a PIO instruction to reset the ACCA. The following instruction accomplishes this:

```
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).

NON-PROGRAMMABLE DEVICE CONSIDERATIONS

This section includes various considerations for the application programmer as he designs applications for non-programmable terminals.

2260/2265 PROGRAMMING CONSIDERATIONS

The following is an example of the coding required to write data to a 2260/2265 terminal and specify the screen line address where the write is to begin:

```
DFHTC TYPE=WRITE,            WRITE DATA TO A TERMINAL SCREEN    *
      LINEADR=10             STARTING AT THIS SCREEN LINE
```

The following are examples of the coding required to write data to a 2260/2265 terminal and dynamically determine the screen line address where the write is to begin.

For Assembler language:

```
MVI    TIOALAC,X'F0'        WRITE STARTING AT SCREEN LINE 1
         .
         .
         .
DFHTC TYPE=WRITE,           WRITE DATA TO A TERMINAL SCREEN    *
      LINEADR=YES           STARTING LINE ALREADY SPECIFIED
```

For ANS COBOL:

```
MOVE 240 TO TIOALAC.        NOTE PLACE STARTING LINE IN TIOA.
         .
         .
         .
DFHTC TYPE=WRITE,           WRITE DATA TO A TERMINAL SCREEN    *
      LINEADR=YES           STARTING LINE ALREADY SPECIFIED
```

For PL/I:

```
TIOALAC=240;                /*START WRITE AT SCREEN LINE 1*/
         .
         .
         .
DFHTC TYPE=WRITE,           WRITE DATA TO A TERMINAL SCREEN    *
      LINEADR=YES           STARTING LINE ALREADY SPECIFIED
```

2770/2780 PROGRAMMING CONSIDERATIONS

The 2770 Data Communication System and 2780 Data Transmission Terminal recognize a read interrupt and respond by transmitting the contents of the I/O buffer. After the contents of the buffer have been transmitted, the 2770 or 2780 responds to the next read continue with an EOT. If the I/O buffer is empty, the 2770 or 2780 transmits an EOT. CICS issues a read interrupt and read continue to relinquish use of the line and to enable the user to write to the 2770 or 2780.

Input from a 2770 or 2780 consists of one or more logical records. CICS provides the user with one logical record per read request. Note that the size of a logical record cannot exceed the contents of one buffer.

Output to a 2780 requires that the application programmer insert the appropriate "escape sequence" for component selection associated with the output message.

A read issued to a 2770 causes a logical record to be presented to the application program. If the input spans multiple buffers, multiple reads must be issued by the application program.

Passbook Control for the 2980

When writing application programs to service the 2980 General Banking Terminal System, the application programmer must be aware of the passbook control considerations described in this section.

Two one-byte fields of the Terminal Control Table terminal entry (TCTTE) may be interrogated by the application program while servicing passbook requests from the 2980. These fields are:

1. TCTTETAB, which contains the binary representation of the number of tabs necessary to position the print element at the passbook area.

2. 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 write operation, respectively. The same indicators are used to indicate the presence of the auditor key on the 2980 Model 2.

By testing indicators TCTTEPCR and TCTTEPCW, positive control can be maintained by the application program with regard to the absence or presence of a passbook during an update operation. However, care must be taken to never alter these indicators or 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). 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. However, the data sent to the terminal (and not printed because of the "no passbook" condition) is 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 generally write an error message to the journal area of the terminal informing the 2980 operator of this error condition. Then CICS automatically causes the transaction to wait for 23.5 seconds before continuing execution to allow the operator to insert the required passbook.

After regaining control from CICS following the writing of the error message, the application program can attempt another write to the passbook area after ensuring 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 specification of the correct number of tabs may be acquired from the field at TCTTETAB.

If the TCTTEPCW indicator is still off following the second attempt to write to the passbook area, the application program can send another error message or take some alternate action (for example, place the terminal "out of service").

In summary, all writes to the passbook area are conditional. That is, all writes require the presence of the passbook before they can be successfully executed. Therefore, a read operation cannot be combined with a passbook write. For example, a

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

macro instruction is an invalid request for 2980 terminal services involving the passbook area.

Note: The application programmer should not insert shift characters in output data since this is done automatically by CICS. CICS removes shift characters from input data.

Segmented Writes Control for the 2980

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 of the passbook as it spaces (indexes) past the bottom of the passbook.

For example, consider a 2982 buffer length of 48 and a 2980 Model 4 logical write (print) area of 100 characters per line. The user can write a logical record (DFHTC TYPE=PASSBK) of 100 characters to this area and it will be segmented by CICS because of buffer size. The user is required to insert the passbook indexing character (X'25') as the last character written in any 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 embeddded passbook index characters and the logical length of the message is such as to cause segmenting, the write terminates if the passbook spaces past the bottom of the passbook; the remaining segments are not printed.

Data Handling for the 2980

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 are 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 E.)

JOURNAL INDEXING: Journal indexing is the responsibility of the application programmer. Carriage returns (X'15') may be inserted anywhere in the logical message.

PASSBOOK INDEXING: Passbook indexing requires special consideration by the application programmer to control bottom line printing on the passbook. (See the section "Passbook Control for the 2980" and the section "Segmented Writes Control for the 2980".)

TAB CHARACTERS: The tab character (X'05') is also controlled by the application programmer. The number of tabs required to position the type element to the first position of the passbook is located in the TCTTE at TCTTETAB. This value is specified by the user 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 lite, openchute, and special banking characters can be used by the application programmer as needed. (See the 2980 Translate Tables in Appendix E.)

AUDITOR KEY MODEL 2: Presence of the Auditor key is controlled through use of the DFHTC TYPE=PASSBK macro instruction and may be used in a manner similar to that for passbook control. (See the section "Passbook Control for the 2980).

2980 MODEL NUMBER: The TCTTETM field of the Terminal Control Table terminal entry (TCTTE) contains the 2980 model number expressed as a hexadecimal value (X'01', X'02', X'04'). CICS uses the model number to select the correct Translate Table for each of the 2980 models; therefore, the user cannot 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.

Writing High-Level Language Programs for the 2980

The high-level language application programmer must concern himself with the following fields of the DFHTCTTE structure when writing programs to run on a 2980 General Banking Terminal System:

FIELD	MEANING
TCTTETAB	Number of tab characters (binary)
TCTTEPCF	Passbook control field
TCTTESID	Station identification
TCTTETID	Model 4 teller identification

This section discusses one way to manipulate these fields.

As discussed in the section "Passbook Control for the 2980", the application programmer is expected to read TCTTETAB to determine the correct number of tab characters to place in his output data. The following examples show how this might be done in ANS COBOL and PL/I programs, respectively.

For ANS COBOL:

```
DATA DIVISION.
WORKING-STORAGE SECTION.
01 DFH2980 COPY DFH2980.
   .
   .
   .
LINKAGE SECTION.
01 DFHBLLDS COPY DFHBLLDS.
   02  TCTTEAR PICTURE S9(8) USAGE IS COMPUTATIONAL.
   02  TIOABAR PICTURE S9(8) USAGE IS COMPUTATIONAL.
   .
   .
   .
```

```
01 DFHTCTTE COPY DFHTCTTE.
01 DFHTIOA COPY DFHTIOA.
   02  DATA PICTURE X(20).
   02  FILLER REDEFINES DATA.
     03  TAB1-1 PICTURE X.
     03  DATA1 PICTURE X(19).
   02  FILLER REDEFINES DATA.
     03  TAB1-2 PICTURE X.
     03  TAB2-2 PICTURE X.
     03  DATA2 PICTURE 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.
   .
   .
   .
```

For PL/I:

```
%INCLUDE DFHTIOA;
       2 DATA CHAR (20);
DECLARE 1 USERTIOA_1 BASED (TIOABAR),
       2 TIOAFILL CHAR (12),
       2 TAB1_1 CHAR (1),
       2 DATA1 CHAR (19);
DECLARE 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 ONETBCH;
       IF (TCTTETAB = TAB_TWO) THEN GO TO TWOTBCH;
       .
       .
       .
ONETBCH:   TAB1_1 = TABCHAR;
           DATA1 = AMOUNT;
           .
           .
```

```
TWOTBCH:   TAB1_2 = TABCHAR;
           TAB2_2 = TABCHAR;
           DATA2 = AMOUNT;
           .
           .
           .
```

In the ANS COBOL example, the structure DFH2980 is copied in the Working Storage Section; in the PL/I example, DFH2980 is included following the INCLUDE statements for the based structures. DFH2980 contains constants that may be used when writing application programs for the 2980.

The application programmer is also expected to test the TCTTEPCF field to determine whether there was a passbook present on a read or write. TCTTEPCR and TCTTEPCW are located in DFH2980 to aid in this testing.

To test the TCTTEPCF field in ANS COBOL, 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 TCTTEPCF allows the ANS COBOL programmer to test for the presence of a passbook on a read. (HOLDPCF and HOLDPCFB are also part of DFH2980.)

To test the TCTTEPCF field in PL/I, statements such as the following might be used:

```
IF (TCTTEPCF | TCTTEPCW) THEN GO TO
BOCK-PRESENT-WRITE;
```

Substituting TCTTEPCR for TCTTEPCF allows the PL/I programmer to test for the presence of a passbook on a read.

To test the station identification and to determine whether the normal station or alternate station is being used, values are predefined in DFH2980 of the form:

```
STATION-#-A  OR  STATION-#-N  (for ANS COBOL)

STATION_#_A  OR  STATION_#_N  (for PL/I)
```

where # is any 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; therefore it can also be tested in an IF statement.

Twenty-three special characters are defined in DFH2980 that may be referenced by the name SPECCHAR-X (ANS COBOL and PL/I) where "X" is an integer (0 through 23). Seven other characters are defined with names which imply their usage; for example, TABCHAR. For further information on these 30 characters, see Appendix E.

Following are the names defined in DFH2980 for ANS COBOL:

STATION-0-N	STATION-6-A	TAB-SIX	MSGLITE	SPECCHAR-11
STATION-0-A	STATION-7-N	TAB-SEVEN	BCKSPACE	SPECCHAR-12
STATION-1-N	STATION-7-A	TAB-EIGHT	TRNPGE	SPECCHAR-13
STATION-1-A	STATION-8-N	TAB-NINE	SPECCHAR-1	SPECCHAR-14
STATION-2-N	STATION-8-A	HOLDPCFB	SPECCHAR-2	SPECCHAR-15
STATION-2-A	STATION-9-N	DFHFILL	SPECCHAR-3	SPECCHAR-16
STATION-3-N	STATION-9-A	HOLDPCF	SPECCHAR-4	SPECCHAR-17
STATION-3-A	TAB-ZERO	TCTTEPCR	SPECCHAR-5	SPECCHAR-18
STATION-4-N	TAB-ONE	TCTTEPCW	SPECCHAR-6	SPECCHAR-19
STATION-4-A	TAB-TWO	TABCHAR	SPECCHAR-7	SPECCHAR-20
STATION-5-N	TAB-THREE	OPENCH	SPECCHAR-8	SPECCHAR-21
STATION-5-A	TAB-FOUR	JRNLCR	SPECCHAR-9	SPECCHAR-22
STATION-6-N	TAB-FIVE	PSBKCR	SPECCHAR-10	SPECCHAR-23

Following are the names defined in DFH2980 for PL/I:

STATION_0_N	STATION_7_A	TCCTEPCR	SPECCHAR_7	SPECCHAR_22
STATION_0_A	STATION_8_N	TCTTEPCW	SPECCHAR_8	SPECCHAR_23
STATION_1_N	STATION_8_A	TABCHAR	SPECCHAR_9	
STATION_1_A	STATION_9_N	OPENCH	SPECCHAR_10	
STATION_2_N	STATION_9_A	JRNLCR	SPECCHAR_11	
STATION_2_A	TAB_ZERO	PSBKCR	SPECCHAR_12	
STATION_3_N	TAB_ONE	MSGLITE	SPECCHAR_13	
STATION_3_A	TAB_TWO	BCKSPACE	SPECCHAR_14	
STATION_4_N	TAB_THREE	TRNPGE	SPECCHAR_15	
STATION_4_A	TAB_FOUR	SPECCHAR_1	SPECCHAR_16	
STATION_5_N	TAB_FIVE	SPECCHAR_2	SPECCHAR_17	
STATION_5_A	TAB_SIX	SPECCHAR_3	SPECCHAR_18	
STATION_6_N	TAB_SEVEN	SPECCHAR_4	SPECCHAR_19	
STATION_6_A	TAB_EIGHT	SPECCHAR_5	SPECCHAR_20	
STATION_7_N	TAB_NINE	SPECCHAR_6	SPECCHAR_21	

7770 PROGRAMMING CONSIDERATIONS

Even though CICS does not distinguish between any of the special codes (characters) that may be 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 Portable Audio Terminal:

KEY	CODE
CALL END	37**
CNCL	18
#	3B** or 7B
VERIFY	2D
RPT	3D
EXEC	26**
F1	B1
F2	B2
F3	B3
F4	B4
F5	B5
00	A0
000	3B** or B0
IDENT	11, 12, 13, or 14 plus two other characters

For further information concerning the 2721, see the publication 2721 Portable Audio Terminal Component Description (GA27-3029).

The following special hexadecimal codes may be entered from a Touch-Tone[1] telephone:

KEY	CODE
*	A0
#	3B** or B0

---

[1]Trademark of the American Telephone & Telegraph Co.

The * and # characters of a Touch-Tone telephone correspond to the 00 and 000 characters, respectively, on a 2721 Portable Audio Terminal.

The above codes denoted by a double asterisk cause a hardware interrupt and are in the Terminal I/O Area (TIOA) immediately following the data; the codes are not included in the data length.

Note: The # and 000 characters cause an EOI (X'3B') hardware interrupt unless the EOI Disable feature (#3540) is installed on the 7770 Audio Response Unit Model 3. In this case, at the option of the user, either or both of the # and 000 characters do not cause a hardware interrupt and are presented in the TIOA along with the rest of 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 "end of inquiry" (EOI) hardware interrupt that ends the read operation.

CREATING USER EXITS FOR ASYNCHRONOUS TRANSACTION PROCESSING

If the Asynchronous Transaction Processing facility is used, the Input Processor (CRDR) and the Output Processor (CWTR) are employed to transfer data to and from CICS. The two programs accomplish the transfer of data without regard to its content. For example, any terminal-dependent characters in an output stream must have been put there by the user's transaction.

However, it may be desirable to perform some preprocessing or postprocessing on the terminal data. Such processing might be for purposes of:

1. Validity and limit checking
2. Removing or inserting device dependencies
3. Summarizing or formatting
4. Provide additional communication with CICS

These and other services can be accomplished through the use of the user exits provided by CRDR and CWTR. During data input to CICS, CRDR offers each transmitted record to a user-written exit routine immediately after it is received. CWTR offers each record to a user-written exit routine immediately after it has been deblocked from its Transient Data input area.

All records are made available to the user routine, including delimiter records.

The exit routine is invoked by specifying its program name suffix in the CRDR or CWTR initiating the message. For example:

```
CRDR EXIT=MD,NAME=WICHITA
```

causes CRDR to load the program named DFHXITMD (where DFHXIT is the standard exit routine base name and MD is the suffix) and pass each record to that routine while building a batch named WICHITA.

Similarly, the statement:

```
CWTR NAME=FINDLAY,TERMID=(TMLA,TMLB,TMLC),EXIT=DI
```

causes CWTR to load the program DFHXITDI and pass each output record (associated with the output of batch FINDLAY) to the routine before it is transmitted to the terminal.

One additional point should be noted concerning records given to the CWTR exit routine. The messages being sent in response to a STATUS request are passed to the routine. For example:

```
CWTR NAME=SUNYVALE,STATUS,EXIT=CN
```

causes the message concerning the status of a batch named SUNYVALE to be passed to DFHXITCN. This permits the user-written exit routine to augment the status message. All CICS service macro instructions may be used in the exit programs.

CODING THE CRDR EXIT ROUTINE

The CRDR Input Processor uses the following basic TCA work area definitions:

```
            COPY    DFHTCADS
TWAREC      DS      A                   Address of record to be inserted
TWAWA       DS      A                   Address of user work area
TWAIND      DS      X                   Indicators
TWAXTRTN    EQU     X'80'               Exit program return indicator
            DS      3X                  Reserved
            DS      20F                 Reserved
```

These fields (plus any additional fields) should be defined by the user-written exit routine within the limits specified in the PCT entry. Information is passed between CRDR and the exit routine by means of this TCA work area.

Upon initial entry to the exit routine TWAWA and the TWAXTRTN bit are zero. On all entries, TWAREC is zero. All modification of the TWAXTRTN bit must be done either by the instruction OI TWAIND,TWAXTRTN or the instruction NI TWAIND,255-TWAXTRTN. The user exit must take care not to modify the bits in the TWAIND field used by CWTR.

On all entries to the exit routine, register contents are:

REGISTER	CONTENTS
15	Exit routine entry address
14	Exit routine return address
13	CSA address
12	TCA address
8	TIOA address of last message read
7	BCA address

The only registers that cannot be used in the routine are registers 12 and 13. The other registers are saved before exiting and are restored by CRDR upon return. The Batch Control Area (BCA) is described in the symbolic storage definition named DFHBCADS.

The exit routine must be enterable at two points. The first entry is for routine initialization and is made via an Assembler BALR 14,15 instruction. This is done only once so that turning on the TWAXTRTN bit does not cause a reentry to occur. The message in the TIOA is the CRDR transaction-invoking message.

All subsequent entries to the exit routine are made via an Assembler BAL 14,4(15) instruction. This entry is made after each message is read.

The exit routine entry coding might appear as follows:

```
DFHXITAB   CSECT
           USING   *,15
           B       INIT
           B       MSGP
           DROP    15
           USING   DFHXITAB,10
           .
           .
           .
INIT       LR      10,15
           .
           .
           .
MSGP       LR      10,15
           .
           .
           .
```

If the record just read is to be accepted without change or is to have its contents altered, it can be done in the TIOA and return made to CRDR via a BR 14 instruction. TWAREC and the TWAXTRTN bit should remain zero.

If the length of the record just read is to be changed, it can be done in the TIOA by altering the TIOATDL field. TWAREC and the TWAXTRTN bit should be zero. If the record is to be lengthened such that it won't fit in the TIOA, the record must be built in a user-defined work area as a standard variable-length record. (The record in the TIOA is not a standard VLR since the value in TIOATDL is four less than a VLR count.) The address of the count field (LLbb) is then put into TWAREC and control is returned to CRDR.

When the exit routine once again gains control, TWAREC is zero and a new message is in the TIOA. A work area used to alter records may be defined in the TCA work area or acquired dynamically through use of a CICS DFHSC TYPE=GETMAIN macro instruction. If acquired dynamically, its address may be stored at TWAWA.

To insert records into the input stream, each new record must be built in an exit routine work area, its address placed at TWAREC, the TWAXTRTN bit set on, and control returned to CRDR. The new record is inserted and control is returned to the exit routine with TWAREC set to zero and the TWAXTRTN bit left unchanged. After all new records have been inserted in this manner, the TWAXTRTN bit must be set to zero and control returned to CRDR with TWAREC containing zero. The original message in the TIOA is placed into the input stream and a new message is read from the terminal.

If the original message in the TIOA is to be deleted, control must be returned to CRDR with TWAREC containing the address of F'0'.

CODING THE CWTR EXIT ROUTINE

The CWTR Output Processor uses the following basic TCA work area definitions:

```
            COPY    DFHTCADS
TWANXREC    DS      A
TWAREC      DS      A
TWAWA       DS      A
TWAIND      DS      X
TWAXTRTN    EQU     X'80'
            DS      3X
            DS      30F
```

These fields (plus any additional fields) should be defined by the user-written exit routine within the limits specified in the PCT entry. Information is passed between CWTR and the exit routine by means of the TCA work area.

Upon initial entry to the exit routine, TWAWA and the TWAXTRTN bit are zero. On all entries TWAREC is zero, and TWANXREC points to the variable-length record which is to be transmitted to the output terminal. Any modification of the TWAXTRTN bit must be done on a bit level since other bits in TWAIND are used by CWTR.

The first four bytes of a variable-length record contain a two-byte length field and, occasionally, two bytes of control information. In the case of the record to be handled by CWTR, the first of these two control bytes (byte three of the record) contains the byte that would ordinarily be moved to TCTTEOS by the DFHTC macro instruction. The second control byte (byte four of the record) applies only to records that are destined for a 2260 Display Station or a 3270 Information Display System; this control byte corresponds to the TIOALAC or TIOACLCR field. If the destination terminal is a 3270 and the TIOACLCR field is not applicable, X'C3' (the default value) must be moved into this control byte.

If the length of the record is to be changed, the two control bytes probably are not affected and the information from the original record can be used. However, building a new record requires that one or both of these control bytes be properly constructed.

On all entries to the exit routine, register contents are:

REGISTER	CONTENTS
15	Exit routine entry address
14	Exit routine return address
13	CSA address
12	TCA address
7	BCA address

The only registers that cannot be used in the routine are registers 12 and 13. The other registers are saved before exiting and restored by CWTR upon return.

The exit routine must be enterable at two points. The first entry is for routine initialization and is made via an Assembler BALR 14,15 instruction. This is done only once so that turning on the TWAXTRTN bit does not cause a reentry to occur. Also, there is no message located by TWANXREC.

All subsequent entries to the exit routine are made via an Assembler BAL 14,4(15) instruction. This entry is made after each message is deblocked and is about to be transmitted.

The exit routine entry coding might appear as follows:

```
DFHXITAB   CSECT
           USING   *,15
           B       INIT
           B       MSGP
           DROP    15
           USING   DFHXITAB,10
           .
           .
           .
INIT       LR      10,15
           .
           .
           .
MSGP       LR      10,15
           .
           .
```

If the record about to be written is to be accepted without change or is to have only its contents altered, it can be done in its current area located by TWANXREC. Return to CWTR is made with a BR 14 instruction; TWAREC and the TWAXTRTN bit should be zero.

If the length of the record is to be altered, it must be done by replacing the record located by TWANXREC with the altered record. The altered record must be built in an exit routine work area as a standard variable-length record. The address of the new record must be put into TWAREC and control returned to CWTR. The new, altered record replaces the old record. When the exit routine once again gains control, TWAREC is zero and a new message is located by TWANXREC.

If the new record just described is to be inserted into the output stream in addition to the record at TWANXREC, the TWAXTRTN bit must be set to one prior to returning to CWTR. The new record (pointed to by TWAREC) is sent to the terminal and control is returned to the exit routine with TWANXREC pointing to the original record; TWAREC is zero. This permits the exit routine to continue inserting records into the output stream until return to CWTR is made with the TWAXTRTN bit and TWAREC set to zero.

Deleting a record can be done by returning control to CWTR with TWAREC containing the address of F'0'.

If dynamic storage is required by the exit routine, it can be acquired from Storage Control and saved by putting its address in TWAWA.

DATA BASE CONSIDERATIONS

SEGMENTED RECORDS

An optional feature of CICS File Management allows the user to create and define a data set containing segmented records. A segmented record is one in which the components of the record have been identified (symbolically) and grouped according to some logical relationship such as function or frequency of use.

The identifiable groups are called segments. A segment is one or more adjacent fields within a record. Some segments appear in all records (for example, those segments containing identification or major record control fields), while other segments apply only to, and appear in, certain records. Before the application programmer can use

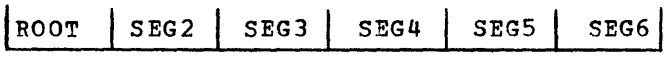
segmented records in his program, the structure and individual segments of a segmented data set must have been previously defined by the user in the File Control Table.

Segmented records offer numerous advantages. Having organized and defined the segments of a data set, the user can group them into segment sets and retrieve any set (or group) of segments by symbolically identifying that set. Since an individual segment can be a member of any number of segment sets, the user gains a high degree of flexibility in the retrieval process. Because only a part (a segment set) of a logical record is requested, CICS can extract just the requested segments, pass them to the processing program, and free the main storage required for the entire logical record or block at the earliest possible time.

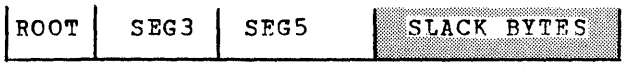
A saving in DASD space can be realized when segmenting is used with variable-length record format, since CICS File Management always compresses (packs) a segmented record before writing it to direct access. The space normally required for missing segments is thus eliminated, as are the slack bytes created when aligning segments in main storage.

With fixed-length records, compression causes the unused space to be consolidated at the end of the record. For example:

- Logical record as defined by the user in the File Control Table



- Logical record as it appears on DASD with missing segments



The following general rules apply to the use of segmented records:

1. Segmented records can be used with either ISAM or DAM organized data sets.
2. Segmented records can be used with any record format (that is, fixed, fixed blocked, variable, undefined) but are primarily advantageous with variable-length records.
3. A data set that contains segmented records may not also be an index data set in an indirect accessing hierarchy. The two CICS features are mutually exclusive for any one data set. However, the primary (target) data set in an indirect accessing hierarchy may contain segmented records.
4. Every segment that could appear in a record, whether or not it actually exists in a particular record, must be defined in the File Control Table.

Segmented Record Formats

It is the user's responsibility to describe, for each segmented data set, all segments within a logical record. Each segmented data set is first described in the File Control Table just as any other data set. That is, its basic characteristics must be described so that CICS File Management can physically access it (for example, block size, logical record length, key length, etc.). As an addendum to this basic data set descriptive section, the user must describe the segmented structure of the data set.

Every segment (any number of adjacent bytes up to a maximum of 255) must be defined, even if it does not exist in every logical record.

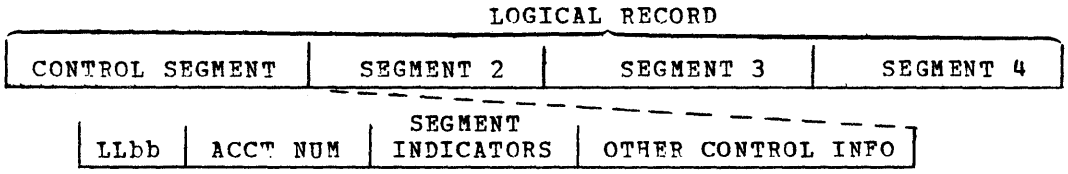
While it is not required that every logical record contain every segment, every logical record must contain at least the root (control) segment.

The root segment is a uniquely defined segment that must appear at the beginning of each logical record. It contains as a minimum:

1. The length of the record, if variable-length records are being used. This is a fullword (four bytes) of the form LLbb, where LL is the record length and bb is two bytes reserved for system use.
2. Segment indicators, which indicate the presence or absence of each segment in the record. Segment indicators are discussed in greater detail below.

In addition, the root segment could contain any other information that might aid in the processing of the record by the user (for example, a major control field such as an account number).

The following is an example of a segmented record and the root (control) segment.



The sequence of the segments within a logical record must be fixed. That is, a segment may not change position in relation to the other segments of the record. Each segment can be fixed or variable in length. If the segment is variable in length, then the first byte must contain the length, in binary, of the segment, not including the length byte. Thus the maximum data length of a variable-length segment is 254 bytes instead of 255. The number of bytes in a fixed-length segment or the maximum length of a variable-length segment is supplied to CICS File Management as part of the segment definitions in the File Control Table.

Each segment has its own characteristics and these can be different from other segment definitions. Each segment can have a different length than other segments, and, if defined as variable length, can change as a result of an update. Segments may be added or deleted; CICS File Management compresses and expands the record accordingly.

CICS File Management provides for the user to specify the alignment requirements of each segment when that segment is brought into main storage. This alignment may be on a one-byte, two-byte, four-byte, or eight-byte boundary. The default alignment is on a one-byte boundary. When the segmented record is written to direct access, any residual space (slack bytes) caused by alignment is eliminated by CICS File Management through the compress (packing) function.

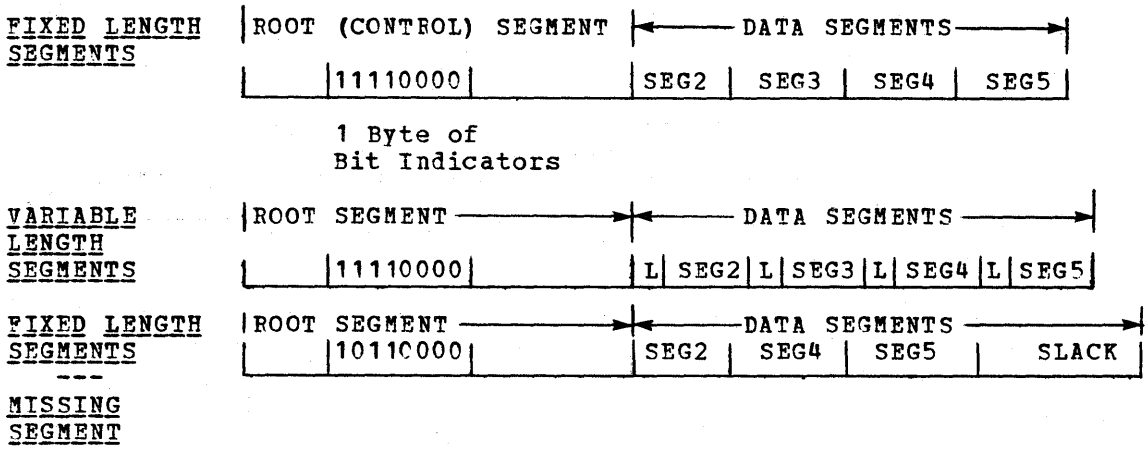
Segment Indicators

Segment indicators are the means by which CICS File Management and the processing program specify, and determine, the presence or absence of specific segments within a logical record. There are two types of indicators available to the user; it is his responsibility to choose the type he wishes to use and to define his data sets accordingly. Regardless of the type of indicator, the following general rules apply to the use of segment indicators in processing the segmented record:

1. Segment indicators are always located in contiguous bytes within the root (control) segment. Note that every logical record contains a root segment and that the root segment is always a part of any segment set brought into main storage. Therefore, the segment indicators are always accessible to the user.
2. The location of the indicators within the root segment is defined by the user in the File Control Table as being some displacement from the beginning of the root segment.
3. There must be one indicator for each segment which is defined, other than the root segment. The position of the indicator determines which segment it represents. Since the root segment does not require an indicator, the first indicator represents the first segment following the root segment (segment 2), the second indicator represents the second segment following the root segment (segment 3), etc.
4. When retrieving segment sets, it is the user's responsibility to test the appropriate indicator to determine if a specific segment is present. He should never assume a segment is present simply because it was requested as part of a segment set.
5. When adding or deleting segments from a record, it is the user's responsibility to reset the appropriate indicator to reflect the change.

BIT TYPE SEGMENT INDICATORS: With the bit type indicator, each segment is represented by a bit position in the segment indicator field. One byte of indicators must be provided within the root segment for each eight segments in the logical record. If a given bit indicator is on (binary 1), the corresponding segment is present in the logical record.

If a given bit indicator is off (binary 0), the corresponding segment is absent from the logical record. The following are examples of bit type segment indicators:



DISPLACEMENT TYPE SEGMENT INDICATORS: With the displacement type indicator, each segment is represented by one halfword (two-bytes) in the segment indicator field. In any given halfword indicator, a value of zero indicates the corresponding segment is absent from the logical record. A nonzero value (binary) in any given halfword indicates that the corresponding segment is present and represents the displacement of the segment from the beginning of the logical record when the segments are packed.

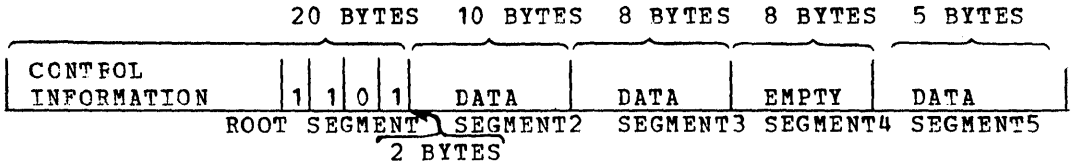
Any displacement value which is placed in the halfword indicators when building a new record or adding and deleting segments from an

existing record, may be modified by CICS File Management when it compresses (packs) the segments before writing the record to direct access. Whenever CICS packs segmented records, it places the displacement value of each segment into the corresponding halfword indicator (if displacement type indicators are being used). However, CICS File Management does not change these displacement values when unpacking a segmented record or when extracting selected segments of a segment set.

The user should not rely on the displacement values in order to access segments he has retrieved in a segment set; he should only use them as zero/nonzero indicators to determine whether or not a requested segment is present. (See "Main Storage Processing of Segmented Records".)

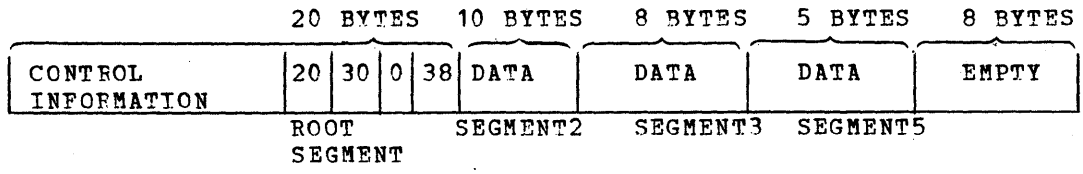
The following example illustrates the basic concepts and considerations when using displacement type segment indicators.

1. The following is the segmented record built by the user in main storage which is to be added to a segmented data set:

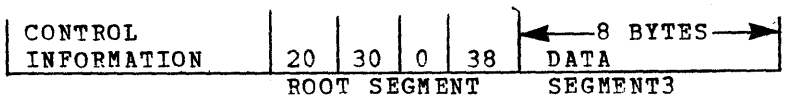


   The user has placed data in three of the four defined segments and indicated their presence by placing a nonzero value in the corresponding halfword displacement indicators. Any nonzero value may be used (the 1 is only an example).

2. Before writing the record to the direct access data set, CICS File Management compresses the segments and modifies the displacement indicators so that the above record would appear as follows before being written to DASD:



3. When retrieving a segment set from the above record, the root segment is included as part of the segment set without any modification. If the user were to request a segment set from the above record (consisting of the Root Segment and Segment3), the data he would receive might appear as follows:



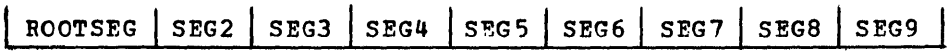
Main Storage Processing Of Segmented Records

When a segment set is requested from a segmented data set, the data is always placed into a File Work Area (FWA). The length of this FWA is variable depending upon the segments retrieved and their attributes. However, it is not the users responsibility to determine this length since CICS File Management automatically calculates it and acquires the FWA through CICS Storage Management. A CICS-provided symbolic storage definition (DFHFWADS) can be used in conjunction with a user-defined layout to map the FWA.

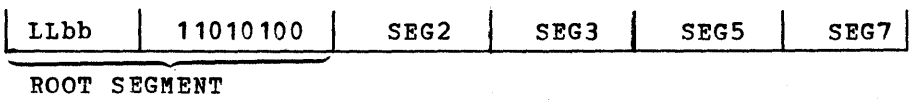
The FWA consists of control fields (used by CICS Management functions) and a data area into which the requested segments are placed by File Management. The format of the retrieved segments within the data portion of the FWA is always in a fixed format. That is, space is provided in the FWA and alignment requirements are met for each segment in the requested segment set, even though a segment may be missing. (For variable-length segments, the maximum space is provided.) It is the user's responsibility to test the appropriate segment indicators to determine the presence or absence of a segment. Note that an update request on a segmented data set causes CICS File Management to automatically use the universal segment set "ALL" when retrieving the record.

The following illustrations should help clarify the various considerations discussed thus far concerning main storage processing of segmented records.

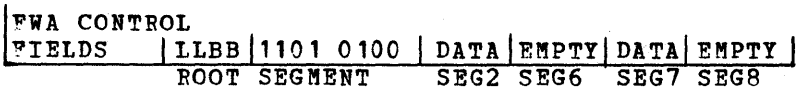
1. Logical record as defined by the user in the File Control Table:



2. Logical record as it appears on DASD. Assume variable-length records and bit type segment indicators:

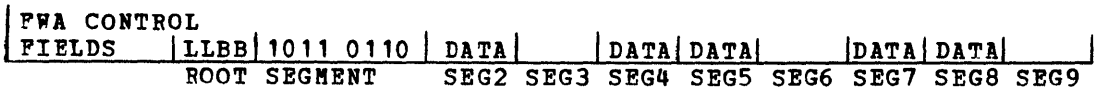


3. Logical record as it appears in the FWA after retrieval of a segment set (read-only) which included Root Segment, SEG2, SEG6, SEG7, SEG8:

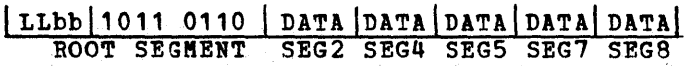


4. Logical record as it appears in the FWA after a retrieval for update (SEGSET=ALL):

5. Logical record as it appears in the FWA after the user has added segments 4 and 8 and deleted segment 3. The indicators have been adjusted by the user to reflect the change.



6. Logical record as it appears on DASD after packing:



Segment Sets

Once each segment has been defined (name and attributes specified), the user can specify as many segment sets as he desires. A segment set is a grouping of the root segment and at least one or more individual segments. Like the individual segments, the segment set is given a symbolic name which is used by the application program when processing a segmented data set. Any retrieval from a segmented data set is always by segment set.

Assume a logical record in a segmented data set has been defined as containing the following symbolic segments:

ROOTSEG
SEGMENT2
SEGMENT3
SEGMENT4
SEGMENT5
SEGMENT6

The user might wish to define the following segment sets:

SEGMENT SET NAME	SEGMENTS
SEGSETA	ROOTSEG
	SEGMENT2
	SEGMENT4
SEGSETB	ROOTSEG
	SEGMENT3
	SEGMENT4
	SEGMENT5

Whenever a segmented data set is defined in the File Control Table, a universal segment set is automatically generated which includes all segments defined for that data set. The symbolic identification of this universal segment set is "ALL", and is automatically used by CICS File Management whenever the application program requests a "read for update" from a segmented data set. In other words, an update operation on a segmented data set always causes all segments to be presented to the user, regardless of the segment set specified by the user.

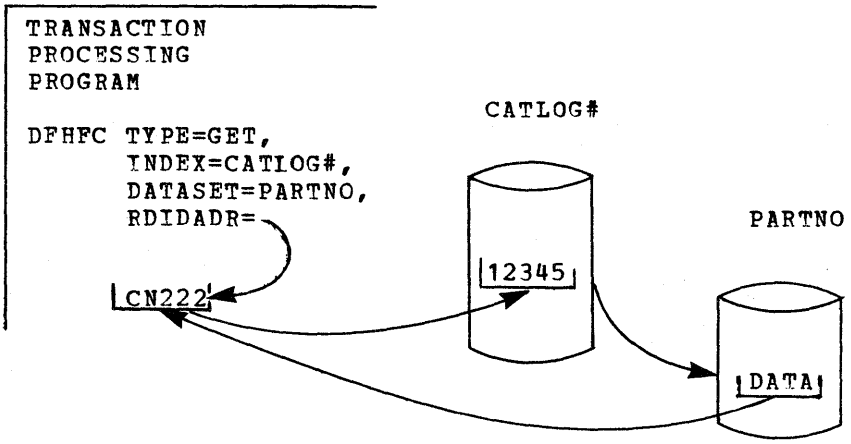
INDIRECT ACCESSING

Indirect accessing, an optional data base feature in CICS, provides for the use of cross-index data sets to access another data set. The data set that is accessed by an index data set is known as the "primary" or "target" data set. This feature allows the user to furnish the search argument for an index data set along with the identification of the primary data set. CICS, utilizing the user-defined index structure, carries out the search, involving as many levels (index data sets) as defined by the user, and ultimately retrieves the prime data required.

The following general rules apply to the Indirect Accessing feature:

1. A primary data set can have any number of index data sets. This is useful when multiple cross references to a master record exist.
2. Any data set can be both an index and a primary data set. The logical record content of any data base data set is user-defined and constructed, and therefore may contain certain master record information as well as a search argument for another data set.
3. There is no logical limit to the number of index levels (data sets) that the user may define in an index hierarchy. For example, data set A is an index to data set B which is an index to data set C which is an index to data set D, etc.
4. An indirect access hierarchy can be any combination of ISAM and DAM data sets.
5. An index data set may not also contain segmented records. The two CICS services are mutually exclusive for any one data set. However, a primary data set, which an index data set accesses, could have segmented records if it were not defined also as an index data set.
6. An index data set cannot reference more than one primary data set unless the index data set is multiply defined in the File Control Table.
7. If the index data set is a DAM data set, it may not be defined as blocked. However, the primary data set may be defined as blocked BDAM.

The following is an example of a simple two-level indirect access hierarchy. The retrieval search begins with the index data set CATLOG#. The primary data set being accessed (and from which data is to be returned to the requesting program) is PARTNO. The search argument to be used in accessing the index data set (CATLOG#) is CN222. The contents of the record located by the search of the index data set (CATLOG#) contains the search argument for the next data set (12345 for search of PARTNO). The primary data set (PARTNO) is searched and the data record returned to the requesting program.



It is the user's responsibility to create and maintain all data sets in his data base, and to define all data sets (both index and primary) in the File Control Table. Each data set, whether it is an index and/or primary data set, is first described as a primary data set in the File Control Table. That is, its basic physical characteristics must be defined so that CICS File Management may access it (for example, BLKSIZE, LRECL, KEYLEN, etc). If the data set is to be further used as an indirect access data set, it must also be defined with the following information:

1. The primary data set for which this data set is an index.

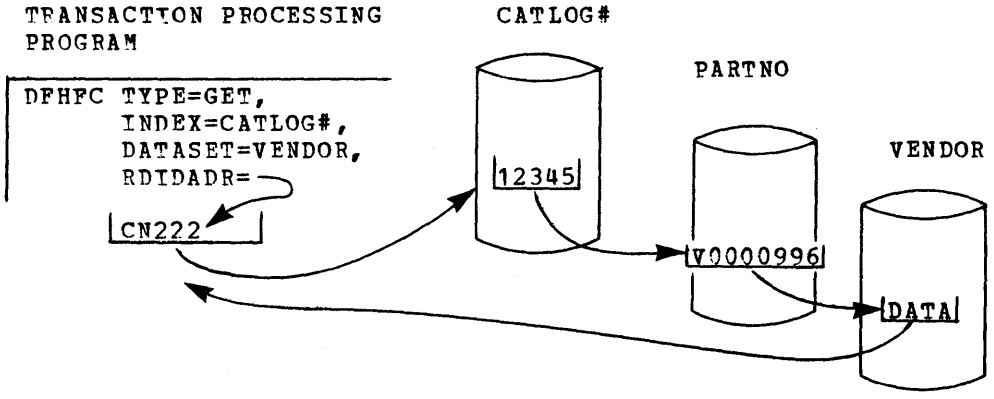
2. The location of the search argument, within the logical record of this data set, to be used for accessing the primary data set (or the next index data set).

If the user creates and properly defines an index hierarchy for indirect accessing, CICS File Management will service any request requiring use of that hierarchy, provided the requesting application program adheres to the following general rules and considerations:

1. The symbolic name of the first index data set to be searched in the retrieval process must be specified through the INDEX operand of the DFHFC macro instruction. This data set can be any index data set in a hierarchy of indexes, not necessarily the highest level index data set. It can also be the primary data set being accessed without the use of an index data set. However, in the latter case, the DATASET operand must be used instead of the INDEX operand.
2. The symbolic name of the primary data set from which data is to be ultimately retrieved and returned to the requesting program must be specified through the DATASET operand of the DFHFC macro instruction. Any number of intervening data sets can be used in the search; however, the user specifies only the first and the last data set. It is possible for the user to limit a search to only a portion of an index hierarchy; that is, it is not necessary to search an entire index hierarchy.
3. The search argument to be used by CICS File Management to access the first referenced data set must be specified through the RDIDADR operand of the DFHFC macro instruction. This search argument is either an ISAM key or a DAM Record Identification Field. If multiple levels of index data sets are involved, CICS File Management acquires a search argument for the next data set from the logical record of each successive data set.

When stepping through a series of index data sets, CICS File Management uses the requesting program's Record Identification field (specified in the RDIDADR operand) to store the search argument for each successive data set to be searched. It is the user's responsibility to ensure that this field is as large as the largest search argument that will be used in any given retrieval operation.

The following is an example of the above consideration in a three-level indirect accessing hierarchy. The search argument provided by the processing program is used to access the first index data set (CATLOG#) that provides the search argument for a second index data set (PARTNO) that provides the search argument for the primary data set (VENDOR) from which the data record is retrieved and returned to the user. Since the search argument retrieved from the second index data set (PARTNO) is eight bytes in length (V0000996), the user's Record Identification field (RDIDADR) must be at least eight bytes in length even though it initially contains only the five-byte search argument (CN222) for the first index data set.



DUPLICATE RECORDS

An optional feature of the indirect accessing approach to data base retrieval is the capability to indicate that a search argument in an index data set, which would normally reference the primary data set, instead references a "duplicates" data set. The need for or use of duplicates data sets may best be described as follows.

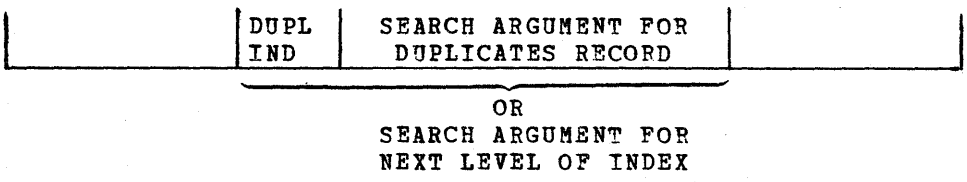
Assume that the application program requires access to an index data set organized by street address to obtain the name of the occupant at that address. The occupant's name is then used to access a primary data set organized by name.

For single occupancy, no problem exists. However, for multiple occupants, the index data set cannot directly equate a street address to a primary data set record. Instead, the search argument field in the index record indicates that multiple occupants (duplicates) exist and that the search argument provided references a duplicates data set rather than the primary data set.

CICS File Management retrieves the referenced record from the duplicates data set and returns it to the application program with a response code indicating a duplicates record. The duplicates record may contain further information, which the application program can use to more accurately retrieve a requested master record.

If an index data set is to indicate that there can be duplicate keys for entries in the primary data set that it references, the user must have previously included the necessary information in the File Control Table entry which describes the index data set. The index data set record must contain in the first byte of the search argument field a unique one-byte duplicates indicator (user-defined). Care must be taken to ensure that this indicator is a unique code, which cannot be the same as the first byte of a normal search argument for the primary data set.

The rest of the search argument field contains the search argument used by File Management to retrieve a record from the duplicates data set. This record has user-defined and user-constructed information that the application program can use to select the appropriate primary data set record. The following is an example of a search argument field in an index record that reflects duplicates:

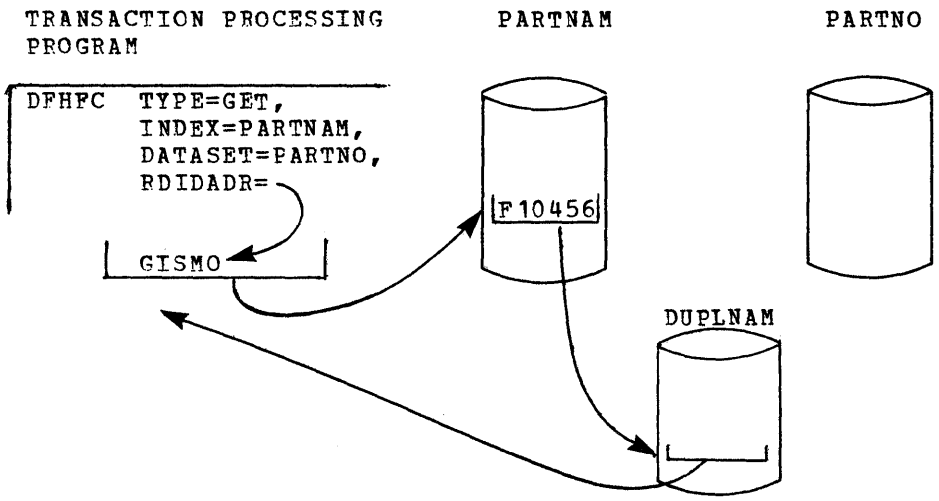


The search argument for the duplicates data set must meet the same search argument format requirements as those for a normal cross-index data set. Note that the length of the search argument used to access a duplicates data set is one byte smaller because of the duplicates indicator.

The following is an example of an index hierarchy with a duplicates data set. The application program begins the retrieval by accessing the index data set (PARTNAM) and ultimately accesses the primary data set (PARTNO). The search argument (GISMO) provided by the application program is a valid one for the index data set (PARTNAM), but it provides a record containing a duplicates flag.

When the duplicate indicator is detected, CICS File Management uses the new search argument (from the PARTNAM data set) to access the duplicates data set (DUPLNAM), returning the duplicates record to the application program.

In this example, the part name (GISMO) is not unique since there are several types of GISMO's in the part number (PARTNO) data set. The requesting program must provide more qualifying data concerning which GISMO is desired.



The record retrieved from the duplicates data set in the example might appear as follows:

GISMO	LARGE	9123	MED	9872	SMALL	9944
PARTNAM	DESC	PARTNO	DESC	PARTNO	DESC	PARTNO

The application program might formulate a message to be routed to the inquiring terminal asking the terminal operator to make a choice. For example:

```
PART NAME REQUESTED HAS MULTIPLE ENTRIES

PLEASE SELECT SPECIFIC PART NUMBER

    PART NAME   DESCRIP     PART NUMBER

      GISMO     LARGE           9123

                MED             9872

                SMALL           9944
```

Once the terminal operator has made a selection, the processing program can make a direct retrieval from the primary (PARTNO) data set.

Note that if the index record in the above example had not contained a duplicates indicator, CICS File Management would have used the search argument to access the primary data set (PARTNO) and retrieve the requested data.

Record Identification Field

The Record Identification field is the means by which the application program communicates to CICS File Control the identity of the specific record which is being sought. (See the discussion of the RDIDADR operand as it applies to the DFHFC macro instruction.) For ISAM organized data sets, this field is relatively simple in structure since it contains only the key of the logical record. However, for DAM organized data sets the Record Identification field structure is a bit more complex, since it is necessary for the application program to supply the block reference information, physical key (if keyed data sets are being used), and the deblocking argument (if blocked data sets are being used).

Note: If more than one browse operation or update operation is to be concurrently performed by a single application program, a unique Record Identification field must exist for each operation.

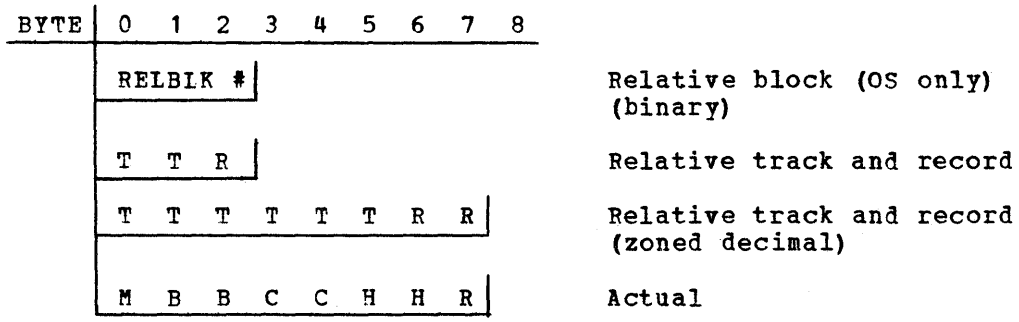
The Record Identification field for DAM data sets is really a concatenation of three subfields, identified as follows:

1. Block reference

   The physical identifier of the DAM block, is specified by the RELTYPE operand of the File Control Table and may be one of the following:

   a. Relative Block (CICS/OS only) three-byte binary (RELTYPE=BLK)
   b. Relative Track and Record - two-byte TT, one-byte R (RELTYPE=HEX)
   c. Relative Track and Record (zoned decimal format) six-byte TTTTTT, two-byte RR (RELTYPE=DEC)
   d. Actual address - eight-byte MBBCCHHR (RELTYPE omitted)

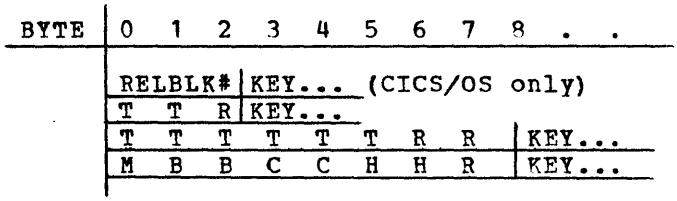
   EXAMPLE



2. Physical key

   The 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 parameter for the File Control Table (FCT) entry which defines the data set. It must immediately follow the block reference information, which can be any of the above.

EXAMPLE

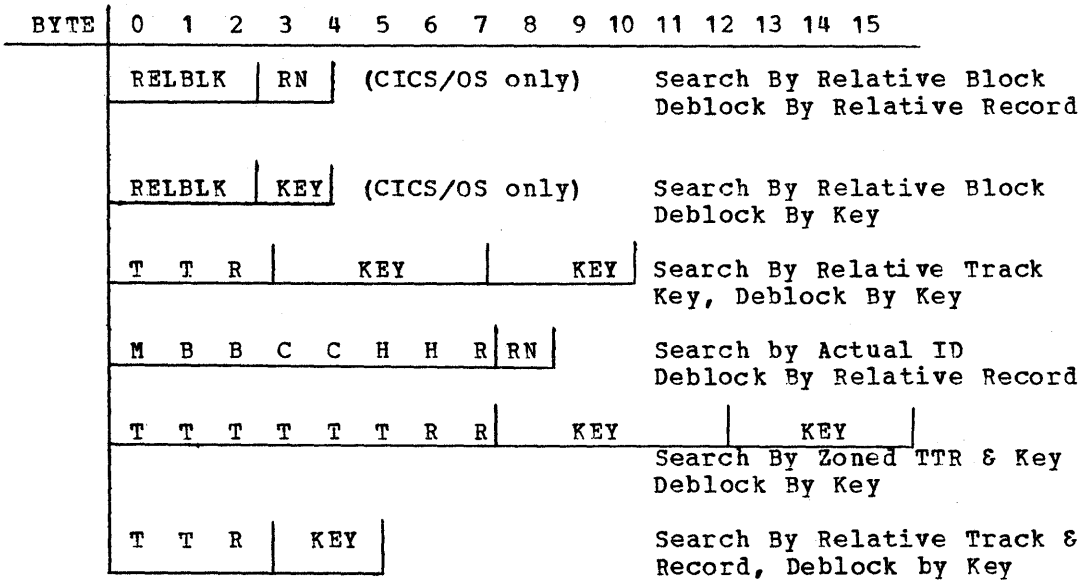


3. Deblocking argument

The deblocking argument is required only if the data set contains blocked records and the user wishes to retrieve a logical record from within a block. It is not mandatory that the user deblock every physical record; he may wish to retrieve the entire block. The deblocking argument may be either a key or a relative record number. The user's choice is specified in the RETMETH operand of the DFHFC macro instruction. If present, 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 parameter of the File Control Table (FCT) entry which describes the data set. Note that the key used for deblocking need not be the same size as the physical record key (BLKKEYL). If the deblocking argument is relative record number, it is represented by a one-byte binary number, with a value of zero representing the first logical record of a block.

EXAMPLE (physical key = 6 bytes, deblocking key = 3 bytes)



Adding Records to DAM Data Sets

When adding new records to DAM data sets, the application programmer should be aware of the following considerations and restrictions:

1. The addition of undefined or variable-length records (keyed or non-keyed) requires the user to indicate the track on which the 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 user's Record Identification field. The track specification may be in any of the acceptable formats except relative block. If zoned decimal relative format is used, the record number is returned as a two-byte zoned decimal number in the seventh and eighth positions of the Record Identification field.

In the CICS/DOS system, an attempt to add a variable-length or undefined record is limited to the single track specified by the user. If not enough space is available on that track, a "no space available" error is returned to the user who may then try to add the record on another track. Under these circumstances, the record is returned by the user in an FWA, the address of which is at TCAFCAA. The user need only modify the track identification and issue another DFHFC TYPE=PUT, TYPEOPR=NEWREC macro instruction to add the record on another track.

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

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. (A dummy record is signified by a key of hexadecimal 'FF's; in CICS/OS, the first byte of data contains the record number.)

3. For non-keyed, 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, destroying whatever was previously recorded at that location.

4. For keyed, fixed-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 is found, the new key and record replaces it, the exact location at which the new record is located is returned to the user in the block reference subfield of the Record Identification field.

   For example, suppose a user wishes to add a keyed, fixed-length record to his DAM data set. He determines (through some algorithm) that the search is to start at relative track 3. His Record Identification field might look like the following:

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

   When control is returned to the user, his 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 the TCA at TCAFCURL in a two-byte binary format. When an undefined record is retrieved, it is the user's responsibility to determine its length.

6. When adding variable length records to a BDAM file, it is the responsibility of the application programmer to insert the Block Length.

REQUESTING DATA LANGUAGE/I SERVICES UNDER CICS/OS

The application programmer can request Data Language/I (DL/I) services under CICS/OS through CALL's written according to DL/I specifications or by issuing a DFHFC macro instruction. In response to such requests, control is passed to a CICS-DL/I interface routine that acts as an interface between the CICS application program and the DL/I request handler in a DL/I task (which is an OS subtask of CICS). This routine performs validity checks on the CALL list, sets up DL/I to handle this particular request, and passes control and the CALL list to DL/I. When the interface regains control, it returns control to the calling program unless a DL/I pseudo-abend has occurred, in which case the CICS transaction (task) is abended.

QUASI-REENTRANT CONSIDERATIONS WITH REGARD TO DL/I CALL'S

Under IMS, programs are not required to be reentrant since only one transaction can use a particular program at any one time. In CICS, if several transactions are being serviced which require the use of one application program, one copy of the program is executed in a reentrant manner by several CICS subtasks. Therefore, DL/I areas that will be modified (such as PCB pointers, I/O work areas, and Segment Search Arguments) may not be placed in either static storage or working storage. Storage for PCB pointers, Segment Search Arguments, and work areas must be obtained from CICS dynamic storage by each transaction using the program. (See the section "Quasi-Reentrance".)

The four steps to requesting DL/I data base services are as follows:

1. Obtain addresses of PCB's used by the application program.
2. Acquire storage for segment search arguments (SSA's) 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 PCB's

An application program that uses the CICS-DL/I interface references data bases by means of Program Communication Blocks (PCB's). In the Program Specification Block (PSB) for the program, there is one PCB for each data base. In a DL/I environment, upon entry, the application program receives the addresses of the PCB's of the data bases it uses. Since CICS handles application programs as main programs and not as subprograms, the PCB addresses cannot be obtained via entry conventions but must be obtained by the application program before it makes any DL/I CALL's.

To successfully process DL/I CALL's within CICS transactions, the PSB for the transaction must be scheduled and the PCB addresses must be obtained before any DL/I CALL's are made. If they are not obtained, any DL/I CALL's made return an INVREQ (invalid request) indicator. The scheduling process gives the transaction exclusive control of the PSB. This prevents other transactions from updating segment types that this transaction is updating.

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 indicator to be returned.

To schedule the desired PSB and obtain PCB addresses, the application programmer uses a special form of the DFHFC macro instruction:

```
DFHFC TYPE=(DL/I,PCB),                                          *
      PSB=psbname,symbolic address,YES,                         *
      NORESP=symbolic address,                                  *
      INVREQ=symbolic address
```

A discussion follows concerning the operands that can be included in this macro instruction.

TYPE: TYPE=(DL/I,PCB) indicates a request for PCB addresses.

PSB: This operand is used to specify the name of the PSB to be used. The name can be the actual name enclosed in quotes, or the name of an eight-byte field containing the name of the PSB, padded to the right with blanks. If the application programmer wishes to enter the PSB name in the TCADLPSB field before the CALL, he specifies PSB=YES. If this operand is omitted, the name of the program associated with the transaction in the PCT is used as the PSB name.

NORESP: This operand is used to specify the label to be branched to if the PSB was located and the PCB addresses were returned. If the PSB could not be scheduled, or if this operand is omitted, processing continues with the next sequential instruction.

INVREQ: This operand is used to specify the entry label of a user-written routine to be given control under any of the following conditions:

1. When an attempt is made to schedule a PSB while the transaction is still using a previously scheduled PSB.
2. The PSB name specified is not in the PDIR (PSB Directory) list.
3. The PSB or one of its associated DMB's (Data Management Blocks) does not exist in the Application Control Block (ACB) Library.
4. One of the DMB's associated with the PSB is not in the DDIR (DMB Directory) list.

If the PSB has been located, the TCADLPCB field contains the address of a list of PCB addresses which is 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 PSB's or DMB's are in their pools, the transaction is terminated with the DLPA abend code; DL/I pseudo-abend code 992 or 993 is placed in the transaction's TCA at TCADLECB and the pseudo-abend message is sent to destination CSMT.

BUILDING SEGMENT SEARCH ARGUMENTS (SSA's)

Segment Search Arguments (SSA's) can be used in a DL/I CALL to identify a specific segment, or, if qualified, to identify the range of values within which a segment exists. If used, SSA's are built by the application programmer before a DL/I CALL is issued. See the

IMS/360 Appplication Programming Reference Manual for information concerning how to build an SSA.

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

The storage acquired to build the SSA's is addressed as follows:

1. For Assembler language programs, the address should be placed in the register that establishes addressability for the SSA dynamic storage.

2. For ANS 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.

3. For PL/I, the address is stored in the variable upon which the SSA dynamic storage is based.

After the storage has been acquired, the Segment Search Arguments are built according to the DL/I specifications found in the IMS/360 Application Programming Reference Manual.

In a DL/I CALL statement, the names of the SSA's to be used, if any, are specified in the parameter list. In a DFHFC TYPE=DL/I macro instruction, the application programmer can specify the number and names of the SSA's in different ways:

1. SSAS=NO indicates that there are no SSA's in this CALL.

2. SSAS=(ssacount,ssa1,ssa2,...), where "ssacount" is optional, represents either the fixed-point number of SSA's in the CALL or the symbolic address of the fullword that contains the number of SSA's. Specifying a field to contain the number of SSA's provides the application programmer with flexibility in writing one DFHFC statement to be used in many different CALL's. "ssa1", "ssa2", etc., are the symbolic names of the SSA's.

3. SSALIST=YES indicates that the application programmer has built a list of fullwords, optionally containing the number of SSA's (which may be zero) in the first word, and the addresses of the SSA's in the following words, and that he has stored the address of this list at TCADLSSA.

4. SSALIST=listname indicates that "listname" is the address of an SSA list built by the application programmer as indicated in item 3.

In Assembler language programs, "ssalist", "ssacount","ssa1", "ssa2", etc., can be contained in registers by enclosing the specification in parentheses.

ACQUIRING AN I/O WORK AREA

A request for services in a DL/I transaction under IMS always includes the address of a work area, either where a current segment is contained, or where DL/I is to place the segment in a retrieval CALL. In a CICS application program, this area must be specified in a CALL or CALLDLI, but may be provided by the interface, if the programmer desires, for a retrieval type DFHFC macro request.

If the application programmer knows the address of the work area to be used in the DFHFC macro instruction, including the case where he acquires storage for a retrieval type (Gxxx) request, he specifies either the name of the pointer to that storage in the WRKAREA=name operand, or places the address of the storage in TCADLIO 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 found at TCADLIO. The address at TCADLIO is the address of the Storage Accounting Area (SAA) preceding the retrieved data. The length of the data is eight bytes less than the value found in the second halfword of the SAA. The area becomes the responsibility of the programmer and is not freed until he frees it or until the transaction terminates.

Note: The address of the I/O area is specified as the address of the Storage Accounting Area preceding the data for a DFHFC request, or as the address of the first byte of the data area for a CALL or CALLDLI.

ISSUING THE DL/I CALL

A CICS application program can request DL/I services in either of two ways: CALL's written according to DL/I specifications or DFHFC macro requests using unique DL/I operands.

For Assembler language:

```
CALLDLI ASMTDLI,(parmcount,function,pcb,workarea,segment
                search arguments,...) or
CALLDLI CBLTDLI,(parmcount,function,pcb,workarea,segment
                search arguments,...)
```

In this macro instruction, which alters the contents of register 1, "parmcount" is an optional parameter. The operation code is CALLDLI rather than CALL since the expansion of the CALL to the interface is not the same as the ordinary expansion of a CALL. If no parameters are specified, it is assumed that register 1 contains the address of the parameter list. ASMTDLI and CBLTDLI are functionally equivalent specifications.

An alternate form of this specification is:

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

which is written in the same format as the E-TYPE OS CALL macro instruction; that is, "address" is the address of the parameter list or "register" contains the address of the parameter list.

For ANS COBOL:

```
CALL 'CBLTDLI' USING parmcount,function,pcb,workarea,
                     segment search arguments,...
```

For PL/I:

```
CALL PLITDLI (parmcount,function,pcb,workarea,
              segment search arguments,...);
```

Note: In a CALLDLI or CALL statement, the "workarea" parameter must point to the first byte of the data area.

The following macro instruction is used to specify the desired DL/I functions to be performed, regardless of the programming language used:

```
DFHFC TYPE=(DL/I,function),                                    *
      PCB=symbolic address,(register),                         *
      WRKAREA=symbolic address,YES,(register),                 *
      SSAS=NO,(ssacount,ssa1,ssa2,...),                        *
      SSALIST=YES,NO,symbolic address,(register),              *
      NORESP=symbolic address,                                 *
      NOTOPEN=symbolic address,                                *
      INVREQ=symbolic address
```

TYPE: This operand is used to specify the two- to four-byte name of the function to be performed. If it is not specified, the function must have been specified in the TCADLFUN field before the DFHFC macro instruction is issued.

PCB: The PCB=symbolic address operand is used to specify the name of the field that contains the address of the PCB.

WRKAREA: WRKAREA=YES indicates that the application programmer has placed the address of the work area to be used at TCADLIO.

WRKAREA=symbolic address specifies the address of the field that contains a pointer to the I/O work area.

If the WRKAREA operand is not specified and this is a Gxxx request, the CICS-DL/I Interface acquires storage for the work area and stores the address at TCADLIO. The user must save this address upon return. In any other type of request, the user must provide the work area.

Note: The work area whose address is specified in a DFHFC macro instruction or whose address was previously placed at TCADLIO includes the CICS Storage Accounting Area prefix; the work area specified in a CALLDLI or CALL statement does not.

SSAS: SSAS=NO indicates that there are no SSA's used in this request. SSAS=NO is the default specification.

SSAS=(ssacount, ssa1, ssa2, ...) is used to specify the names of Segment Search Arguments in this request (thereby creating an SSA list). "ssacount" is used, optionally, to specify the number of SSA's to be used in this request; this parameter represents the address of a fullword containing the count, or, in the case of Assembler language, may be expressed as a numeric value. If the ssacount parameter is omitted, the ssa1 specification represents the first element of the SSA list. For a further description of the SSA list, see the following discussion under SSALIST.

SSALIST: SSALIST= symbolic address is used to specify the name of a field that contains the address of an SSA list. The first element of an SSA list may, optionally specify either the number of SSA's to be used in this request or the address of a full word containing this value; the remaining elements represent addresses of SSA's. If the first element of an SSA list does not represent "ssacount", all elements of the SSA list are assumed to be addresses of SSA's; the high-order

bit of the last element of the list must be set on to indicate the end of a variable-length list.

SSALIST=YES indicates that the user has previously placed the address of the SSA list at TCADLSSA.

SSALIST=NO indicates that no SSA list is used in this request. The default is SSALIST=NO.

If either WRKAREA=YES or SSALIST=YES is specified the address of the I/O work area or SSA list must be placed in the TCA prior to issuing the DFHFC macro instruction. The TCA fields containing these addresses are altered during the service of the request.

Note: SSAS and SSALIST are mutually exclusive operands.

NORESP: NORESP=name is used to specify the label to which control is to be passed after this transaction has regained control. The CICS-DL/I Interface must have been able to pass control to DL/I and a DL/I pseudo-abend of the transaction must not have occurred. The user must check the return code in the PCB to see if DL/I was able to properly service the request. If this operand is omitted, control is passed to the next sequential instruction.

NOTOPEN: NOTOPEN=symbolic address is used to specify the label to which control is to be passed if this data base is logically (not necessarily physically) closed. The PCB will not contain an AI status code.

INVREQ: INVREQ=symbolic address is used to specify the label of a user-written routine to which control is to be returned if the transaction attempts to access DL/I without first scheduling a PSB and obtaining PCB addresses.

Note: In Assembler language application programs certain operands may be specified as registers and enclosed in parentheses; for example:

1: PCB=(register) where "reg" contains the address of the PCB to be used in this request.

2: WRKAREA=(register) where "reg" contains the address of the work area to be used in this request.

3: SSAS=((register 1), (register 2), (register 3),...) where "register 1" contains the optional count of SSA's, and "register 2", "register 3", etc., point to SSA's to be used.

4: SSALIST=(register) where "register" points to a previously constructed SSA list (described above under SSALIST=symbolic address).

RELEASING A PSB IN THE CICS APPLICATION PROGRAM

To reduce pool and intent contention, the CICS application program may release the PSB. Before making any other DL/I CALL's, the program must again issue a scheduling CALL.

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.

A CICS application program can release a PSB for use by other transactions by issuing a

```
DFHFC TYPE=(DL/I,T)
```

request. The PSB is released for use by other transactions, or if not required, its pool space and associated DMB pool space may be released. No other DL/I CALL's may be made in this transaction until another scheduling (PCB) CALL is made.

CHECKING THE RESPONSE TO A REQUEST FOR DL/I SERVICES (CHECK)

To test whether or not the CICS-DL/I Interface successfully processed the DL/I request, the

```
DFHFC TYPE=CHECK,                                          *
      NORESP=symbolic address,                             *
      INVREQ=symbolic address,                             *
      NOTOPEN=symbolic address
```

macro instruction can be issued.

NORESP: NORESP is used to specify the label of the user-written routine to which control is to be passed upon normal execution of the request.

INVREQ: INVREQ is used to specify the label of the user-written routine to which control is to be passed in the event the transaction has not scheduled a PSB and obtained PCB addresses.

NOTOPEN: NOTOPEN specifies the label of a user-written routine to which control is to be passed in the event the requested data base named in the PCB used in the request was logically (not necessarily physically) closed. The PCB will not contain an AI status code.

The application programmer may use the DFHFC TYPE=CHECK macro instruction following a CALLDLI, CALL, or DFHFC TYPE=(DL/I). This macro instruction does not check for DL/I return codes in the PCB. In the event DL/I issues a pseudo-abend during processing of the request, control is not returned to the transaction. The transaction is terminated with CICS abend code DLPA.

DL/I REQUESTS WRITTEN IN ASSEMBLER LANGUAGE

The application programmer must first get the PCB addresses. (There are several examples below.) When CICS returns from servicing the DFHFC TYPE=(DL/I,PCB) request, if the programmer loads register 1 from TCADLPCB, his program is in the same state as after an

```
ENTRY DLITCBL
```

statement when executing an IMS DL/I application program.

The examples that follow show the options available to the application programmer in a few of the acceptable combinations. Note that the application program must be kept quasi-reentrant; that is,

addresses, etc., should not be stored in static storage. Note also that if a DFHFC macro insturction is used the PCB and WRKAREA operands are used to specify the address of a pointer to the field rather than the field itself.

For a complete discussion concerning the checking of these responses, see the section "Test Response to a Request for File Services".

The following is an example of the coding required to request DL/I services in an Assembler language application program.

```
        COPY DFHTCADS                   COPY TCA DEFINITION - INCLUDES
*                                       DL/I FIELDS
PSBNAME DC   CL8'PSBNAME1'              NAME OF PSB TO BE USED
PCBPTRS DSECT                           PCB POINTERS RETURNED BY
*                                       INTERFACE
PCB1PTR DS   F                          STORAGE FOR PCB POINTERS
PCB2PTR DS   F
        .
        .
        .

WORKAPTR DS  F                          STORAGE FOR PRINTER 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
* ACQUIRE STORAGE FOR WORKAREA
       DFHSC TYPE=GETMAIN,...           GET STORAGE FOR WORKAREA
       L     R2,TCASCSA                 REG 2 IS BASE FOR WORKAREA
       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,WRKAREA,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,         WORKAREA 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,              WORKAREA 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 SSAS
       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 PROGRAM PROVIDES SSA LIST
       L     R3,TCADLIO                PICK UP ADDRESS OF SUPPLIED
*                                      WORKAREA
```

DL/I REQUESTS WRITTEN IN ANS COBOL

Upon program entry the ANS COBOL programmer should obtain PCB addresses by issuing a DFHFC TYPE=(DL/I,PCB) request. After CICS returns control, the programmer moves the TCADLPCB field 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 PCB's to their BLL pointers to provide the base addresses for the PCB's. When this is done, the program is in the same state that it would be in after execution of the ENTRY 'DLITCBL' USING PCB1,PCB2 statement if the program were written for DL/I.

For an explanation of how BLL pointers to 01 statements in the Linkage Section are defined, see the discussion of ANS COBOL application programming in the section "Storage Definition".

Various examples are provided below concerning how to write DL/I requests; only some combinations of operands are shown, but other combinations are acceptable. Note that in a DFHFC request the BLL pointers to the PCB and work area are used rather than the actual field names themselves. This is the only way the addresses can be passed to DL/I.

The following is an example of the coding required to request DL/I services in an ANS COBOL application program:

```
WORKING-STORAGE SECTION.
77  PSBNAME PICTURE X(8) VALUE 'COBOLPSB'.
77  FUNCTION-1 PICTURE X(4) VALUE 'DLET'.
77  SSA-COUNT PICTURE 9(8) COMPUTATIONAL VALUE +2.
LINKAGE SECTION.
01  DFHBLLDS COPY DFHBLLDS
    02  ...                              POINTERS TO OTHER CICS AREAS
*                                        NEEDED
    02  B-PCB-PTRs PICTURE 9(8) COMPUTATIONAL.
    02  B-PCB1 PICTURE 9(8) COMPUTATIONAL.
    02  B-PCB2 PICTURE 9(8) COMPUTATIONAL.
    02  B-WORKAREA PICTURE 9(8) COMPUTATIONAL.
    02  B-SSAS PICTURE 9(8) COMPUTATIONAL.
01  DFHCDASD COPY DFHCSADS.
01  DFHTCASD COPY DFHTCADS.
    .                                    NOTE TWO DEFINITIONS.
    .                                    NOTE OTHER AREA DEFINITIONS.
    .
01  PCB-PTRS.
    02  PCB1-PTR PICTURE 9(8) COMPUTATIONAL.
    02  PCB2-PTR PICTURE 9(8) COMPUTATIONAL.
01  PCB1.
    .
    .
    .
01  PCB2.
    .
    .
    .
01  WORKAREA.
    02  FILLER PICTURE X(8).            STORAGE PREFIX.
    02  WORKA1 PICTURE X(40).
    .
    .
    .
01  SSAREA.
    02  FILLER PICTURE X(8).
    02  SSA1 PICTURE X(40).
    02  SSA2 PICTURE X(60).
    .
    .
    .
PROCEDURE DIVISION.
* GET PCB ADDRESSES
    DFHEC TYPE=(DL/I,PCB)               GET PSB FOR THIS PROGRAM
* 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,WORKAREA,SSA1,SSA2.
* EXAMPLE 1 OF DFHEC MACRO INSTRUCTION
    DFHEC TYPE=(DL/I,GHU),           FUNCTION                          *
          PCB=B-PCB1,                PCB POINTER                       *
          WRKAREA=B-WORKAREA,        WORKAREA POINTER                  *
          SSAS=(SSA-COUNT,SSA1,SSA2) SSA COUNT AND NAMES
* EXAMPLE 2 OF DFHEC MACRO INSTRUCTION
    MOVE 'GNP' to TCADLFUN           NOTE PRELOAD FUNCTION.
    MOVE B-PCB1 to TCADLPCB          NOTE PRELOAD PCB ADDRESS.
```

```
    DFHFC TYPE=DL/I,                    FUNCTION PRELOADED                *
                                        PCB ADDRESS PRELOADED             *
                                        WORKAREA TO BE ACQUIRED           *
          SSAS=NO                       NO SSA'S
    MOVE TCADLIO to B-WORKAREA.         NOTE SAVE ACQUIRED WORK AREA ADDR.
```

DL/I REQUESTS WRITTEN IN PL/I

Upon entry to his program, the PL/I application programmer should get PCB addresses via a DFHFC TYPE=(DL/I,PCB) statement. When CICS returns, the BASE of a structure of PCB pointers is in TCADLPCB. The PL/I programmer must move the BASE value from TCADLPCB to the BASE of his declared structure of PCB pointers. He then loads the BASE's of all the PCB's from this structure. The program is now in the same state that the DL/I application program would be following execution of the

```
DLITPLI: PROCEDURE (pcbname1, ...) OPTIONS (MAIN);
```

statement, if the program were an IMS DL/I application program.

The PL/I programmer may then make DL/I requests, either via CALL's or via DL/I DFHFC macro instructions. Note that in a DFHFC request the PCB and WRKAREA operands specify the address of a pointer to the field rather than the field itself.

The following is an example of the coding required to request DL/I services in a PL/I application program:

```
%INCLUDE DFHCSADS;                          /* CSA DEFINITION */
%INCLUDE DFHTCADS;                          /* TCA DEFINITION - INCLUDES */
                                            /* DL/I FIELDS */
    DECLARE 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 */
              2 STORAGE_PREFIX CHAR (8),    /* I-O AREA */
              2 IOKEY CHAR (6),             /* DEFINITION */
              2 ... ;
    DECLARE 1 DLI_SSADS BASED (BSSADS),     /* DL/I */
              2 STORAGE_PREFIX CHAR(8),     /* SSA */
              2 SSA1,                       /* DEFINITIONS */
                3 SSA1KEY CHAR(6),
                3 ...
              2 SSA2,
                3 ...
                3 ...;
/*  OBTAIN PCB POINTERS  */
       DFHFC TYPE=(DL/I,PCB)
/*  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;                   PRELOAD PCB POINTER
    DFHFC TYPE=(DL/I,GU),             PCB PRELOADED                  *
                                      WORKAREA TO BE ACQUIRED        *
        SSAS=(SSA_COUNT,SSA1,SSA2)    SSA COUNT NAMES
    BDLIIO=TCADLIO;                   /* SAVE ACQUIRED WORK AREA ADDR */
/* EXAMPLE 3 OF DFHFC MACRO INSTRUCTION */
    TCADLFUN='GN';                    /* PRELOAD FUNCTION */
    TCADLIO=BDLIIO;                   /* PRELOAD WORKAREA ADDRESS */
    DFHFC TYPE=DL/I,                  FUNCTION PRELOADED             *
          PCB=BPCB1,                  PCB POINTER                    *
          WRKAREA=YES,                WORK AREA ADDRESS PRELOADED    *
          SSAS=NO                     NO SSA'S
```

BASIC MAPPING SUPPORT FOR THE 3270

CICS provides Basic Mapping Support (BMS) for use with the IBM 3270 Information Display System. By use of BMS, the CICS application programmer has access to input and output 3270 data streams without the need to include any 3270 device-dependent code in the CICS application program.

Application programs that utilize BMS under CICS remain independent of the 3270 data stream format. They also remain compatible with future additions of new fields to the existing input and output map formats.

Two types of maps are assembled offline through use of CICS macro instructions: (1) a physical map which is used by CICS to convert 3270 native mode data into the format desired by the application programmer, and (2) a symbolic description map which is used by the application programmer to symbolically reference the data in the 3270 buffer. The CICS DFHMDI macro instruction is used to build both types of maps; DFHMDI TYPE=MAP indicates a physical map while DFHMDI TYPE=DSECT indicates a symbolic description map.

The user defines and names fields and groups of fields that may be written to and received from the 3270. The assembled physical map contains all the 3270 device-dependent control characters necessary for the 3270 data stream.

The symbolic description map can be copied into each application program that uses the associated physical map. Data is 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 by referencing each field by name, 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 to make the appropriate changes in the macro instructions that describe the map and then reassemble both the physical map and symbolic description map. The new symbolic description map must then be copied into the application program and the program reassembled.

An application program has access to the input and output data fields using the names supplied to the fields when the maps were generated. The application 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 screen 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 screen format of those fields.

Basic Mapping Support (BMS) is available to application programmers coding in PL/I, ANS COBOL, or Assembler language. Input maps describe the fields which are potentially receivable from a 3270 screen; output maps specify the format of data to be sent to a 3270 screen or printer.

By using BMS to construct and interpret the 3270 data streams, application programmers can insulate application programs from the device-dependent considerations required to handle 3270 data streams. If necessary, the application program has the facility to temporarily modify the attributes of an output map or of any named field in an output map. BMS supplies a collection of named attribute combinations so that the application program remains essentially independent of the 3270 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 3270 formats. Design and programming of the first stages of applications can begin before later stages have even been designed. This early implementation is protected from updates in the screen formats.

MAP DEFINITION

Input Mapping

Input maps are defined using the DFHMDI and DFHMDF macro instructions during offline map generation.

Each field to be read must be defined as to maximum data length and starting position. This operation produces a map and a symbolic storage definition of the TIOA data supplied by BMS.

The physical map is used by BMS to construct a TIOA as defined by the symbolic storage definition to be returned to the user transaction.

The TIOA symbolic storage definition contains the length of the input data followed by the data read. Space is reserved for the maximum length defined for each field (not to exceed 256 bytes).

Pen-detectable fields have one reserved byte that contains a hexadecimal 'FF' if the field is selected or a hexadecimal '00' if the field is not selected. The length field always contains a halfword binary one.

The length specified may differ from the actual number of characters in the field. If more data is keyed than specified, the data is

truncated to the number of characters requested in the map; the length is returned as the truncated length. If less data is keyed than specified, the remaining character positions are filled with blanks or zeros and the length of the keyed data is returned in the length field. The maximum length allowed for any one field is 256 characters.

Any keyed fields not defined by the map are discarded. Any fields defined but not keyed have their length field set to zeros and the data field set to nulls (X'00').

The program can access the length or data of any field by symbolic labels. The length field is a halfword binary field and is addressed by the label "fieldname.L" or "groupname.L". The data portion of each field (or group of fields) is contiguous with the length field. A group of fields, or a single field not within any group of fields, has the data portion addressed by the name "groupname.I" or "fieldname.I". For fields contained within a group, there are no intervening length fields (only "groupname.L" exists) and each field has the name "fieldname.I".

Note that the "." is a concatenation symbol and is not used when referencing either the data or the data length. For example, in the case of field name XYZ, the data is referenced as XYZI; the data length is referenced as XYZL.

Output Mapping

Output maps, like input maps, are created offline during map generation using the DFHMDI and DFHMDF macro instructions. Each field to be displayed must be defined as to starting location, length, field characteristics, and default data (if desired).

When defining fields, the user may name any field he desires to override at execution time. Any named fields are produced in a symbolic storage definition of the TIOA to allow symbolic reference to each field. The user may temporarily override the field characteristics, the data, or both field characteristics and data, by inserting the desired changes into the TIOA under the field names in the symbolic storage definition map which he has in his program.

The fields are assigned names as specified in the DFHMDF macro instruction. 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", and there is no separate attribute byte for the field. (Only groupnames can have an attribute byte.) For a groupname, or a field not contained within a group, the data area is given the name "groupname.O" or "fieldname.O". A field not contained within a group is treated as a group containing just a single field entry.

Note that the "." is a concatenation symbol and is not used when referencing either the data or the data attributes. For example, in the case of field name XYZ, the data is referenced as XYZO; the attribute byte is referenced as XYZA.

Pen-detectable fields should be "auto skip" to prevent data from being keyed into them.

Note: Due to the nature of the pen-detectable fields, they should normally not be modified. However, if the data field is modified, the first character must be a "?" or blank character; otherwise, the field is no longer pen detectable.

Output field data, whether initial map data or data supplied by the program, must not begin with a null character (X'00'). Blank characters (X'40') should be used to position displayable data down a field.

OFFLINE MAP BUILDING

The following macro instruction is the initial and final macro instruction for offline map generation and is used to build the physical map and symbolic description map:

```
mapname DFHMDI TYPE=DSECT,MAP,FINAL,                                    *
               TERM=3270,                                               *
               LANG=ASM,COBOL,PL1,                                      *
               BASE=name,                                               *
               MODE=IN,OUT,                                             *
               CTRL=(PRINT,L40,L64,L80,HONEOM,FREEKB,ALARM,FRSET)
```

All maps must be given a user-defined map name of from one to seven characters, beginning with an alpha character. If the map is to reside in the CICS program load library, the map name chosen must be different from other map names or program names in the system.

TYPE=MAP or TYPE=DSECT may be specified if this is the initial macro instruction for offline map generation; TYPE=MAP indicates a physical map and TYPE=DSECT indicates a symbolic description map. TYPE=FINAL must be specified if this is the final macro instruction.

When a symbolic storage definition is generated in response to a DFHMDI TYPE=DSECT, MODE=IN specification, an "I" is appended to each map name; when generated in response to a DFHMDI TYPE=DSECT, MODE=OUT specification, an "O" is appended to each map name. For example:

```
MAP1   DFHMDI TYPE=DSECT,                                               *
              TERM=3270,                                                *
              LANG=ASM,                                                 *
              MODE=IN,...
```

In this example, the name generated in the symbolic storage definition is MAP1I and must be referenced as such within the application program. This is true irrespective of the programming language used.

DSECT: A DSECT (symbolic storage definition) map generation run creates the list of field names which the user copies or includes in the application program. If the same map definition is used by application programs written in different languages, a separate DSECT run is required for each language to put the table of field names into the Copy library of each language.

MAP: A MAP generation run creates the control information block used by BMS to perform the mapping. This map is stored in the CICS program load library and is loaded as required by BMS. The Assembler language application programmer may generate the map in his code and pass the address across to BMS.

FINAL: This parameter must always be coded as part of the last macro instruction of a map definition (after all the field definition macro instructions). No other operands are required with DFHMDI TYPE=FINAL; they are ignored if coded.

TERM: This operand can only contain the 3270 keyword parameter. If this operand is omitted, it defaults to TERM=3270.

LANG: Required only for a DFHMDI TYPE=DSECT run, this operand is ignored in the case of the DFHMDI TYPE=MAP and DFHMDI TYPE=FINAL specifications, both of which are language independent.

BASE: The BASE=name operand is used to group symbolic storage definitions by specifying the group name in each applicable DFHMDI TYPE=DSECT specification. This operand is applicable only when the programming language is ANS COBOL or PL/I; it is not applicable in the case of a TYPE=MAP operation or if the programming language is Assembler language.

The following example illustrates the use of the BASE operand:

```
MAP1    DFHMDI TYPE=DSECT,                                      *
               LANG=COBOL,                                      *
               MODE=IN,                                         *
               BASE=DATAREA1 ...

MAP2    DFHMDI TYPE=DSECT,                                      *
               LANG=COBOL,                                      *
               MODE=OUT,                                        *
               BASE=DATAREA1 ...
```

The symbolic storage definitions of this example might be referenced in an ANS COBOL application program as follows:

```
       LINKAGE SECTION.
       01 DFHBLLDS COPY DFHBLLDS.
          .
          .
          .
          02 TIOABAR PICTURE S9(8) COMPUTATIONAL.
          02 MAPBASE1 PICTURE S9(8) COMPUTATIONAL.
          .
          .
          .
       01 DFHTIOA COPY DFHTIOA.
       01 DATAREA1 PICTURE X(1920).
       01 name COPY MAP1I.
       01 name COPY MAP2O.
```

MAP1 and MAP2 multiply redefine DATAREA1; only one 02 statement is needed to establish addressability. However, the program can only reference fields in one of the symbolic map areas at a time; to reference fields in the other symbolic map areas, the program must establish addressability to each of those areas.

If BASE=DATAREA1 is deleted from this example, an additional 02 statement is needed to establish addressability for MAP2; the 01 DATAREA1 statement would not be needed. The program could then reference 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.

MODE: MODE=IN specifies an input map generation. MODE=OUT specifies an output map generation. This operand is not required for the DFHMDI TYPE=FINAL macro instruction.

CTRL: This operand is used to specify various control functions for a particular output map which are allowable on certain of the 3270 devices. This operand is not required for input maps.

CTRL=PRINT, CTRL=L40, CTRL=L64, CTRL=L80, and CTRL=HONEOM are options that relate exclusively to the printer functions. CTRL=PRINT must be specified if the printer is to be started; otherwise, the data is sent to the printer buffer but is not printed. CTRL=L40, CTRL=L64, CTRL=L80, and CTRL=HONEOM are mutually exclusive options that control the line length on the printer. The L40, L64, and L80 parameters force a carriage return/line feed at the end of their specified numbers of characters, respectively. CTRL=HONEOM causes the printer to honor all new-line (NL) characters and the first end-of-message (EM) character that appear in displayable fields of the data stream. It is the user's responsibility to insert these characters into displayable fields if they are to be honored. If the NL character is omitted, a carriage return/line feed occurs at the physical end of the carriage or at the right margin stop, whichever is encountered first.

When a data entry key is used by the 3270 operator, the keyboard is inhibited from entering further data. CTRL=FREEKB specifies that the keyboard should be unlocked when this map is written out.

CTRL=ALARM is used to activate the 3270 audible alarm special feature.

CTRL=FRSET (field reset) specifies that the modified data tags (MDT's) of all fields currently in the 3270 data buffer are to be reset to the "not modified" condition 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 which are written or rewritten in response to an online mapping (DFHBMS) service request.

The following macro instruction is used during offline map generation to define individual fields within a map:

```
name   DFHMDF                                                          *
          LENGTH=number,                                              *
          POS=number,                                                 *
          ATTRB=(ASKIP,PROT,UNPROT,NUM,BRT,DRK,NORM,DET,IC,FSET),     *
          JUSTIFY=(LEFT,RIGHT,BLANK,ZERO),                            *
          INITIAL='any user information',                             *
          GRPNAME=user group name
```

Fields must be defined in ascending order based on the value specified in the POS operand.

The name field of the DFHMDF macro instruction is optional. If coded, the one- to seven-character name is used by the user-written application program as a symbolic reference to the output map field and is used to pass the data both for input and output map operations.

If the name field is omitted, symbolic reference to the field by the application program is not possible. For an output map, omitting the name field is appropriate when the INITIAL operand is used to specify field contents. An input map field description with no field name causes no symbolic storage definition entry to be generated for the field; this prevents any access to the field by the application program.

All field names and group names specified when defining fields for a symbolic storage definition (DFHMDI TYPE=DSECT) are suffixed by CICS with an "I" if MODE=IN specified and an "O" if MODE=OUT is specified. The entire name, including suffix, must be used within the application program to reference the fields, irrespective of the programming language used.

LENGTH: This operand is used to specify the length (1 to 256 bytes) of the individual field. Although an attribute byte is associated with each field, its length is not included in the LENGTH value.

POS: This operand is used to specify up to 1920 individually addressable character locations (0-1919) possible in a map. The value specified is the location of the attribute byte that precedes each field. For input fields, the POS=number specification should be the same as that specified for the keyable or detectable field generated in the output map (which is the source of this input field).

The location of the data on the device depends on the model of the 3270 being used. For the 480-character 3270, any POS=number specification that is an integral multiple of 40 results in a new line; any POS=number specification for the 480-character 3270 greater than 479 produces unpredictable output. For a 1920-character 3270, a POS=number specification that is an integral multiple of 80 results in a new line.

For printers, new lines are determined by the CTRL specification of the DFHMDI macro instruction; the POS specification controls only those character positions that are in the buffer.

All DFHMDF macro instructions must be coded in ascending order based on the value specified in the POS operand. Otherwise, fields may be omitted during input or output mapping operations.

ATTRB: This operand is used to specify the device-dependent characteristics and attributes applicable to individual fields. Applicable keyword parameters are ASKIP, PROT, UNPROT, NUM, BRT, DET, DRK, IC, NORM, and FSET. If no parameters are specified, ASKIP and NORM are assumed. If any parameter is specified, UNPROT, NORM, and alphameric are assumed for any field unless overridden by a specified parameter.

The ASKIP, PROT, UNPROT, and NUM attributes are used to describe the capability of the field to receive data. Fields with the ASKIP attribute imply the PROT attribute; the cursor automatically skips over the field. Data cannot be keyed into an ASKIP field. The PROT attribute is similar to the ASKIP attribute except that no automatic skipping of the field by the cursor occurs. The UNPROT attribute allows a field to be keyed; the NUM attribute ensures that the data entry keyboard is set to numeric shift for this field unless the operator pressed the alpha shift key. The NUM attribute also prevents entry of non-numeric data if the keyboard numeric lock feature is installed. The ASKIP, PROT, and UNPROT attributes are mutually exclusive.

The BRT, NORM, and DRK attributes specify high intensity, normal intensity, and non-display/non-print respectively. These attributes are mutually exclusive.

The DET attribute specifies that a field is potentially pen-detectable. As required for a 3270 pen-detectable field, the first data character must be a "?", a ">", or a blank. See "IBM 3270 Information Display System Component Description", form number GA27-2749, for the functions of these characters and other requirements of pen-detectable fields. Note that a field which has the BRT attribute is always potentially pen-detectable to the 3270, but is not recognized as such by the Basic Mapping Support unless the DET attribute is also specified. DET and DRK are mutually exclusive options. For input map fields, DET and NUM are the only valid options (all others are ignored).

An input DET field has a one-byte reserved data area which is set to X'00' when the field is unselected, or X'FF' when the field is selected. No other data is supplied.

The IC attribute indicates that the cursor is to be placed in the first position of this field. The IC attribute for the last field in the map for which it is specified is the one that takes effect. If the IC attribute is not specified for any fields, the default location is zero. Specifying the IC attribute with the ASKIP attribute or PROT attribute causes the cursor to be placed in an unkeyable field. The FSET (field set) attribute specifies that this field should have the modified data tag (MDT) set on when the field is sent out to the 3270. This causes the 3270 to treat the field as if it had been modified, meaning that on a subsequent read from the terminal, this field is read in even though the field may not have been modified. This facility is useful for providing duplicate information or constant information from the screen as input. Note that the MDT remains on until the field is rewritten or until an input/output map request (for example, DFHMDI CTRL=FRSET or DFHBMS CTRL=FRSET) causes MDT's to be reset.

JUSTIFY: This operand is used to specify the format of an input field. Normally, input fields are left-justified (JUSTIFY=LEFT), and, if the data area is not filled, trailing blanks are inserted (JUSTIFY=BLANK). However, numeric fields are often easier to manipulate if they are right-justified (JUSTIFY=RIGHT) and are preceded by zeros (JUSTIFY=ZERO). Note that LEFT and RIGHT are mutually exclusive parameters, as are BLANK and ZERO.

In the absence of certain of these parameters, the following is assumed:

SPECIFIED	ASSUMED
LEFT	BLANK
RIGHT	ZERO
BLANK	LEFT
ZERO	RIGHT

If the JUSTIFY operand is omitted, the following is assumed:

SPECIFIED	ASSUMED
NUM attribute	RIGHT,ZERO
Other than NUM attribute	LEFT,BLANK

INITIAL: This operand is used only in output map field descriptions to supply constant or default data for a field. If the name field of the DFHMDF macro instruction is not used, the user-written application program cannot access the output field map to alter the data or its attributes. If the name field of the DFHMDF macro instruction is used, the INITIAL data is always in the field but is overlayed by any data supplied by the user under this name field specification. For fields with the DET attribute, initial data that begins with a blank character, "?", or ">" should be supplied.

GRPNAME: This operand is used to generate symbolic storage definitions and to combine individual fields under one group name by specifying the group name for each of the fields in the group. The fields

composing a group must be consecutive (contiguous). Each DFHMDF macro instruction that names a field that is to belong to the group must include the GRPNAME operand specifying the common group name. For example:

```
MAPX     DFHMDI TYPE=DSECT,...

FLD1     DFHMDF                                                  *
            LENGTH=20,                                           *
            POS=10,...

GRPFLDA  DFHMDF                          LOCATE FIRST FIELD OF GROUP  *
            LENGTH=20,                                           *
            POS=81,                                              *
            GRPNAME=GRP1,...

GRPFLDB  DFHMDF                          LOCATE SECOND FIELD OF GROUP *
            LENGTH=20,                                           *
            POS=101,                                             *
            GRPNAME=GRP1,...

FLD2     DFHMDF                                                  *
            LENGTH=15,                                           *
            POS=161,...
```

In the above example, if DFHMDI TYPE=DSECT,MODE=IN is specified, the generated names are FLD1I, GRP1I, GRPFLDAI, etc,; if DFHMDI TYPE=DSECT,MODE=OUT is specified, the generated names are FLD1O, GRP1O, GRPFLDAO, etc. These generated names must be used within the application program to reference the fields.

A group of fields exists as a single field on the 3270; the individual field names (specified in the name field of the DFHMDF macro instruction) provide the user with access to portions of the complete 3270 field.

Fields coded without a group name entry are considered to be group fields consisting of a single entry.

An entry with a group name but no field name results in an error condition.

ONLINE MAP INVOCATION

Online mapping operations are requested by issuing the DFHBMS macro instruction. Basic Mapping Support (BMS) performs any required input/output operations via Terminal Control. The data returned from an input mapping operation is in TIOA format; the address of this TIOA is found at TCTTEDA.

For an output mapping operation, if DATA=YES or DATA=ONLY, the application programmer must first have obtained, via Storage Control, a TIOA large enough to contain the symbolic storage definition of the map being used. Any fields not requiring data 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 instruction. Before issuing the DFHBMS macro instruction, the address of the TIOA must have been placed at TCTTEDA.

The following BMS services are available through use of the DFHBMS macro instruction:

1. Input - BMS performs a READ/WAIT via Terminal Control and maps the data (under control of the input map) into TIOA format.

2. Output - BMS converts the TIOA to 3270 data stream format, merges fields from the map (if desired), schedules a write operation, and waits for completion (if requested).

3. Map - BMS maps, upon request, any 3270 input TIOA into a mapped TIOA.

The following operands can be included in the DFHBMS macro instruction:

```
DFHBMS TYPE=(IN,OUT,ERASE,WAIT,SAVE,MAP),                    *
       MAP='map name',YES,                                   *
       DATA=NO,YES,ONLY,                                     *
       CTRL=(PRINT,L40,L64,L80,HONEOM,FREEKB,ALARM,FRSET),   *
       CURSOR=number,YES,                                    *
       MAPADR=symbolic address,YES
```

TYPE: This operand is used to specify the type of mapping operation and to request screen erase and/or write synchronization in connection with an output operation.

TYPE=IN specifies an input mapping operation. Input is accepted from the terminal via 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.

After return is made to the application program from this macro operation, the fields entered are available to the application program under the symbolic names specified in the name fields of the input map DFHMDF macro instructions, with the letter "I" suffixed to correspond to the name CICS generates in the DSECT expansion.

TYPE=OUT specifies an output mapping operation. The output TIOA (addressed at TCTTEDA by the user) is converted to a 3270 data stream and is written to the terminal.

TYPE=(ERASE,OUT) is used to specify that the screen is to be erased before the output map is transmitted.

TYPE=(OUT,WAIT) is used to specify that the output operation is to be synchronized with the completion of a write request. Since a wait is automatically issued in response to a read request, the TYPE=(IN,WAIT) specification is unnecessary.

Note: Multiple writes without wait may cause unpredictable results.

TYPE=MAP specifies an operation similar to an input mapping operation (TYPE=IN) except that a Terminal Control read is not performed. If TYPE=MAP is specified, the user must have placed at TCTTEDA the address of the input TIOA containing 3270 data to be mapped. An example is the initial TIOA given to a transaction upon entering a transaction code.

TYPE=SAVE may be specified with any use of the OUT parameter to indicate that the TIOA (addressed by TCTTEDA at the time the DFHBMS macro instruction is issued) is not to be freed.

MAP: This operand is used to specify the name of the map to be used for input or output operations. The map must reside in the CICS program library and must have a corresponding entry in the Processing Program Table (PPT).

MAP='map name' specifies the one- to seven-character name of the map to be used.

MAP=YES indicates that the user has placed at TCABMSMN the seven-character name of the map. If the name contains fewer than seven characters, it must be left justified and padded with blanks to seven characters.

DATA: Applicable only to output mapping operations, this operand is used to specify one of three output mapping functions: (1) write only default data, (2) merge default fields with user fields, or (3) write only user data. If this operand is not specified, DATA=NO is assumed. no user data stream to be mapped into this output map description. (The user has not specified a TIOA.) Only the initial data (and/or default data) specified for the output map fields is transmitted to the terminal.

DATA=YES indicates that data specified in the user's TIOA (the address of which is at TCTTEDA) is to be merged with the data in the output map. Data in the TIOA overrides the initial data and/or field characteristics in the output map.

DATA=ONLY specifies that no initial fields are to be written; only the data supplied in the user's TIOA is to be written. No attribute bytes are sent from the map to the terminal. Only attributes specified by the user as "fieldname.A" or "groupname.A" are transmitted.

CTRL: Used in conjunction with the TYPE=OUT operand, this optional operand is used to temporarily override control functions specified for a particular output map. This operand is effective as a temporary override only for this output request.

CTRL=PRINT, CTRL=L40, CTRL=L64, CTRL=L80, and CTRL=HONEOM are options that relate exclusively to the printer functions. CTRL=PRINT must be specified if the printer is to be started; otherwise, the data is sent to the printer buffer but is not printed. CTRL=L40, CTRL=L64, CTRL=L80, and CTRL=HONEOM are mutually exclusive options that control the line length on the printer. The L40, L64, and L80 parameters force a carriage return/line feed at the end of their specified numbers of characters. CTRL=HONEOM causes the printer to honor all new line (NL) characters and the first end-of-message (EM) character in the data stream. If the NL character is omitted, a carriage return/line feed occurs at the physical end of the carriage or at the right margin stop, whichever is encountered first.

When a data entry key is used by the 3270 operator, the keyboard is inhibited from entering further data. CTRL=FREEKB specifies that the keyboard should be unlocked when this map is written out.

CTRL=ALARM is used to activate the 3270 audible alarm special feature.

CTRL=FRSET specifies that the modified data tag is to be reset to the "not modified" condition on all fields.

CURSOR: Applicable only to output mapping operations, CURSOR=number is used to position the cursor at a particular position on the screen upon completion of a WRITE. Any integral value in the range 0-1919 may be specified, depending upon the screen size of the 3270 being used. This operand is effective as a temporary override only for this output request.

CURSOR=YES indicates that the application programmer has previously specified the desired cursor position at TCABMSCP.

MAPADR: Restricted to application programs coded in Assembler language, this optional operand is used to specify the address of a user-coded map. This operand allows maps to be coded within the user-written application program.

MAPADR=YES is used by the Assembler language programmer to indicate that the address of the map has been placed at TCABMSMA.

Note: In the case of the CICS/DOS-ENTRY system, the MAPADR operand must not specify any address within the limits of the program. Instead, the user must obtain an area of main storage via a Storage Control GETMAIN macro instruction and then move the map to this area.

Specifying Maps for 3270 Basic Mapping Support

The map used for input or output operations must be specified for BMS. If the user has placed the map in the CICS program library, the user must use the MAP='mapname' specification, or, if preferred, the user may place the seven-character name of the map at TCABMSMN and specify MAP=YES.

Assembler language programmers may "hard code" maps in their program and place the address of the map at TCABMSMA and code MAPADR=YES. If desired, the user may code MAPADR=symbolic address, where address is the label of the hard-coded map. Caution must be exercised when BMS is invoked and MAPADR is specified in the CICS/DOS-ENTRY system. (The address must be in subpool 0 to avoid rollout.)

Maps placed in the CICS program library are accessed by BMS through a Program Control LOAD. Therefore, the map name must be an entry in the Processing Program Table (PPT).

Implied Read/Write

Input and output requests result in a Terminal Control READ and WRITE, respectively. Therefore, the user is not required to code any Terminal Control macro instructions.

Nothing prevents the user from alternately coding native mode and BMS operations. If desired, BMS will map a native mode input TIOA by requesting only a MAP operation. However, for input to a non-formatted buffer with no MAP operation requested, mapping will not be performed and a NULL TIOA will be returned.

Note: The read that contains the transaction code and causes initiation of the transaction is a native 3270 data stream. The MAP request may be used to convert this TIOA to a mapped TIOA.

Using the DFHBMS Macro Instruction

Regardless of the programming language used (Assembler language, ANS COBOL, or PL/I), the same form of the DFHBMS macro instruction is used to request a mapping operation. In the case of ANS COBOL and PL/I, the CICS Preprocessor resolves the macro instruction and expands it into the statements required to invoke the mapping function.

Terminal input, which causes a task to be initiated, is stored in the task's initial TIOA as a native mode 3270 data stream. By requesting a MAP operation via DFHBMS, the application program is given the capability to map this TIOA into a particular input format. The

format of this initial input data must correspond to that of the requested map.

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. DFHBMSCA consists of a set of EQU statements in the case of Assembler language, a set of 01 statements in the case of ANS 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.

Listed below are the field attributes/printer control characters and corresponding symbolic names.

SYMBOLIC NAME	FIELD ATTRIBUTE/PRINTER CONTROL CHARACTER
DFHBMPEM	3270 Printer end of message
DFHBMPNL	3270 Printer new line symbol
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

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 instruction to obtain the desired combinations or must assume responsibility to generate new attribute combinations offline.

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 AID's) that can be used to test the value at TCTTEAID. He can obtain this set of attention identifiers by copying DFHAID into his program.

DFHAID consists of a set of EQU statements in the case of Assembler language, a set of 01 statements in the case of ANS COBOL, and DECLARE statements defining elementary character variables in the case of PL/I. Listed below are the symbolic names for the attention identifiers and the corresponding 3270 function.

SYMBOLIC NAME	3270 FUNCTION
DFHENTER	Enter key
DFHCLEAR	Clear key
DFHPEN	Immediately detectable field
DFHPA1	PA1 key
DFHPA2	PA2 key
DFHPA3	PA3 key
DFHPF1	PF1 key
.	.
.	.
.	.
DFHPF12	PF12 key

BMS TIOA Specification

Depending on the programming language used, the BMS symbolic storage definition of the TIOA must be provided in the application program as shown in the following examples. Note that mapname1, mapname2, and mapname3 in these examples are the names of modules that contain the assembly of a BMS symbolic storage definition (TYPE=DSECT).

1. Assembler language COPY statements.

```
COPY DFHTIOA
COPY mapname1
.
.
.
COPY mapname2
.
.
.
COPY mapname3
```

2. ANS COBOL COPY statements for each symbolic storage definition.

```
LINKAGE SECTION.
01 DFHBLLDS COPY DFHBLLDS.
.
.
.
01 DFHCSADS COPY DFHCSADS.
01 DFHTCADS COPY DFHTCADS.
01 DFHTIOA COPY DFHTIOA.
01 name COPY mapname1.
01 name COPY mapname2.
01 name COPY mapname3.
```

3. PL/I INCLUDE statements.

```
%INCLUDE DFHTIOA;
%INCLUDE mapname1;
%INCLUDE mapname2;
%INCLUDE mapname3;
```

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 ORG statement immediately preceding the symbolic storage definition for each map, starting with the second map. For example:

```
COPY DFHTIOA
COPY mapname1
ORG  TIOADBA
COPY mapname2
ORG  TIOADBA
COPY mapname3
.
.
.
DFHSC TYPE=GETMAIN,                                          *
      NUMBYTE=120,                                           *
      CLASS=TERMINAL,                                        *
      INITIMG=00
L     TIOABAR,TCASCSA       ESTABLISH TIOA ADDRESSABILITY
```

2. ANS COBOL 02 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 coded subsequently. For example:

```
LINKAGE SECTION.
01 DFHBLLDS COPY DFHBLLDS.
.
.
.
    02 TIOABAR PICTURE S9(8) COMPUTATIONAL.
    02 MAPBASE1 PICTURE S9(8) COMPUTATIONAL.
    02 MAPBASE2 PICTURE S9(8) COMPUTATIONAL.
    02 MAPBASE3 PICTURE S9(8) COMPUTATIONAL.
.
.
.
01 DFHTIOA COPY DFHTIOA.
01 name COPY mapname1.
01 name COPY mapname2.
01 name COPY mapname3.
.
.
.
PROCEDURE DIVISION.
.
.
.
DFHSC TYPE=GETMAIN,                                          *
      NUMBYTE=120,                                           *
      CLASS=TERMINAL,                                        *
      INITIMG=00
    MOVE TCASCSA TO TIOABAR.
    ADD 12 TIOABAR GIVING MAPBASE1.
    ADD 12 TIOABAR GIVING MAPBASE2.
    ADD 12 TIOABAR GIVING MAPBASE3.
```

3. PL/I based pointer variable (BMSMAPBR). For example:

```
DCL TIOABAA FIXED BINARY(31,0) BASED(TIOABAB);
.
.
.
%INCLUDE DFHTIOA;
%INCLUDE mapname1;  /*EACH OF THESE MAPS IS*/
%INCLUDE mapname2;  /*BASED ON THE SAME POINTER*/
%INCLUDE mapname3;  /*VARIABLE - BMSMAPBR*/
.
.
.
DFHSC TYPE=GETMAIN,                                          *
      NUMBTYE=120,                                           *
      CLASS=TERMINAL,                                        *
      INITIMG=00
TIOABAR=TCASCSA;
TIOABAB=ADDR(TIOABAR);
TIOABAA=TIOABAA+12;
BMSMAPBR=TIOABAR;
```

Examples of the Use of BMS

The examples in this section are based on a fairly simple screen exercising problem and are intended to show the results of generating symbolic storage definition maps.

In the examples, an input symbolic storage definition and an output symbolic storage definition are illustrated for each of the programming languages supported by CICS: Assembler language, ANS COBOL, and PL/I. Each of these examples is generated from the screen definition of the first example; only the initial DFHMDI entry is changed.

```
SAMPLE DFHMDI TYPE=DSECT,LANG=ASM,MODE=IN,TERM=3270,CTRL=FREEKB
       DFHMDF POS=0,LENGTH=17,INITIAL='ENTER YOUR NAME--'
NAME   DFHMDF POS=18,LENGTH=18,ATTRB=(IC,UNPROT)
       DFHMDF POS=40,LENGTH=17,INITIAL='WHAT IS THE DATE?'
MONTH  DFHMDF POS=58,LENGTH=2,INITIAL='MM',GRPNAME=DATE
DAY    DFHMDF POS=60,LENGTH=2,INITIAL='DD',GRPNAME=DATE
YEAR   DFHMDF POS=62,LENGTH=2,INITIAL='YY',GRPNAME=DATE
       DFHMDF POS=80,LENGTH=26,INITIAL='SELECT YOUR FAVORITE COLOR'
BLUE   DFHMDF POS=120,LENGTH=9,ATTRB=DET,INITIAL='?ØBLUEØØØ'
RED    DFHMDF POS=131,LENGTH=8,ATTRB=DET,INITIAL='?ØREDØØØ'
AMBER  DFHMDF POS=141,LENGTH=10,ATTRB=DET,INITIAL='?ØAMBERØØØ'
       DFHMDF POS=160,LENGTH=19,ATTRB=(PROT,BRT),
             INITIAL='NOW HIT A PF KEY...'
ERROR  DFHMDF POS=240,LENGTH=19,ATTRB=DRK,
             INITIAL='SORRY, TRY AGAIN...'
       DFHMDI TYPE=FINAL
       END
```

Example 1. Symbolic storage definition input

```
SAMPLEI DS 0C
         SPACE 2
NAMEL    DS H                        DATA LENGTH
NAMEI    DS CL18                     DATA OR FLAG
         SPACE 2
DATEL    DS H                        DATA LENGTH
DATEI    DS 0C                       GROUP DATA
         SPACE 2
         SPACE 2
MONTHI   DS CL2                      DATA
         SPACE 2
DAYI     DS CL2                      DATA
         SPACE 2
YEARI    DS CL2                      DATA
         SPACE 2
BLUEL    DS H                        DATA LENGTH
BLUEI    DS CL1                      DATA OR FLAG
         SPACE 2
REDL     DS H                        DATA LENGTH
REDI     DS CL1                      DATA OR FLAG
         SPACE 2
CLUEL    DS H                        DATA LENGTH
CLUEI    DS CL5                      DATA OR FLAG
         SPACE 2
AMBERL   DS H                        DATA LENGTH
AMBERI   DS CL1                      DATA OR FLAG
         SPACE 2
ERRORL   DS H                        DATA LENGTH
ERRORI   DS CL19                     DATA OR FLAG
* * * END OF MAP DEFINITION * * *
         DFHBMSKS
```

Example 2. Symbolic storage definition using LANG=ASM,MODE=IN specification

```
SAMPLEO DS 0C
           SPACE 2
NAMEA    DS C                          USER ATTRIBUTE
         DS C                          RESERVED
NAMEO    DS CL18                       DATA FIELD
           SPACE 2
           SPACE 2
DATEA    DS CL1                        USER ATTRIBUTE
         DS CL1                        RESERVED
DATEO    DS 0C                         GROUP START
           SPACE 2
MONTHO   DS CL2                        DATA FIELD
           SPACE 2
DAYO     DS CL2                        DATA FIELD
           SPACE 2
YEARO    DS CL2                        DATA FIELD
           SPACE 2
           SPACE 2
BLUEA    DS C                          USER ATTRIBUTE
         DS C                          RESERVED
BLUEO    DS CL5                        DATA FIELD
           SPACE 2
           SPACE 2
REDA     DS C                          USER ATTRIBUTE
         DS C                          RESERVED
REDO     DS CL5                        DATA FIELD
           SPACE 2
           SPACE 2
CLUEA    DS C                          USER ATTRIBUTE
         DS C                          RESERVED
CLUEO    DS CL5                        DATA FIELD
           SPACE 2
           SPACE 2
AMBERA   DS C                          USER ATTRIBUTE
         DS C                          RESERVED
AMBERO   DS CL5                        DATA FIELD
           SPACE 2
           SPACE 2
ERRORA   DS C                          USER ATTRIBUTE
         DS C                          RESERVED
ERRORO   DS CL19                       DATA FIELD
           SPACE 2
* * * END OF MAP DEFINITION * * *
        DFHBMSKS
```

Example 3. Symbolic storage definition using LANG=ASM,MODE=OUT specification

```
01  SAMPLEI SYNCHRONIZED.
    02  NAMEL COMP PIC S9(4).
    02 NAMEI PIC X(18).
    02  DATEL COMP PIC S9(4).
    02 DATEI.
       03 MONTHI PIC X(2).
       03 DAYI PIC X(2).
       03 YEARI PIC X(2).
    02  BLUEL COMP PIC S9(4).
    02 BLUEI PIC X(1).
    02  REDL COMP PIC S9(4).
    02 REDI PIC X(1).
    02  CLUEL COMP PIC S9(4).
    02 CLUEI PIC X(5).
    02  AMBERL COMP PIC S9(4).
    02 AMBERI PIC X(1).
    02  ERRORL COMP PIC S9(4).
    02 ERRORI PIC X(19).
```

Example 4. Symbolic storage definition using LANG=COBOL,MODE=IN specification

```
01  SAMPLEO SYNCHRONIZED.
    02  NAMEA PICTURE X.
    02  FILER PICTURE X.
    02  NAMEO PICTURE X(18).
    02  DATEA PICTURE X.
    02  FILLER PICTURE X.
    02  DATEO.
      03 MONTHO PICTURE X(2).
      03 DAYO PICTURE X(2).
      03 YEARO PICTURE X(2).
    02  BLUEA PICTURE X.
    02  FILLER PICTURE X.
    02  BLUEO PICTURE X(5).
    02  REDA PICTURE X.
    02  FILLER PICTURE X.
    02  REDO PICTURE X(5).
    02  CLUEA PICTURE X.
    02  FILLER PICTURE X.
    02  CLUEO PICTURE X(5).
    02  AMBERA PICTURE X.
    02  FILLER PICTURE X.
    02  AMBERO PICTURE X(5).
    02  ERRORA PICTURE X.
    02  FILLER PICTURE X.
    02  ERRORO PICTURE X(19).
```

Example 5. Symbolic storage definition using LANG=COBOL,MODE=OUT specification

```
DECLARE 1 SAMPLEI ALIGNED BASED (BMSMAPBR),
     2 NAMEL FIXED BINARY (15,0),
     2 NAMEI CHARACTER (18),
     2 DATEL FIXED BINARY (15,0),
     2 DATEI,
       3 MONTHI CHARACTER (2),
       3 DAYI CHARACTER (2),
       3 YEARI CHARACTER (2),
     2 BLUEL FIXED BINARY (15,0),
     2 BLUEI CHARACTER (1),
     2 REDL FIXED BINARY (15,0),
     2 REDI CHARACTER (1),
     2 CLUEL FIXED BINARY (15,0),
     2 CLUEI CHARACTER (5),
     2 AMBERL FIXED BINARY (15,0),
     2 AMBERI CHARACTER (1),
     2 ERRORL FIXED BINARY (15,0),
     2 ERRORI CHARACTER (19),
     2 FILL0030 CHARACTER (1),
/* END OF MAP DEFINITION */
```

Example 6. Symbolic storage definition using LANG=PL1,MODE=IN specification

```
DECLARE 1 SAMPLEO ALIGNED BASED (BMSMAPBR),
     2 NAMEA CHARACTER (1),
     2 FILL0008 CHARACTER (1),
     2 NAMEO CHARACTER (18),
     2 DATEA CHARACTER (1),
     2 FILL0014 CHARACTER (1),
     2 DATEO,
       3 MONTHO CHARACTER (2),
       3 DAYO CHARACTER (2),
       3 YEARO CHARACTER (2),
     2 BLUEA CHARACTER (1),
     2 FILL0029 CHARACTER (1),
     2 BLUEO CHARACTER (5),
     2 REDA CHARACTER (1),
     2 FILL0035 CHARACTER (1),
     2 REDO CHARACTER (5),
     2 CLUEA CHARACTER (1),
     2 FILL0038 CHARACTER (1),
     2 CLUEO CHARACTER (5),
     2 AMBERA CHARACTER (1),
     2 FILL0041 CHARACTER (1),
     2 AMBERO CHARACTER (5),
     2 ERRORA CHARACTER (1),
     2 FILL0047 CHARACTER (1),
     2 ERRORO CHARACTER (19),
     2 FILL0050 CHARACTER (1);
/* END OF MAP DEFINITION */
```

Example 7. Symbolic storage definition using LANG=PL1,MODE=OUT specification

PROGRAM TESTING AND DEBUGGING

Testing in the information system environment has always been difficult. The information system, including the operating system, CICS, and the user's application programs, must be responsive to many factors concurrently. The equipment configuration includes many lines and terminals through which requests for varied services are coming constantly on a random, nonscheduled basis. The precise relationship of all programs and data set (file) activity generated from the terminal inputs is never the same from one moment to the next.

Even at the simplest level of program testing, the implementer faces problems. He cannot efficiently test his program from a terminal which requires that all test data be keyed into the system each time that he requires a test shot. He cannot easily retain a backlog of proven test data and quickly test his programs through the key-driven terminal as program changes are made.

CICS allows the application programmer to begin testing his programs without requiring the use of a telecommunication device. It is possible to specify through the Terminal Control Table that sequential devices be used as terminals. At the same time, the Terminal Control Table can include references to the other terminals on the system. The sequential devices are the card reader, line printer, disk, and magnetic tape. In fact, a Terminal Control Table can include combinations of sequential devices such as: card reader and line printer, one or more disk data sets as input, one or more disk data sets as output. The same table can also include references to the other terminals on the system.

The input data must be prepared in the form that it would come from a terminal. A transaction identification must appear in the first four positions of the first input for a transaction, and, if a sequential device is being used as a terminal, a 0-2-8 punched card code or the equivalent must follow the input message. 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.

Consequently, it is possible to prepare a stream of transaction test cases to do the basic testing of a program module. As the testing progresses, the user would want to generate additional transaction streams to validate the multiprogramming capabilities of his programs or to allow different transaction test cases to be run concurrently.

User-written application programs can make use of the facilities of Dump Control and Trace Control to capture the status of the programs during testing. The Dump Control output is printed by using the CICS Dump Utility program. For a description of the Dump Control facilities, see "Dump Services".

At some point in testing, it is necessary to use the telecommunication devices to ensure that the transaction formats are satisfactory, that the terminal operational approach is satisfactory, and that the transactions can be processed on the terminal. The Terminal Control Table can be altered to contain more and different devices as the testing requirements change.

When the testing has proven that multiple transactions can be processed concurrently and the necessary data sets (actual or duplicate)

for online operation are created, the user begins testing in a controlled environment with the telecommunication devices. In the controlled environment, the business activity should represent all functions of the eventual system, but be on a smaller and a measurable scale. For example, a company whose information system will work with 15 district offices would select one district office for the controlled test. During the controlled test, all transactions, data set activity, and output activity from the system would be closely measured.

Testing is a continuing process; it is not complete when customer conversion occurs. The entire testing cycle is repeated as the applications are upgraded and new applications are added to the system.

TRACE CONTROL FUNCTIONS

The optional CICS Trace facility is designed as a debugging aid for the application programmer. This facility makes use of a Trace Table which is produced by requests for Trace Control services and which consists of standard and nonstandard entries. Standard entries are recorded in the table each time one of the following CICS macro instructions is issued by an application program or by a CICS management program:

1. DFHKC (Task Control)
2. DFHSC (Storage Control)
3. DFHPC (Program Control)
4. DFHIC (Interval Control)
5. DFHDC (Dump Control)
6. DFHFC (File Control)
7. DFHTD (Transient Data Control)
8. DFHTS (Temporary Storage Control)

Each standard entry contains a unique ID and information which will aid the application programmer in determining where the macro instruction was issued and what type of request was made to the management program. Thus, without any additional programming, the application programmer is provided with a useful tool to aid in the debugging process.

In addition, the application programmer may make direct, nonstandard entries in the Trace Table by using the DFHTR macro instruction in his application program. The user assigns his own identification and accompanying data for each trace entry. Thus, the user could define several unique trace entries and trace the logical path through a particular application or group of application programs.

Trace Control is branched to by its requesting program and executes as a service routine under the requesting program's TCA. Registers are saved and restored. Return is always made to the next sequential instruction in the requesting program once the requested service has been performed.

If the user has generated the Trace feature in his system, he may dynamically control which trace entries are to be made in the table. Trace activity is controlled by two sets of flag bytes in the CSA (CSATRMF1 and CSATRMF2) and one flag byte in the TCA (TCATRMF). The meaning of the individual bits of the flag bytes is as follows:

CSATRMF1

Bit	Meaning
0	Master Flag - if off, no trace occurs.
1	System Master Flag - if off, no system entries (ID 200-239) are traced.
2	User Master Flag - if off, no user entries (ID 0-199) are traced.
3-7	Reserved

CSATRMF2

Bit	Meaning	Trace ID
0	On to trace Task Control macro instructions.	X'F0'
1	On to trace Storage Control macro instructions.	X'F1'
2	On to trace Program Control macro instructions.	X'F2'
3	On to trace Interval Control macro instructions.	X'F3'
4	On to trace Dump Control macro instructions.	X'F4'
5	On to trace File Control macro instructions.	X'F5'
6	On to trace Transient Data Control macro instructions.	X'F6'
7	On to trace Temporary Storage Control macro instructions.	X'F7'

Bit 0 of the TCA flag byte (TCATRMF) is used only if the user master flag (X'20') is off in the CSA flag byte CSATRMF1. If the user master flag is off, only those user entries that are issued by tasks with the TCA flag on are traced.

The Trace Control macro instruction (DFHTR) is used to request any of the following services:

1. Dynamically allow the Trace facility to begin logging appropriate entries into the Trace Table.
2. Dynamically cause the Trace facility to stop logging entries into the Trace Table.
3. Dynamically cause a specified entry to be logged into the Trace Table.

The following operands can be included in the DFHTR macro instruction:

```
DFHTR   TYPE=ON,                                                *
        STYPE=SINGLE,ALL,(system symbol),SYSTEM,USER

DFHTR   TYPE=OFF,                                               *
        STYPE=SINGLE,ALL,(system symbol),SYSTEM,USER

DFHTR   TYPE=ENTRY,                                             *
        STYPE=SYSTEM,USER,                                      *
        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
```

TRACE ON FUNCTION

The ON function of Trace Control is used to dynamically allow the Trace facility to begin logging appropriate entries into the Trace Table. The application programmer invokes it by use of the

```
DFHTR TYPE=ON,                                                    *
      STYPE=SINGLE,ALL,(system symbol),SYSTEM,USER
```

macro instruction.

STYPE: Identifies which of the types of entries are to be traced. The meaning of each of the parameters is as follows:

1. SINGLE, specifies that the trace capability is to be turned on for the single transaction issuing the DFHTR macro instruction. STYPE=SINGLE has no effect unless the USER designation has been turned off.
2. ALL, specifies that the complete trace function is to be turned on.
3. System symbol, specifies one or more of the valid system functions. A special Trace Table entry is created each time one of the CICS macro instructions is issued. This parameter allows the user to selectively turn on the appropriate system macro trace facility. The valid system symbols are:

   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)

4. SYSTEM, specifies that the trace capability is to be turned on for all entries made from within CICS, excluding the CICS macro entries controlled by the CSATRMF2 flag byte.
5. USER, specifies that the trace capability is to be turned on for all user entries.

TRACE OFF FUNCTION

The OFF function of Trace Control is used to dynamically cause the Trace facility to stop logging entries into the Trace Table. The application programmer invokes this function by issuing the

```
DFHTR TYPE=OFF,                                                   *
      STYPE=SINGLE,ALL,(system symbol),SYSTEM,USER
```

macro instruction.

STYPE: Indicates which of the types of entries are not to be traced. Each of the parameters has the same meaning as when used with the DFHTR TYPE=ON macro instruction.

TRACE ENTRY FUNCTION

The ENTRY function of Trace Control is used to dynamically cause a specified entry to be logged into the Trace Table if the Trace facility has been turned on for that type of entry. The application programmer invokes this function by issuing the

```
DFHTR TYPE=ENTRY,                                              *
      STYPE=SYSTEM,USER,                                       *
      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
```

macro instruction.

STYPE: Indicates whether this entry is a CICS entry or user entry.

ID: Specifies the identification number to be used on this entry and must be coded as a self-defining term. The following range of numbers may be coded:

```
  0-199  with STYPE=USER
200-239  with STYPE=SYSTEM
```

Numbers 240-253 are reserved for system macro trace entries. 254 and 255 indicate the TYPE-ON and TYPE=OFF entries, respectively.

DATA1: Specifies the address of the data to be placed in the first data field of the table entry. If parentheses are used, the specified address is an address of an area that contains the address of the data.

RDATA1: Specifies the register whose contents are to be placed in the first data field of the table entry. If parentheses are used, the specified register contains the address of the data. RDATA1 and DATA1 are mutually exclusive.

DATA2: Similar to DATA1 except that it is used for the second data field of the Trace Table entry.

RDATA2: Similar to RDATA1 except that it is used for the second data field of the Trace Table entry.

DATA1TP: Valid only for ANS COBOL and PL/I programs, this operand specifies the format of the data to be placed in the first data field of the Trace Table entry. The default is DATA1TP=FBIN.

The applicable keyword parameters are HBIN, FBIN, CHAR, PACK, and POINTER, and are used as follows:

Specification	Data Format	Field Definition
DATA1TP=HBIN	Halfword, binary	COBOL: 9(4) COMP PL/I: BIN FIX(15)
DATA1TP=FBIN	Fullword, binary	COBOL: 9(8) COMP PL/I: BIN FIX(31)
DATA1TP=CHAR	1 to 4 characters	COBOL: X(4) PL/I: CHAR(4)
DATA1TP=PACK	1 to 4 bytes, packed decimal	COBOL: 9(7) COMP-3 PL/I: DEC FIX(7)
DATA1TP=POINTER	PL/I pointer variable	

DATA2TP: Similar to DATA1TP except that it is used for the second data field of the Trace Table entry. The default is DATA2TP=FBIN.

TRACE TABLE

The optional CICS Trace Table consists of a variable number of fixed-length entries and may be generated during system generation. It is used to trace the logical flow of transaction activity through the system. Following generation, the trace feature may be invoked during system initialization by specifying the number of Trace Table entries to be other than zero. If the Trace Table is invoked, the address of the table is placed in the CSA at CSATRTBA.

Each entry in the table is a fixed 16 bytes in length, and is aligned on a doubleword boundary. The table is used in a wrap-around manner so that when the last entry is used, the next entry is placed at the beginning of the table. The first 16 bytes of the table are a control field for the balance of the table and contain the following information:

BYTES	CONTENTS
0-3	Address of the current entry
4-7	Address of the beginning of the table
8-11	Address of the end of the table
12-15	Reserved

The format of the individual trace entry is:

BYTES	CONTENTS
0	Trace identification of entry.
1-3	Contents of register 14 at entry to the Trace program, or if the ID is X'F0' through X'F7', it is the contents of register 14 at entry to the CICS management program concerned.
4	If the Trace ID is one of the following, this field contains the type of request code as it relates to the applicable CICS management program.

Program	Trace ID
Task Control	X'F0'
Storage Control	X'F1'
Program Control	X'F2'
Interval Control	X'F3'
Dump Control	X'F4'
File Control	X'F5'
Transient Data Control	X'F6'
Temporary Storage Control	X'F7'
CICS/OS-DL/I Interface	X'F8'

5-7	Transaction identification as found in the CICS control section of the TCA. This identification is unique for each transaction.
8-11	Data field 1.
12-15	Data field 2.

The CICS Trace Table entries are indicated in Tables 1-10 which follow. (For a discussion of the CICS/OS-DL/I Interface Trace Table entries, see the section "Requesting Data Language/I Services".)

Table 1. Task Control

TRACE ID	REGISTER 14	TYPE OF REQ	TRANSACTION ID	FIELD A	FIELD B
		REQUEST CODE FROM TCATCTR			
X'F0'		X'80'	(DETACH)	Not used	Not used
		X'40'	(WAIT)	Dispatch Control Indicator TCATCDC	Event Control Address TCATCEA
		X'20'	(CHAP)	New priority TCATCDP	Not used
		X'14'	(AVAIL)	Facility Control Address	Not used
		X'12'	(SCHEDULE)	Terminal ID or AID address TCAKCTA	Transaction ID TCAKCTI
		X'11'	(Conditional ATTACH)	Facility Control Address	Transaction ID TCAKCTI
		X'10'	(ATTACH)	Facility Control Address	Transaction ID TCAKCTI
		X'08'	(RESUME)	TCA (TXA) address of resumed transaction	Not used
		X'04'	(SUSPEND)	Not used	Not used
		X'02'	(DEQUEUE)	Queue name address TCATCQA	Not used
		X'01'	(ENQUEUE)	Queue name address TCATCQA	Not used

Table 2. Storage Control

TRACE ID	REGISTER 14	TYPE OF REQ	TRANSACTION ID	FIELD A	FIELD B

REQUEST CODE FROM TCASCTR

	Bit	Condition	Byte		Not used
X'F1'	0	1=GETMAIN	0	Not used	
	1	1=FREEMAIN	1	Initialization byte for GETMAIN	
	2	1=Release all Terminal strg if bit 0=0 and bit 1=1	2-3	Requested number of bytes	
		1=Conditional GETMAIN if bit 0=1 and bit 1=0			
		1=RELEASED (used by CICS to obtain initial storage cushion if bits 0,1=0			
	3	1=Conditional Storage is to be initialized	2-3	Number of bytes released following FREEMAIN	
	4	0=Subpool 0			
		1=Subpool 1			
	5	0=Unchained storage			
		1=Chained storage			
	6	1=TCA type of storage			
	7	1=Terminal type of storage			
X'C8'	Not used		0-3	Address off main storage acquired	Storage accounting
X'C9'	Not used		0-3	Address of main storage released	Storage accounting

Table 3. Program Control

TRACE ID	REGISTER 14	TYPE OF REQ	TRANSACTION ID	FIELD A	FIELD B
		REQUEST CODE FROM TCAPCTR			
X'F2'		X'90'	(REFRESH - CICS/DOS-ENTRY only)	PPT entry address TCAPCTA	Not used
		X'84'	(Conditional LOAD)	Program name from TCAPCPI	Not used
		X'60'	(ABEND with dump)	Abend code from TCAPCAC	Not used
		X'40'	(ABEND without dump)	Not used	Not used
		X'10'	(RETURN)	Not used	Not used
		X'08'	(DELETE)	Not used	Not used
		X'04'	(LOAD)	Program name from TCAPCPI	
		X'02'	(XCTL)	Program name from TCAPCPI	
		X'01'	(LINK)	Program name from TCAPCPI	

Table 4. Interval Control

TRACE ID	REGISTER 14	TYPE OF REQ	TRANSACTION ID	FIELD A	FIELD B
		REQUEST CODE FROM TCAICTR OR TCAICRC			
X'F3'		X'1x'	(GETIME)	Return time to user address TCAICDA	Not used
		where "x" consists of the low-order four bits:			

Table 4. Interval Control (continued)

TRACE ID	REGISTER 14	TYPE OF REQ	TRANSACTION ID	FIELD A	FIELD B
			REQUEST CODE FROM TCAICTR OR TCAICRC Bit Condition 4,5 Always zero 6 0=Refresh CSA Time only 1=Return time to user 7 0=Binary format 1=Packed format		
			X'2x' (WAIT)	INTRVAL or TIME value (TCAICRT)	Not used
			X'3x' (POST)	INTRVAL or TIME value (TCAICRT)	Not used
			where "x" consists of the low-order four bits: Bit Condition 4 0=INTRVAL parameter provided 1=TIME parameter provided 5 0=No Request ID provided 1=User-provided Request ID 6,7 Always zero		
			X'4x' (INITIATE)	INTRVAL or TIME value (TCAICRT)	Transaction ID (TCAICTI)
			X'5x' (PUT)	INTRVAL or TIME value (TCAICRT)	Transaction ID (TCAICTI)
			where "x" consists of the low-order four bits: Bit Condition 4 0=INTRVAL parameter		

Table 4. Interval Control (continued)

TRACE ID	REGISTER 14	TYPE OF REQ	TRANSACTION ID	FIELD A	FIELD B
			REQUEST CODE FROM TCAICTR OR TCAICRC		
			provided 1=TIME parameter provided		
			5 0=No Request ID provided 1=User-provided Request ID		
			6 Always zero		
			7 0=Non-terminal destination 1=Terminal destination		
			X'8x' (GET)	User-provided data address	Not used
			where "x" consists of the low-order four bits:		
			Bit Condition		
			4,5 Always zero		
			6 0=User-provided data address 1=Return data address to user		
			7 Always zero		
			X'90' (RETRY)	Not used	Not used
			X'Fx' (CANCEL)	Request ID	(TCAICQID)
			where "x" consists of the low-order four bits:		
			Bit Condition		
			4 Always Zero		
			5 0=No Request ID provided 1=User-provided Request ID		
			6,7 Always zero		

Table 5. Dump Control

TRACE ID	REGISTER 14	TYPE OF REQ	TRANSACTION ID	FIELD A	FIELD B
				REQUEST CODE NOT USED (see field A)	CONTENTS FROM TCADCTR (Bytes 2-3 not used)	
X'F4'				TRANSACTION	X'FE00'	Abend code
				CICS	X'00FF'	
				COMPLETE	X'FEFF'	
				PARTIAL		
				TCA	X'0000'	
				SEGMENT	X'0100'	
				TRANSACTION	X'0400'	
				TERMINAL	X'0800'	
				PROGRAM	X'2000'	

Table 6. File Control

TRACE ID	REGISTER 14	TYPE OF REQ	TRANSACTION ID	FIELD A	FIELD B
		REQUEST CODE FROM TCAFCTR OR TCAFCRC			
X'F5'		X'80' (GET)			
		X'84' (GET W/UPDATE)			
		X'40' (PUT)			
		X'44' (PUT W/NEWREC)			
		X'20' (GETAREA)			
		X'28' (GETAREA W/INITIMG)			
		X'10' (RELEASE or ESETL)	Data set name from TCSFCDI fill fields A and B		
		X'C0' (OPEN)			
		X'E0' (CLOSE)			
		X'F0' (LOCATE)			
		X'A0' (SETL)			
		X'B0' (GETNEXT)			
		X'A4' (RESETL)			

Table 7. Transient Data Control

TRACE ID	REGISTER 14	TYPE OF REQ	TRANSACTION ID	FIELD A	FIELD B
		REQUEST CODE FROM TCATDTR OR TCATDRC			
X'F6'		X'80'	(GET)	Not used	Not used
		X'40'	(PUT)	Data address from TCATDAA	Destination ID from TCATDDI
		X'20'	(FEOV)	Not used	Not used
		X'10'	(LOCATE)	Not used	Not used
		X'04'	(PURGE)		
		X'88'	(GET)	Issued by the Asynchronous Transaction Control program (DFHATP)	
		X'48'	(PUT)		

Table 8. Temporary Storage Control

TRACE ID	REGISTER 14	TYPE OF REQ	TRANSACTION ID	FIELD A	FIELD B
		REQUEST CODE FROM TCATSTR OR TCATSRC			
X'F7'		X'80'	(GET)		
		X'90'	(GET ADDRESS SUPPLIED)		
		X'40'	(PUT)	Data identification from TCATSDI	
		X'48'	(PUT IN MAIN)		
		X'20'	(RELEASE)		

Table 9. Trace Control

TRACE ID	REGISTER 14	TYPE OF REQ	TRANSACTION ID	FIELD A	FIELD B

REQUEST CODE

REQUEST CODE			
X'FD'	Not used	Number of repeated entries (packed decimal) in Trace Table	
		Byte Contents	Byte Contents
X'FE' (Trace turn on)		0 CSATRMF1	0 TCATRTR
		1 CSATRMF2	1 Reserved
X'FF' (Trace turn off)		2 TCATRMF	2 Reserved
		3 RESERVED	3 Reserved

Table 10. System Termination

TRACE ID	REGISTER 14	TYPE OF REQ	TRANSACTION ID	FIELD A	FIELD B

REQUEST CODE

X'EF'	Not used	Not used	Not used

Table 11. CICS-DL/I Interface

TRACE ID	REGISTER 14	TYPE OF REQ	TRANSACTION ID	FIELD A	FIELD B
X'F8'		REQUEST CODE from TCAFCTR		CALL type from TCADLLAN	PCB address from TCADLPCB

Bit 0 — Off - DFHFC
On - CALL or CALLDLI

Bits 1-2 — 00 - Assembler language
01 - ANS COBOL
10 - PL/I

Bits 3-6 — Not used

Bit 7 — On - Storage was acquired to build CALLDLI parameter list or SSA list in DFHFC macro instruction

APPENDIX A: EXECUTABLE CICS PROGRAM EXAMPLES

This section contains an executable application program that performs a limited message switching function; that is, data collection, message entry, and message retrieval. The source coding is written in Assembler language, ANS COBOL, and PL/I.

```
*********************************************************************
      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 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    Y(0)
         DC    X'15'                    NEW LINE SYMBOL CONSTANT
```

```
         DC    08X'17'                  HARD COPY TERM IDLE CHARACTERS
         DC    C'DESTINATION IDENTIFICATION ERROR - PLEASE RESUBMIT'
         DC    X'15'                    NEW LINE SYMBOL CONSTANT
MCPDEML  EQU   *-MCPDIEM                TERMINAL MESSAGE TOTAL LENGTH
*********************************************************************
*********************************************************************
*                      D A T A   C O L L E C T I O N                *
*********************************************************************
DCPDCAML DC    Y(L'DCPDCAMD)            DATA COLL ACKNOWLEDGEMENT LEN
         DC    H'0'
DCPDCAMD DC    C' DATA COLLECTION HAS BEEN REQUESTED AND IS ABOUT TO BE*
               GIN  '                   DATA COLLECTION ACKNOWLEDGEMENT
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    Y(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    Y(0)
         DC    X'15'                    NEW LINE SYMBOL CONSTANT
         DC    08X'17'                  HARD COPY 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 ACKNOWLEDGEMENT LNGTH
         DC    H'0'
MEPMEAMD DC    C' YOUR MESSAGE HAS BEEN RECEIVED AND DISPATCHED TO THE *
               DESIGNATED DESTINATION  ' MESSAGE ENTRY ACKNOWLEDGEMENT
*********************************************************************
         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    Y(0)
         DC    X'15'                    NEW LINE SYMBOL CONSTANT
         DC    08X'17'                  HARD COPY 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    Y(0)
         DC    X'15'                    NEW LINE SYMBOL CONSTANT
         DC    08X'17'                  HARD COPY 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
```

```
*************************************************************************
* * *                    I M P E R A T I V E S                      * * *
*************************************************************************
* *                                                                   * *
*************************************************************************
         DS    0D                        STORAGE ALIGNMENT - DOUBLE WORD
         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'CSDC',TIOATID          COMPARE TRANSACTION IDENT
         BE    ALPDCPN                   GO TO DATA COLLECTION PROG IF =
         CLC   =C'CSME',TIOATID          COMPARE TRANSACTION IDENT
         BE    ALPMEPN                   GO TO MESSAGE ENTRY PROG IF =
         CLC   =C'CSMR',TIOATID          COMPARE TRANSACTION IDENT
         BE    ALPMRPN                   GO TO MESSAGE RETRIEVAL PROG
         DFHPC TYPE=ABEND,               DFHPC - TYPE = ABEND           *
               ABCODE=AAPT               DFHPC - ABCODE = AAPT
         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'CSDC'       MOVE TEMP STRG DATA IDENT
         MVC   TCATSDI+4(4),TCTTETI      MOVE TEMP STRG DATA IDENT
         DFHTS TYPE=GET,                 DFHTS - TYPE = GET             *
               TSDADDR=TWATSRL,          DFHTS - T S DATA ADDR = TWATSRL*
               NORESP=DCPRRNR,           DFHTS - NORMAL RESP = DCPRRNR  *
               RELEASE=YES               DFHTS - RELEASE = YES
         DFHPC TYPE=ABEND,               DFHPC - TYPE = ABEND           *
               ABCODE=ADCR               DFHPC - ABCODE = ADCR
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
DCPFRNR  EQU   *                         RESUME REQUEST NORMAL RESPONSE
         DFHTC TYPE=(WRITE)              DFHTC - TYPE = WRITE
         DFHTC TYPE=(READ)               DFHTC - TYPE = READ
*************************************************************************
         SPACE 4                         LISTING CONTROL CARD - SPACE 4
DCPTEWN  DS    0H                        TERMINAL EVENT WAIT ENTRY
         DFHTC TYPE=(WAIT)               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'CSDC'      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,                DFHTS - TYPE = PUT             *
               TSDADDR=TWATSRL,         DFHTS - T S DATA ADDR = TWATSRL*
               STORFAC=MAIN             DFHTS = STOR FAC = MAIN
         B     DCPSRAB                  GO TO AUX STRG FACILITY BYPASS
DCPSRMB  EQU   *                        MAIN STORAGE FACILITY BYPASS
         DFHTS TYPE=PUT,                DFHTS - TYPE = PUT             *
               TSDADDR=TWATSRL,         DFHTS - T S DATA ADDR = TWATSRL*
               STORFAC=AUXILIARY        DFHTS - STOR FAC = AUXILIARY
DCPSRAB  EQU   *                        AUX STORAGE FACILITY BYPASS
         DFHTS TYPE=CHECK,              DFHTS - TYPE = CHECK           *
               NORESP=DCPSRNR           DFHTS - NORMAL RESP = DCPSRNR
         DFHPC TYPE=ABEND,              DFHPC - TYPE = ABEND           *
               ABCODE=ADCS              DFHPC - ABCODE = ADCS
DCPSRNR  EQU   *                        SUSPEND REQUEST NORMAL RESPONSE
         MVC   TIOATDL(DCPSRAL),DCPSRAM MOVE TERMINAL MESSAGE TO OUTPUT
         DFHTC TYPE=(WRITE)             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)              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,                TYPE OF REQ - PUT TRANS DATA   *
               TDADDR=TIOATDL,          TRANSIENT DATA ADDRESS         *
               NORESP=DCPNRCN,          NORMAL RESP CODE ENTRY ADDRESS *
               IDERROR=DCPDIEN          DESTINATION IDENT ERROR ENTRY
         DFHPC TYPE=ABEND,              DFHPC - TYPE = ABEND           *
               ABCODE=ADCP              DFHPC - ABCODE = ADCP
*********************************************************************
DCPNRCN  DS    0H                       NORMAL RESP CODE ENTRY ADDRESS
         ST    TIOABAR,TCASCSA          STORE TERM I / O AREA ADDRESS
         DFHSC TYPE=FREEMAIN            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)             DFHTC - TYPE = WRITE
         B     RETURN                   GO TO RETURN ENTRY
*********************************************************************
         EJECT                          LISTING CONTROL CARD - EJECT
*********************************************************************
*                          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,               TYPE OF REQ - PUT TRANS DATA   *
               TDADDR=TIOATDL,         TRANSIENT DATA ADDRESS         *
               NORESP=MEPNRCN,         NORMAL RESP CODE ENTRY ADDRESS *
               IDERROR=MEPDIEN         DESTINATION IDENT ERROR ENTRY
         DFHPC TYPE=ABEND,             TYPE OF REQ - ABEND PROG CONT  *
               ABCODE=AMEP             ABNORMAL TERMINATION CODE
*********************************************************************
MEPNRCN  DS    0H                      NORMAL RESP CODE ENTRY ADDRESS
         MVC   TIOATDL((4+L'MEPMEAMD)),MEPMEAML
         DFHTC TYPE=(WRITE)            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,               DFHTD - TYPE = GET DATA        *
               NORESP=MRPNRCN,         NORMAL RESP CODE ENTRY ADDRESS *
               QUEZERO=MRPQERN,        DESTINATION QUEUE EMPTY ENTRY  *
               IDERROR=MRPDIEN         DESTINATION IDENT ERROR ENTRY
         DFHPC TYPE=ABEND,             TYPE OF REQ - ABEND PROG CONT  *
               ABCODE=AMRP             ABNORMAL TERMINATION CODE
*********************************************************************
         SPACE 2                       LISTING CONTROL CARD - SPACE 2
*********************************************************************
MRPNRCN  DS    0H                      NORMAL RESP CODE ENTRY ADDRESS
         L     TDIABAR,TCATDAA         LOAD TRANS DATA AREA ADDRESS
         DFHTC TYPE=(WAIT)             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,            DFHTC - TYPE = WRITE           *
               SAVE)                   DFHTCT - SERV REQ = SAVE AREA
         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)                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)               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)               DFHTC - TYPE = WRITE
*********************************************************************
         SPACE 4                          LISTING CONTROL CARD - SPACE 4
RETURN   DS    0H                         RETURN TO CONTROL SYSTEM
         DFHPC TYPE=RETURN                DFHPC - TYPE = RETURN
*********************************************************************
         LTORG *                          LITERAL ORIGIN AT *
*********************************************************************
         END   CICSATP                    END OF ASSEMBLY - APPL TEST PGM
```

```
*************************************************************************
        C O B O L       E X A M P L E       P R O B L E M
*************************************************************************

       DFHCOVER
       IDENTIFICATION DIVISION.
       PROGRAM-ID.
           'CICSATP'.
       ENVIRONMENT DIVISION.
       DATA DIVISION.
       WORKING-STORAGE SECTION.
       01  MESSG1.
           02  MCPDIEM PICTURE 99 USAGE IS COMPUTATIONAL VALUE IS 60.
           02  FILL1 PICTURE 99 USAGE IS COMPUTATIONAL VALUE IS ZERO.
           02  MESSAGE1.
             03  FILL2 PICTURE X VALUE IS ' '.
             03  FILL3 PICTURE X(8) VALUE IS ALL ' '.
             03  FILL4 PICTURE X(50) VALUE IS
                 'DESTINATION IDENTIFICATION ERROR - PLEASE RESUBMIT'.
             03  FILL5 PICTURE X VALUE IS ' '.
       01  MCPDEML PICTURE 99 USAGE IS COMPUTATIONAL VALUE IS 64.
       01  DCPDCAML PICTURE 99 USAGE IS COMPUTATIONAL VALUE IS 58.
       01  DCPDCAMD PICTURE X(58) VALUE IS
           ' DATA COLLECTION HAS BEEN REQUESTED AND IS ABOUT TO BEGIN '.
       01  DCPEODML PICTURE 99 USAGE IS COMPUTATIONAL VALUE IS 73.
       01  DCPEODMD PICTURE X(74) VALUE IS ' THE DATA HAS BEEN RECEIVED
      -    'AND DISPATCHED TO THE DESIGNATED DESTINATION  '.
       01  MEPMEAML PICTURE 99 USAGE IS COMPUTATIONAL VALUE IS 77.
       01  MEPMEAMD PICTURE X(77) VALUE IS 'YOUR MESSAGE HAS BEEN RECEIV
      -    'ED AND DISPATCHED TO THE DESIGNATED DESTINATION  '.
       01  MESSG2.
           02  MRPNMMM PICTURE 99 USAGE IS COMPUTATIONAL VALUE IS 64.
           02  FILL11 PICTURE 99 USAGE IS COMPUTATIONAL VALUE IS ZERO.
           02  MESSAGE2.
             03  FILL21 PICTURE X VALUE IS ' '.
             03  FILL31 PICTURE X(8) VALUE IS ALL ' '.
             03  FILL41 PICTURE X(54) VALUE IS 'THERE ARE NO MORE MESSAG
      -      'ES QUEUED FOR THIS DESTINATION'.
             03  FILL51 PICTURE X VALUE IS ' '.
       01  MRPNMML PICTURE 99 USAGE IS COMPUTATIONAL VALUE IS 68.
       01  MESSG3.
           02  MRPNMQM PICTURE 99 USAGE IS COMPUTATIONAL VALUE IS 59.
           02  FILL12 PICTURE 99 USAGE IS COMPUTATIONAL VALUE IS ZERO.
           02  MESSAGE3.
             03  FILL22 PICTURE X VALUE IS ' '.
             03  FILL32 PICTURE X(8) VALUE IS ALL ' '.
             03  FILL42 PICTURE X(49) VALUE IS
                 'THERE ARE NO MESSAGES QUEUED FOR THIS DESTINATION'.
             03  FILL52 PICTURE X VALUE IS ' '.
       01  MRPNQML PICTURE 99 USAGE IS COMPUTATIONAL VALUE IS 63.
       LINKAGE SECTION.
       01  DFHBLLDS COPY DFHBLLDS.
           02  TCTTEAR PICTURE S9(8) USAGE IS COMPUTATIONAL.
           02  TIOABAR PICTURE S9(8) USAGE IS COMPUTATIONAL.
           02  TDIABAR PICTURE S9(8) USAGE IS COMPUTATIONAL.
       01  DFHCSADS COPY DFHCSADS.
       01  DFHTCADS COPY DFHTCADS.
           02  TWATDDI PICTURE X(4).
           02  TWAREAI PICTURE X(4).
           02  TWAQEMCI PICTURE S9 USAGE IS COMPUTATIONAL.
       01  DFHTCTTE COPY DFHTCTTE.
       01  DFHTIOA COPY DFHTIOA.
           02  TIOADATA.
             03  FILLER PICTURE X(80).
           02 FILLER REDEFINES TIOADATA.
```

```
      03 EODTEST PICTURE X(3).
    02  FILLER REDEFINES TIOADATA.
      03  TIOATID PICTURE X(4).
      03  FILLER PICTURE X.
      03  TIOADID.
        04  FILLER PICTURE X(4).
      03  FILLER REDEFINES TIOADID.
        04  TIOARAI1 PICTURE X(3).
      03  FILLER PICTURE X.
      03  TIOARAI2.
        04  FILLER PICTURE X(3).
      03  FILLER REDEFINES TIOARAI2.
        04  TIOAMBA PICTURE X.
01  DFHTDIA COPY DFHTDIA.
    02  TDIADBA PICTURE X(80).
PROCEDURE DIVISION.
ATPIPIN.
    MOVE CSACDTA TO TCACBAR.
    MOVE TCAFCAAA TO TCTTEAR.
    MOVE TCTTEDA TO TIOABAR.
    IF TIOATID = 'BSDC' GO TO ALPDCPN.
    IF TIOATID = 'BSME' GO TO ALPMEPN.
    IF TIOATID = 'BSMR' GO TO ALPMRPN.
  DFHPC TYPE=ABEND,                                              *
        ABCODE=AAPT
    NOTE   DATA COLLECTION PROGRAM ***.
ALPDCPN.  MOVE TIOADID TO TWATDDI.
    MOVE DCPDCAML TO TIOATDL.
    MOVE DCPDCAMD TO TIOADATA.
  DFHTC TYPE=(WRITE,READ,WAIT)
DCPTEWN.
    MOVE TCTTEDA TO TIOABAR.
    IF EODTEST = 'EOD' GO TO DCPEXIT.
    MOVE TWATDDI TO TCATDDI.
    MOVE ZEROES TO TCTTEDA.
  DFHTC TYPE=(READ,WAIT)
    ADD 4 TO TIOATDL.
  DFHTD TYPE=PUT,                                                *
        TDADDR=TIOATDL,                                          *
        NORESP=DCPNRCN,                                          *
        IDERROR=DCPDIEN
  DFHPC TYPE=ABEND,                                              *
        ABCODE=ADCP
DCPNRCN.
    MOVE TIOABAR TO TCASCSA.
  DFHSC TYPE=FREEMAIN
    GO TO DCPTEWN.
DCPEXIT.
    MOVE DCPEODML TO TIOATDL.
    ADD 4 TO TIOATDL.
    MOVE DCPEODMD TO TIOADATA.
  DFHTC TYPE=WRITE
    GO TO RETURN1.
    NOTE  MESSAGE ENTRY PROGRAM ***.
ALPMEPN.
    MOVE TIOADID TO TCATDDI.
    MOVE TCTTETI TO TIOATID.
    ADD 4 TO TIOATDL.
  DFHTD TYPE=PUT,                                                *
        TDADDR=TIOATDL,                                          *
        NORESP=MEPNRCN,                                          *
        IDERROR=MEPDIEN
  DFHPC TYPE=ABEND,                                              *
        ABCODE=AMEP
MEPNRCN.
```

```
     MOVE MEPMEAML TO TIOATDL.
     ADD 4 TO TIOATDL.
     MOVE MEPMEAMD TO TIOADATA.
   DFHTC TYPE=WRITE
     GO TO RETURN1.
     NOTE  MESSAGE RETRIEVAL PROGRAM ***.
ALPMRPN.
     MOVE TIOARAI2 TO TWAREAI.
     MOVE TCTTETI TO TWATDDI.
     IF TIOARAI1 NOT EQUAL 'ALL' GO TO MRPAI1B.
     MOVE TIOARAI11 TO TWAREAI.
     GO TO MRPDEBN.
MRPAI1B.
     IF TIOADID EQUAL ' ' 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=AMRP
MRPNRCN.
     MOVE TCATDAA TO TDIABAR.
     MOVE TDIAIRL TO TIOATDL.
     MOVE TDIADBA TO TIOADATA.
     SUBTRACT 4 FROM TIOATDL.
   DFHTC TYPE=(WRITE,WAIT,SAVE)
     IF TWAREAI NOT EQUAL 'ALL' GO TO RETURN1.
     MOVE 255 TO TWAQEMCI.
     GO TO MRPGTDN.
MRPQERN.
     IF TWAQEMCI EQUAL 255 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
```

```
*************************************************************************
          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);
    DECLARE 1 MCPDEML BINARY FIXED (15) INITIAL (60);
    DECLARE 1 MCPDIEM CHAR(60) INITIAL ('      DESTINATION IDENTIFI
    CATION ERROR - PLEASE RESUBMIT ');
    DECLARE 1 DCPDCAML BINARY FIXED (15) INITIAL (59);
    DECLARE 1 DCPDCAMD CHAR(59) INITIAL (' DATA COLLECTION HAS BEEN RE
    QUESTED AND IS ABOUT TO BEGIN  ');
    DECLARE 1 DCPEODML BINARY FIXED (15) INITIAL (74);
    DECLARE 1 DCPEODMD CHAR (74) INITIAL (' THE DATA HAS BEEN RECIEVED
    AND DISPATCHED TO THE DESIGNATED DESTINATION  ');
    DECLARE 1 MEPMEAML BINARY FIXED (15) INITIAL (77);
    DECLARE 1 MEPEAMD CHAR(77) INITIAL (' YOUR MESSAGE HAS BEEN RECEIV
    ED AND DISPATCHED TO THE DESIGNATED DESTINATION  ');
    DECLARE 1 MRPNMML BINARY FIXED (15) INITIAL (64);
    DECLARE 1 MRPNMMM CHAR(64) INITIAL ('      THERE ARE NO MORE ME
    SSAGES QUEUED FOR THIS DESTINATION ');
    DECLARE 1 MRPNQML BINARY FIXED (15) INITIAL (59);
    DECLARE 1 MRPNMQN CHAR(59) INITIAL ('      THERE ARE NO MESSAGE
    S 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=AAPT
    DECLARE 1 CON1 CHAR (32) INITIAL ('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;
        DFHTC TYPE=(READ,WAIT)
          TIOATDL = TIOATDL + 4;
        DFHTD TYPE=PUT,                                                 *
              TDADDR=TIOATDL,                                           *
              NORESP=DCPNRCN,                                           *
              IDERROR=DCPDIEN
        DFHPC TYPE=ABEND,                                               *
              ABCODE=ADCP
DCPNRCN:  TCASCSA = TIOABAR;
        DFHSC TYPE=FREEMAIN
          GO TO DCPTEWN;
  DCPEXIT: TIOATDL = DCPEODML;
          TIOADATA = DCPEODMD;
        DFHTC TYPE=WRITE
          GO TO RETURN;
    DECLARE 1 CON2 CHAR (32) INITIAL ('MESSAGE ENTRY PROGRAM');
ALPMEPN:  TCATDDI = TIOADID;
          TIOATID = TCTTETI;
          TIOATDL = TIOATDL + 4;
        DFHTD TYPE=PUT,                                                 *
              TDADDR=TIOATDL,                                           *
              NORESP=MEPNRCN,                                           *
              IDERROR=MEPDIEN
        DFHPC TYPE=ABEND,                                               *
              ABCODE=AMEP
  MEPNRCN: TIOATDL = MEPMEAML;
          TIOADATA = MEPEAMD;
        DFHTC TYPE=WRITE
          GO TO RETURN;
    DECLARE 1 CON3 CHAR (32) INITIAL ('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=AMRP
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;
```

APPENDIX B: CICS MACRO INSTRUCTIONS

This section lists the CICS macro instructions used to request supervisory and data management services. These macro instructions are written in Assembler language and, as all Assembler language instructions, are written in the following format:

Name	Operation	Operands	Comments
blank or symbol	DFHxxxxx	One or more operands separated by commas	

The name field of a CICS macro instruction must be left blank if the macro instruction is used in conjunction with a high-level language (ANS COBOL or PL/I); if a label is desired for the macro instruction, it may be placed on the card preceding the macro instruction.

The operation field of a CICS macro instruction must begin before card 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 the symbolic name for the appropriate program or table. Since DFH is reserved for CICS macro instrucitons, no other statement may begin with this three-character combination.

The operand field of a CICS macro instruction contains one or more operands separated by commas. In this publication, parentheses are used to indicate those operands where more than one applicable parameter (keyword and otherwise) can be specified with a single use of the operand. Where parentheses are not used, only one parameter at a time can be specified as part of the operand; a choice must be made in the case of more than one applicable parameter. Since a blank character indicates the end of the operand field, the operand field must not contain blanks except after a comma on a continued card or after the last operand of the macro instruction. The first operand on a continuation card must begin in column 16.

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

In the following listing of CICS macro instructions, default parameters (where applicable) are indicated by an underscore. An asterisk in card column 72 indicates that the macro instruction is continued on the next card.

TASK SERVICES

```
      DFHKC TYPE=ATTACH,                                          *
            FCADDR=symbolic address,                              *
            TRANSID=name

      DFHKC TYPE=CHAP,                                            *
            PRTY=priority value
```

```
        DFHKC TYPE=WAIT,                                              *
              DCI=SINGLE,LIST,DISP,                                   *
              ECADDR=symbolic address

        DFHKC TYPE=ENQ,DEQ,                                           *
              QARGADR=symbolic address,                               *
              QARGLNG=number

        DFHKC TYPE=PURGE,NOPURGE
```

STORAGE SERVICES

```
        DFHSC TYPE=GETMAIN,                                           *
              INITIMG=number,YES,                                     *
              NUMBYTE=number,                                         *
              COND=YES or (YES,symbolic address) or                   *
                   (NO,symbolic address),                             *
              CLASS=TERMINAL,USER,TRANSDATA,TEMPSTRG

        DFHSC TYPE=FREEMAIN,                                          *
              RELEASE=ALL
```

PROGRAM SERVICES

```
        DFHPC TYPE=LINK,                                              *
              PROGRAM=name

        DFHPC TYPE=XCTL,                                              *
              PROGRAM=name

        DFHPC TYPE=LOAD,                                              *
              PROGRAM=name,                                           *
              LOADLST=NO

        DFHPC TYPE=RETURN,                                            *
              TRANSID=transaction code

        DFHPC TYPE=DELETE,                                            *
              PROGRAM=name

        DFHPC TYPE=ABEND,                                             *
              ABCODE=value,YES
```

DUMP SERVICES

```
        DFHDC TYPE=TRANSACTION,                                       *
              DMPCODE=value

        DFHDC TYPE=CICS,                                              *
              DMPCODE=value

        DFHDC TYPE=COMPLETE,                                          *
              DMPCODE=value

        DFHDC TYPE=PARTIAL,                                           *
              LIST=TERMINAL,PROGRAM,SEGMENT,TRANSACTION,              *
              DMPCODE=value
```

TERMINAL SERVICES

```
      DFHTC TYPE=(WRITE,WRITEL,READ,READL,WAIT,ERASE,SAVE,OIU,        *
            DISCONNECT,RESET,READB,COPY,ERASEAUP,CBUFF,               *
            PASSBK,TRANSPARENT,PSEUDOBIN,NOTRANSLATE),                *
            LINEADR=number,YES,                                       *
            CTLCHAR=hexadecimal number,YES,                           *
            DEST=symbolic name,YES,                                   *
            EOF=symbolic address

      DFHTC TYPE=(GET,PUT,ERASE,SAVE,TRANSPARENT,PSEUDOBIN),          *
            LINEADR=number,YES,                                       *
            CTLCHAR=hexadecimal number,YES,                           *
            DEST=symbolic name,YES,                                   *
            EOF=symbolic address

      DFHTC TYPE=(PAGE,SAVE),                                         *
            LINEADR=number,YES,                                       *
            CTLCHAR=hexadecimal number,YES,                           *
            DEST=symbolic name,YES

      DFHTC TYPE=(CONVERSE,ERASE,SAVE),                               *
            LINEADR=number,YES,                                       *
            CTLCHAR=hexadecimal number,YES,                           *
            DEST=symbolic name,YES

      DFHTC EOF=symbolic address
```

FILE SERVICES

```
      DFHFC TYPE=GET,                                                 *
            DATASET=symbolic name,                                    *
            RDIDADR=symbolic address,                                 *
            SEGSET=symbolic name,YES,ALL,                             *
            INDEX=symbolic name,YES,                                  *
            TYPOPER=UPDATE,                                           *
            RETMETH=RELREC,KEY,                                       *
            NORESP=symbolic address,                                  *
            DSIDER=symbolic address,                                  *
            SEGIDER=symbolic address,                                 *
            NOTFND=symbolic address,                                  *
            INVREQ=symbolic address,                                  *
            IOERROR=symbolic address,                                 *
            DUPDS=symbolic address,                                   *
            NOTOPEN=symbolic address

      DFHFC TYPE=PUT,                                                 *
            RDIDADR=symbolic address,                                 *
            SEGSET=YES,                                               *
            TYPOPER=NEWREC,UPDATE,                                    *
            NORESP=symbolic address,                                  *
            DUPREC=symbolic address,                                  *
            INVREQ=symbolic address,                                  *
            IOERROR=symbolic address,                                 *
            NOSPACE=symbolic address,                                 *
            NOTOPEN=symbolic address
```

```
DFHFC TYPE=GETAREA,                                          *
      DATASET=symbolic name,                                 *
      INITIMG=value,YES,                                     *
      DSIDER=symbolic address,                               *
      NORESP=symbolic address,                               *
      INVREQ=symbolic address,                               *
      NOTOPEN=symbolic address

DFHFC TYPE=RELEASE,                                          *
      INVREQ=symbolic address

DFHFC TYPE=SETL,                                             *
      DATASET=symbolic name,                                 *
      RDIDADR=symbolic address,                              *
      SEGSET=symbolic name,YES,ALL,                          *
      RETMETH=RELREC,KEY,                                    *
      NORESP=symbolic address,                               *
      DSIDER=symbolic address,                               *
      SEGIDER=symbolic address,                              *
      INVREQ=symbolic address,                               *
      NOTOPEN=symbolic address

DFHFC TYPE=GETNEXT,                                          *
      SEGSET=symbolic name,YES,ALL,                          *
      NORESP=symbolic address,                               *
      SEGIDER=symbolic address,                              *
      INVREQ=symbolic address,                               *
      IOERROR=symbolic address,                              *
      NOTOPEN=symbolic address,                              *
      ENDFILE=symbolic address

DFHFC TYPE=ESETL,                                            *
      INVREQ=symbolic address

DFHFC TYPE=RESETL,                                           *
      SEGSET=symbolic name,YES,ALL,                          *
      NORESP=symbolic address,                               *
      SEGIDER=symbolic address

DFHFC TYPE=CHECK,                                            *
      NORESP=symbolic address,                               *
      DSIDER=symbolic address,                               *
      SEGIDER=symbolic address,                              *
      NOTFND=symbolic address,                               *
      DUPREC=symbolic address,                               *
      INVREQ=symbolic address,                               *
      IOERROR=symbolic address,                              *
      DUPDS=symbolic address,                                *
      NOSPACE=symbolic address,                              *
      NOTOPEN=symbolic address,                              *
      ENDFILE=symbolic address
```

TRANSIENT DATA SERVICES

```
DFHTD TYPE=PUT,                                              *
      DESTID=symbolic name,                                  *
      TDADDR=symbolic address,                               *
      NORESP=symbolic address,                               *
      IDERROR=symbolic address,                              *
      IOERROR=symbolic address,                              *
      NOTOPEN=symbolic address,                              *
      NOSPACE=symbolic address
```

```
        DFHTD TYPE=GET,                                          *
              DESTID=symbolic name,                              *
              NORESP=symbclic address,                           *
              QUEZERO=symbclic address,                          *
              IDERROR=symbclic address,                          *
              IOERROR=symbolic address,                          *
              NOTOPEN=symbclic address

        DFHTD TYPE=FEOV,                                         *
              DESTID=symbolic name,                              *
              NORESP=symbolic address,                           *
              IDERROR=symbclic address,                          *
              NOTOPEN=symbclic address

        DFHTD TYPE=PURGE,                                        *
              DESTID=symbolic name,                              *
              IDERROR=symbclic address,                          *
              NORESP=symbclic address

        DFHTD TYPE=CHECK,                                        *
              NORESP=symbolic address,                           *
              QUEZERO=symbclic address,                          *
              IDERROR=symbolic address,                          *
              IOERROR=symbolic address,                          *
              NOTOPEN=symbclic address,                          *
              NOSPACE=symbclic address
```

TEMPORARY STORAGE SERVICES

```
        DFHTS TYPE=PUT,                                          *
              DATAID=name,                                       *
              TSDADDR=symbolic address,                          *
              STORFAC=AUXILIARY,MAIN,                            *
              NORESP=symbolic address,                           *
              INVREQ=symbolic address

        DFHTS TYPE=GET,                                          *
              DATAID=name,                                       *
              TSDADDR=symbclic address,YES,                      *
              RELEASE=YES,NO,                                    *
              NORESP=symbolic address,                           *
              IDERROR=symbclic address,                          *
              IOERROR=symbclic address

        DFHTS TYPE=RELEASE,                                      *
              DATAID=name,                                       *
              NORESP=symbolic address,                           *
              IDERROR=symbclic address

        DFHTS TYPE=CHECK,                                        *
              NORESP=symbolic address,                           *
              IDERROR=symbclic address,                          *
              IOERROR=symbclic address,                          *
              INVREQ=symbolic address
```

TIME SERVICES

```
        DFHIC TYPE=GETIME,                                       *
              FORM=BINARY,PACKED,                                *
              TIMADR=symbolic address,YES,                       *
              NORESP=symbolic address,                           *
              INVREQ=symbolic address
```

```
DFHIC TYPE=WAIT,                                                *
      INTRVAL=numeric value,YES,                                *
      TIME=numeric value,YES,                                   *
      REQID=name,YES,                                           *
      NORESP=symbolic address,                                  *
      INVREQ=symbolic address,                                  *
      EXPIRD=symbolic address

DFHIC TYPE=POST,                                                *
      INTRVAL=numeric value,YES,                                *
      TIME=numeric value,YES,                                   *
      REQID=name,YES,                                           *
      NORESP=symbclic address,                                  *
      INVREQ=symbolic address,                                  *
      EXPIRD=symbolic address

DFHIC TYPE=INITIATE,                                            *
      INTRVAL=numeric value,YES,                                *
      TIME=numeric value,YES,                                   *
      REQID=name,YES,                                           *
      TRANSID=name,                                             *
      TRMIDNT=name,YES,                                         *
      NORESP=symbclic address,                                  *
      INVREQ=symbolic address,                                  *
      TRNIDER=symbclic address,                                 *
      TRMIDER=symbclic address

DFHIC TYPE=PUT,                                                 *
      INTRVAL=numeric value,YES,                                *
      TIME=numeric value,YES,                                   *
      REQID=name,YES,                                           *
      TRANSID=name,                                             *
      TRMIDNT=name,YES,                                         *
      ICDADDR=symbolic address,YES,                             *
      NORESP=symbolic address,                                  *
      INVREQ=symbolic address,                                  *
      TRNIDER=symbclic address,                                 *
      TRMIDER=symbolic address,                                 *
      IOERROR=symbclic address

DFHIC TYPE=GET,                                                 *
      ICDADDR=symbclic address,YES,                             *
      NORESP=symbolic address,                                  *
      INVREQ=symbolic address,                                  *
      ENDDATA=symbclic address,                                 *
      NOTFND=symbolic address,                                  *
      IOERROR=symbclic address

DFHIC TYPE=RETRY,                                               *
      NORESP=symbolic address,                                  *
      INVREQ=symbolic address,                                  *
      NOTFND=symbolic address,                                  *
      IOERROR=symbolic address

DFHIC TYPE=CANCEL,                                              *
      REQID=name,YES,                                           *
      NORESP=symbolic address,                                  *
      INVREQ=symbclic address,                                  *
      NOTFND=symbolic address
```

```
        DFHIC TYPE=CHECK,                                              *
              NORESP=symbolic address,                                 *
              INVREQ=symbolic address,                                 *
              EXPIRD=symbolic address,                                 *
              TRNIDER=symbolic address,                                *
              TRMIDER=symbclic address,                                *
              IOERROR=symbclic address,                                *
              NOTFND=symbolic address,                                 *
              ENDDATA=symbolic address
```

PROGRAM TESTING AND DEBUGGING

```
        DFHTR TYPE=ON,                                                 *
              STYPE=SINGLE,ALL,(system symbol),SYSTEM,USER

        DFHTR TYPE=OFF,                                                *
              STYPE=SINGLE,ALL,(system symbol),SYSTEM,USER

        DFHTR TYPE=ENTRY,                                              *
              STYPE=SYSTEM,USER,                                       *
              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
```

3270 OFFLINE MAP BUILDING

```
mapname DFHMDI TYPE=DSECT,MAP,FINAL,                                   *
               TERM=3270,                                              *
               LANG=ASM,COBOL,PL1,                                     *
               BASE=name,                                              *
               MODE=IN,OUT,                                            *
               CTRL=(PRINT,L40,L64,L80,HONEOM,FREEKB,ALARM,FRSET)

name    DFHMDF
               LENGTH=number,                                          *
               POS=number,                                             *
               ATTRB=(ASKIP,PROT,UNPROT,NUM,BRT,DRK,NORM,DET,IC,FSET), *
               JUSTIFY=(LEFT,RIGHT,BLANK,ZERO),                        *
               INITIAL='any user information',                         *
               GRPNAME=user group name
```

3270 ONLINE MAP INVOCATION

```
        DFHBMS TYPE=(IN,OUT,ERASE,WAIT,SAVE),                          *
               MAP='map name',YES,                                     *
               DATA=NO,YES,ONLY,                                       *
               CTRL=(PRINT,L40,L64,L80,HONEOM,FREEKB,ALARM,FRSET),     *
               CURSOR=number,YES,                                      *
               MAPADR=symbolic address,YES
```

DATA LANGUAGE/I SERVICES

```
        DFHFC TYPE=(DL/I,PCB),                                         *
              PSB=psbname,symbolic name,YES,                           *
              NORESP=symbclic address,                                 *
              INVREQ=symbolic address
```

```
DFHFC TYPE=(DL/I,function),                                          *
      PCB=symbolic address,(register),                               *
      WRKAREA=symbolic address,YES,(register),                       *
      SSAS=NO,(ssacount,ssa1,ssa2,...),                              *
      SSALIST=YES,NO,symbolic address,(register),                    *
      NORESP=symbolic address,                                       *
      NOTOPEN=symbolic address,                                      *
      INVREQ=symbolic address

DFHFC TYPE=(DL/I,T),                                                 *
      NORESP=symbolic address,                                       *
      INVREQ=symbolic address

      CALLDLI  ASMTDLI,(parmcount,function,pcb,workarea,
                        segment search arguments,...) or
               CBLTDLI,(parmcount,function,pcb,workarea,
                        segment search arguments,...)
```

(CALLDLI is a special form of the CALL macro instruction for DL/I CALL's in Assembler language programs.)

APPENDIX C: CICS DUMP CODES

When abnormal conditions occur, the message

TRANSACTION xxxx ABEND xxxx AT xxxx

is sent to Transient Data destination CSMT, indicating that the identified transaction attached to the identified terminal has been abnormally terminated. The ABEND (dump) code indicates the origin or cause of the error, and may be originated by the user or by CICS. Following are the ABEND codes for abnormal terminations initiated by CICS.

Code	Detecting Program	Cause
AACA	Abnormal Condition	Invalid error code passed to DFHACP in the TCA at location TCAPCABR. A complete system dump is provided to assist in determining the problem.
AICA	Interval Control	A runaway task condition has been detected and the task is being abnormally terminated. The condition indicates a possible logical loop within the user's program.
AKCA	Task Control	Another CICS task has requested Task Control to abnormally terminate this task as a result of actions initiated by: • Terminal Abnormal Condition program (DFHTACP); the appropriate message is found at destination CSMT. • Task Termination portion of the Master Terminal facility. The Asynchronous Transaction Control program (DFHATP) terminates asynchronous tasks when: • User requests deletion of a batch via CWTR delete option while CICS is actively processing that batch; DFHATP abnormally terminates the task and purges all remaining data from the queues. • An asynchronous task tries to read more data than is available; DFHATP abnormally terminates the task.
AKCD	Task Control	Invalid code in the dispatch control indicator field. The invalid code can be found in the TCA at symbolic location TCATCDC. Valid codes (masks): X'10' Not dispatchable (not

Code	Detecting Program	Cause
		applicable to CICS/DOS-ENTRY) X'20' Dispatchable X'40' Wait on list of events X'80' Wait on single event
AKCP	Task Control	A stall condition has been detected and this task is being abnormally terminated. This task carries a code indicating it is purgeable.
AKCR	Task Control	The type of request code is invalid. The invalid code can be located in the TCA at symbolic location TCATCTR. Valid codes: X'01' Enqueue X'02' Dequeue X'04' System X'08' System X'10' Task Origination X'11' System X'12' System X'14' System X'20' Priority Change X'40' Task Wait X'80' Task Termination
AKCS	Task Control	The request exceeds available Subpool 1 storage. CICS/DOS-ENTRY only.
APCB	Program Control	An attempt was made to execute a PL/I program but the proper support was not included in DFHSAP. For example, PL/I F level execution attempted but support generated only for PL/I Optimizing Compiler.
APCC	Program Control	An attempt was made to execute an ANS COBOL program but ANS COBOL support was not generated in Program Control.
APCI	Program Control	An attempt was made to execute a PL/I program but PL/I support was not generated in Program Control.
APCL	Program Control	There is insufficient main storage available for the requested program.
APCP	Program Control	An error occurred on the read of a requested program from the library.
APCR	Program Control	Task request for service is invalid. The invalid code can be located in the TCA at TCAPCTR. Valid Codes: X'01' LINK X'02' XCTL X'04' LOAD X'08' DELETE X'10' RETURN X'40' ABEND X'60' ABEND with DUMP

Code	Detecting Program	Cause
		X'90' RETURN from Task Control pgm
APCT	Program Control	A task issued a request for a program which is not in the PPT. The invalid program ID is in the TCA at TCAPCPI.
APIA	Program Interrupt	A program check has occurred during the subject task execution. The PSW at the time of interrupt is saved in the task's TCA.
ASCR	Storage Control	The request for service is invalid. Valid codes: X'20' Released Storage X'40' Release Storage X'80' Acquire Storage
ASCT	Storage Control	The request exceeds available Subpool 1 storage. CICS/DOS-ENTRY only.
ATDI	Transient Data	The type of destination code is invalid. The invalid code can be located in the DCT at symbolic location TDDCTDT. Valid Codes: X'20' Indirect X'40' Extrapartition X'80' Intrapartition X'90' Intrapartition with automatic transaction initiation
ATDT	Transient Data	Request for service is invalid. The invalid code is in the TCA and can be located at TCATDTR. Valid codes: X'04' Purge destination X'08' Destination entry address passed to the Transient Data Control program X'10' Locate Destination Control Table (DCT) entry X'20' Forced end of volume on extrapartition data set X'40' Output service on intrapartition data set X'80' Input service on intrapartition data set
BMIP	Basic Mapping Support	An input mapping request was issued and the map provided was for output.
BMOP	Basic Mapping Support	An output mapping request was issued and the map provided was for input.
BMTT	Basic Mapping Support	A request was made for 3270 mapping support and the device is not a 3270.
DLDY	DL/I Interface	A DL/I CALL was issued under CICS/OS, but the DL/I Interface dummy program was loaded at system initialization.

Code	Detecting Program	Cause
DLIA	DL/I Interface	An irrecoverable error occurred during execution of the CICS-DL/I Interface program under CICS/OS. The DLIA code is returned to all transactions from which DL/I CALL's are subsequently issued.
DLPA	DL/I Interface	A DL/I abend (or pseudo abend) occurred during transaction processing under CICS/OS. The abend code is found in the TCA at TCADLECB.

System Action: In addition to the dump services requested by application programs, CICS also requests dumps for abnormal conditions and places specific dump codes in the dumps for ready identification.

Action: Analyze the error condition indicated by the abend code.

APPENDIX D: 3270 MAP GENERATION AND ASSEMBLY ERROR MESSAGES

This section contains a listing of error messages applicable to CICS Basic Mapping Support (BMS) for the 3270 Information Display System. The severity of program assembly errors is indicated by codes 4, 8, 12, and 16; codes 4 and 8 indicate an error condition that might not prevent program execution, while codes 12 and 16 indicate an error condition so severe that program execution is impossible.

DFHBM0001 TYPE IS NOT VALID; DSECT ASSUMED

Severity: 12

Meaning: The DFHMDI TYPE=parameter specification is invalid. CICS assumes TYPE=DSECT and continues to analyze the map.

Action: Supply a valid DFHMDI TYPE=parameter specification and reassemble.

DFHBM0002 INVALID LANG OPERAND; ASM IS ASSUMED.

Severity: 4

Meaning: The DFHMDI LANG=parameter specification is invalid. CICS assumes LANG=ASM and continues to analyze the map.

Action: Supply a valid DFHMDI LANG=parameter specification and reassemble.

DFHBM0003 MODE INVALID; OUT IS ASSUMED

Severity: 12

Meaning: The DFHMDI MODE=parameter specification is invalid. CICS assumes MODE=OUT and continues to analyze the map.

Action: Supply a valid DFHMDI MODE=parameter specification and reassemble.

DFHBM0005 CONFLICTING PRINTER FORMATS; HONEOM ASSUMED

Severity: 4

Meaning: The DFHMDI CTRL=parameter specification includes more than one of the parameters HONEOM, L40, L64, and L80. CICS assumes CTRL=HONEOM.

Action: Supply required printer format specification via the CTRL operand and reassemble, or accept the default.

DFHBM0006 INVALID CTRL OPERAND IS REJECTED

Severity: 12

Meaning: The DFHMDI CTRL=parameter specification is invalid. CICS rejects the option specified and continues to analyze the map.

Action: Check coding of CTRL options against macro description and reassemble the map.

DFHBM0007 ONLY 3270 IS VALID. ASSUMED.

Severity: 4

Meaning: The DFHMDI TERM=parameter specification specifies a terminal other than the 3270. CICS assumes TERM=3270 and continues to analyze the map.

Action: TERM=3270 is the only valid specification. If omitted, the default is TERM=3270.

DFHBM0007A MAPNAME IS GT 7 CHARS

Severity: 8

Meaning: The map name is greater than seven characters in length.

Action: Reduce the name to seven characters or less and reassemble the map.

DFHBM0008 FIELD MACRO AFTER DFHMDI TYPE=FINAL DISCARDED

Severity: 8

Meaning: The DFHMDF macro instruction was encountered after a DFHMDI TYPE=FINAL macro instruction and before another DFHMDI TYPE=DSECT macro instruction or DFHMDI TYPE=MAP macro instruction; CICS ignores the DFHMDF macro instruction.

Action: Examine macro instructions for correct sequence and reassemble map.

DFHBM0009 NO LENGTH; MACRO DISCARDED

Severity: 8

Meaning: The DFHMDF LENGTH=number specification has been omitted. CICS ignores this field macro instruction and continues to analyze the map.

Action: Supply a valid LENGTH value (1-256) for the field and reassemble map.

DFHBM0010 NO POS; MACRO DISCARDED

Severity: 8

Meaning: The DFHMDF LENGTH=number specification has been omitted. CICS ignores this field macro instruction and continues to analyze the map.

Action: Supply a valid POS value (0-1919) for the field and reassemble map.

DFHBM0011 LENGTH OUT OF RANGE; MACRO DISCARDED

Severity: 8

Meaning: The DFHMDF LENGTH=number specification is not within the range 1-256. CICS ignores this field macro instruction and continues to analyze the map.

Action: Supply a LENGTH value within the range 1-256 and reassemble map.

DFHBM0013 POS OUT OF RANGE; MACRO DISCARDED

Severity: 8

Meaning: The DFHMDF POS=number specification is less than zero or greater than 1919. CICS ignores this field macro instruction and continues to analyze the map.

Action: Supply a valid POS value within the range 0-1919 and reassemble map.

DFHBM0014 FIELD POSITION REQIRES 3270 MODEL 2

Severity: 0

Meaning: The DFHMDF POS=number specification specifies a location that requires the 1920-character 3270 (Model 2).

Action: Ensure that this map is never used for a 3270 Model 1.

DFHBM0015 OVERLAP WITH PREVIOUS FIELD

Severity: 4

Meaning: The DFHMDF POS=number specification specifies a position that is within the scope of the preceding field definition. CICS accepts the specified value and continues to analyze the map.

Action: Ensure that the field overlap is acceptable. If not, correct by supplying a POS value that is at least one greater than the sum of the POS and LENGTH values of the previous field in the map. As an alternative, change the POS or LENGTH values of the previous field and reassemble map.

DFHBMC016 POS NOT IN ASCENDING SEQUENCE. MACRO DISCARDED.

Severity: 8

Meaning: The DFHMDF POS=number specification is not greater than the POS value of the preceding field. CICS ignores this field macro instruction and continues to analyze the map.

Action: Check the POS values for the two fields and the order of the macro instructions and reassemble map.

DFHBMC017 IRRECOVERABLE ERROR ENCOUNTERED BY DFHMDF

Severity: 16

Meaning: An irrecoverable situation was encountered by DFHMDF during map analysis. CICS abandons any further map analysis.

Action: Examine the map specification carefully for invalid parameters; see that the macro instructions are properly ordered. Correct any errors and reassemble map. If the error persists, contact your IBM representative after ensuring the availability of (1) a listing of the map analysis with the error messages, and (2) the input causing the error message to be generated.

DFHBMC018 FIELDNAME MUST BE CODED WITH GROUPNAME PRESENT

Severity: 8

Meaning: The DFHMDF macro instruction was coded with a group name but the name field was not supplied. CICS assigns a null value to the name field and continues to analyze the map.

Action: All fields within a named group require the name field to be coded. Supply a unique field name and reassemble map.

DFHBMC019 NO FIELD NAME. MACRO DISCARDED.

Severity: 4

Meaning: The DFHMDF MODE=IN specification encountered an entry with no name field entry. CICS ignores this field macro instruction and continues to analyze the map.

Action: If a symbolic storage definition entry is required for this field, supply a name in the name field and reassemble map. Rejection of a DFHMDF MODE=IN specification with an empty name field may be quite valid if the same map generation submitted for output symbolic storage definition is used to generate the symbolic storage definition for the input from that map.

DFHBM0020 DETECTABLE FIELD CANNOT BE CONTAINED UNDER A GROUP NAME

Severity: 8

Meaning: DFHMDF ATTRB=DET was specified for a field contained witin a group. CICS ignores this field macro instruction and continues to analyze the map.

Action: Check the specifications of grouped fields within the map and the ATTRB specification for this field. Reassemble map.

DFHBM0021 INVALID xxxxxxxx ATTRIBUTE SPECIFIED; IGNORED

Severity: 4

Meaning: The DFHMDF ATTRB=parameter specification is invalid. CICS ignores the invalid specification and continues to analyze the map.

Action: Check the coding of the ATTRB operand and reassemble map.

DFHBM0022 xxxxxxxx AND xxxxxxxx ARE INCOMPATIBLE; ASKIP ASSUMED

Severity: 8

Meaning: Conflicting attributes were specified for this field in the DFHMDF ATTRB=parameter macro instruction. CICS assumes ATTRB=ASKP and continues to analyze the map.

Action: Correct the conflicting specification of attributes in the ATTRB operand and reassemble map.

DFHBM0023 IC REQUESTED FOR A PROTECTED FIELD

Severity: 4

Meaning: The DFHMDF ATTRB=parameter macro instruction requested insertion of the cursor within a protected field. CICS accepts the request and continues to analyze the map.

Action: Ensure the validity of the request for this field. If invalid, correct and reassemble map.

DFHBM0024 ASKIP IMPLIES xxxxxxxx

Severity: 4

Meaning: The DFHMDF ATTRB=parameter macro instruction specified two attributes, one of which implied the other; for example, ATTRB=(ASKIP,PROT) where ASKIP includes PROT. CICS uses the more emcompassing attribute and continues to analyze the map.

Action: If the more encompassing attribute is acceptable, no action is necessary. Otherwise, correct the ATTRB specification and reassemble map.

DFHBM0025 BRT IMPLIES DET

Severity: 4

Meaning: The DFHMDF ATTRB=parameter macro instruction specified the BRT attribute which also implies the DET attribute. CICS uses the BRT attribute and continues to analyze the map.

Action: If the BRT attribute is not required for this field, change the ATTRB specification and reassemble map.

DFHBM0026 PROT AND NUM IMPLY ASKIP

Severity: 4

Meaning: The DFHMDF ATTRB=parameter macro instruction specified PROT and NUM. The combination of these two parameters creates a field that also has the ASKIP attribute.

Action: No action is necessary; this message is informative only.

DFHBM0027 ATTRIBUTE xxxxxxxx REPEATED. IGNORED.

Severity: 4

Meaning: The DFHMDF ATTRB=parameter specification contains the repetition of an attribute. CICS accepts the repetition without action and continues to analyze the map.

Action: Eliminate the repetition to remove the error message (if required).

DFHBM0028 DUPLICATE TYPE OPTION IGNORED

Severity: 0

Meaning: A duplicated map TYPE specification was encountered and ignored.

Action: No action message is necessary; this message is informative only. If desired, remove the duplicate specification before reassembling the map.

DFHBM0029 INVALID TYPE SPECIFIED; OUT ASSUMED BY DEFAULT

Severity: 8

Meaning: A type specification was found which was not IN, OUT, ERASE, WAIT, MAP, or SAVE. OUT is assumed by default.

Action: If OUT is not an acceptable default, correct the error and reassemble the map.

DFHBM0030 MAPNAME IS GT 7 CHARS; TRUNCATED

Severity: 12

Meaning: A map name greater than seven characters was encountered and truncated to seven characters.

Action: Correct the map name and reassemble the map.

DFHBM0031 DATA = SPECIFIED INCORRECTLY; NO IS ASSUMED AS DEFAULT.

Severity: 8

Meaning: A data specification was encountered which was not YES, NO, or ONLY. DATA=NO is assumed.

Action: If NO is not an acceptable default, correct the DATA specification and reassemble the map.

DFHBM0032 DATA SPEC NOT REQUIRED WITH THIS TYPE; IGNORED.

Severity: 4

Meaning: Initial DATA was specified for a map which is not specified as an output map. The specification is ignored.

Action: If it is desired that an output map be generated, change the TYPE specification to OUT and reassemble the map.

DFHBM0033 CURSOR POSITION REQUIRES TYPE=OUT; THIS REQUEST IGNORED.

Severity: 4

Meaning: A cursor specification was provided for a map which was not an output map. The specification is ignored.

Action: If the map TYPE was specified incorrectly, change the specification to OUT and reassemble the map.

DFHBM0034 MAPADR SYMBOL GT 8 CHARS.

Severity: 4

Meaning: The MAPADR operand specified a name greater than eight characters. Only the first eight will be used to address the map.

Action: Correct the MAPADR specification and reassemble the map.

DFHBM0035 INVALID LANGUAGE ASSEMBLER ASSUMED.

Severity: 4

Meaning: The LANG operand was not ASM, COBOL, or PL1. BMS assumes LANG=ASM.

Action: If the language desired is not Assembler, correct the LANG specification and reassemble the map.

DFHBM0036 INPUT SPEC WITH INCONSISTENT OPERANDS; INPUT, WAIT ASSUMED.

Severity: 4

Meaning: TYPE=INPUT was specified along with OUT, ERASE, or MAP. These combinations are inconsistent and only INPUT is processed.

Action: If some other specification is desired, correct the TYPE specification and reassemble the map.

DFHBM0037 OUTPUT SPEC WITH INCONSISTENT OPERANDS; OUTPUT, WAIT ASSUMED.

Severity: 4

Meaning: TYPE=(OUT,MAP) was specified which is inconsistent.

Action: Correct the TYPE specification and reassemble the map.

DFHBM0038 ERASE SPEC WITH INCONSISTENT OPERANDS; OUTPUT, ERASE, WAIT ASSUMED.

Severity: 4

Meaning: A) Either TYPE=(ERASE,MAP) was specified or

B) TYPE=(ERASE) was specified without OUT.

Action: Correct the specification and reassemble the map.

DFHBM0039 SAVE REQUIRES OUT; SAVE IGNORED

Severity: 4

Meaning: The TYPE operand specified SAVE but not OUT.

Action: Correct the specification and reassemble the map.

DFHBM0040 INVALID CURSOR POSITION DEFAULTS TO ZERO

Severity: 4

Meaning: The cursor keyword specified a value less than 0 or greater than 1919 and therefore invalid for the 3270.

Action: Correct the specification and reassemble the map.

DFHBM0041 CURSOR POSITION REQUIRES 3270 Model 2

Severity: 0

Meaning: The cursor specification is between 480 and 1919 and therefore only valid for a 3270 Model 2.

Action: Do not try to use this map on a 3270 Model 1 or unpredictable results will occur.

DFHBM0042 DATA = NOT SPECIFIED; NO IS ASSUMED AS A DEFAULT.

Severity: 4

Meaning: DATA= was not specified for a TYPE=OUT specification. DATA=NO is assumed.

Action: If NO is not an acceptable default, correct the DATA specification and reassemble the map.

DFHBM9999 IRRECOVERABLE ERROR ENCOUNTERED BY DFHMDI

Severity: 16

Meaning: The DFHMDI macro instruction encountered an irrecoverable situation during map analysis. CICS abandons any further map analysis.

Action: Examine the map specification carefully for invalid parameters and see that the macro instructions are properly ordered. Correct any errors and reassemble map. If the error persists, contact your IBM Representative after ensuring the availability of (1) a listing of the map analysis with the error messages and (2) the input causing the error message to be generated.

APPENDIX E: TRANSLATE TABLES FOR THE 2980

This section contains translate tables for the following components of the 2980 General Banking Terminal System:

1. 2980 Teller Station Model 1
2. 2980 Administrative Station Model 2
3. 2980 Teller Station Model 4

The line codes and CPU codes listed in these tables are unique to CICS and are represented as standard EBCDIC characters.

2980-1 CHARACTER SET/TRANSLATE TABLE

KEY No.	ENGRAVING Top(LC)	ENGRAVING Front(UC)	PRINTING Numeric(LC)	PRINTING Alpha(UC)	LINE Code	CPU CODE Numeric(LC)	CPU CODE Alpha(UC)	High Level Lang. ID
0	MSG ACK	1	↕	1	F1	AA	F1	1
1	SEND AGAIN	Q	R	Q	D8	D9	D8	
2	CORR	A	C	A	C1	C3	C1	
3	HOLD OVERRIDE	2	H	2	F2	C8	F2	
4	VOID	Z	V	Z	E9	E5	E9	
5	ACCT INQ	W	Q	W	E6	D8	E6	
6	ACCT TFR	S	↕	S	E2	AB	E2	2
7	CIF	3	↕	3	F3	AC	F3	3
8	MISC	X	↕	X	E7	AD	E7	4
9	CLSD ACCT	E	X	E	C5	E7	C5	
10	NO BOOK	D	↕	D	C4	AE	C4	5
11	MORT LOAN	4	$\overline{M}$	4	F4	AF	F4	6
12		C	‡	C	C3	B0	C3	7
13	NEW ACCT	R	$\overline{A}$	R	D9	B1	D9	8
14	BOOK BAL	F	↕	F	C6	B2	C6	9
15	INST LOAN	5	$\overline{L}$	5	F5	B3	F5	10
16	SPEC TRAN	V	↕	V	E5	B4	E5	11
17	SAV BOND	T	$\overline{B}$	T	E3	B5	E3	12

2980-1 CHARACTER SET/TRANSLATE TABLE

KEY No.	ENGRAVING Top(LC)	ENGRAVING Front(UC)	PRINTING Numeric(LC)	PRINTING Alpha(UC)	LINE Code	CPU CODE Numeric(LC)	CPU CODE Alpha(UC)	High Level Lang. ID
18	SAV	G	$\overline{S}$	G	C7	B6	C7	13
19	XMAS CLUB	6	$\overline{C}$	6	F6	B7	F6	14
20		B	.	B	C2	4B	C2	
21	DDA	Y	$\overline{D}$	Y	E8	B8	E8	15
22	@	H	@	H	C8	B9	C8	16
23	MON ORD	7	$\overline{M}$	7	F7	BA	F7	17
24	0	N	0	N	D5	F0	D5	
25	7	U	7	U	E4	F7	E4	
26	4	J	4	J	D1	F4	D1	
27	CSHR CHK	8	$\overline{I}$	8	F8	BB	F8	18
28	1	M	1	M	D4	F1	D4	
29	8	I	8	I	C9	F8	C9	
30	5	K	5	K	D2	F5	D2	
31	CASH RECD	9	$\overline{9}$	9	F9	BC	F9	19
32	2	,	2	,	6B	F2	6B	
33	9	O	9	O	D6	F9	D6	
34	6	L	6	L	D3	F6	D3	

2980-1 CHARACTER SET/TRANSLATE TABLE

KEY No.	ENGRAVING Top(LC)	ENGRAVING Front(UC)	PRINTING Numeric(LC)	PRINTING Alpha(UC)	LINE Code	CPU CODE Numeric(LC)	CPU CODE Alpha(UC)	High Level Lang. ID
35	UTIL BILL	0	U	0	F0	E4	F0	
36	3	.	3	.	4B	F3	4B	
37	DEP +	P	+	P	D7	4E	D7	
38	WITH -	$	-	$	5B	60	5B	
39	FEES	-	F	-	60	C6	60	
40	TOTL	/	T	/	61	E3	61	
41	CASH IN	*	$	*	5C	BD	5C	20
42	CASH CHK	#	$	#	7B	BE	7B	21
43	VAL	&	A-K	&	50	STATION ID	50	
44	TAB				05	05	05	TABCHAR
45	ALPHA ENTRY				36			
46	NUMERIC ENTRY				06			
47	SEND				26-ETB 03-ETX			
48	RETURN				15	15	15	JRNLCR
49	NUMERIC ENTRY				06			
50	SPACE				40	40	40	
58	MSGLIGHT				17	17	17	MSGLITE

2980-2 CHARACTER SET/TRANSLATE TABLE

KEY No.	ENGRAVING Top(LC)	ENGRAVING Front(UC)	PRINTING Numeric(LC)	PRINTING Alpha(UC)	LINE Code	CPU CODE Numeric(LC)	CPU CODE Alpha(UC)	High Level Lang. ID
0	= 1		1	=	F1	F1 (1)	7E (=)	
1	Q		q	Q	D8	98 (q)	D8 (Q)	
2	A		a	A	C1	81 (a)	C1 (A)	
3	2		2	<	F2	F2 (2)	4C (<)	
4	Z		z	Z	E9	A9 (z)	E9 (Z)	
5	W	NO KEYFRONT ENGRAVING	w	W	E6	A6 (w)	E6 (W)	
6	S ;		s	S	E2	A2 (s)	E2 (S)	
7	3		3	;	F3	F3 (3)	5E (;)	
8	X		x	X	E7	A7 (x)	E7 (X)	
9	E		e	E	C5	85 (e)	C5 (E)	
10	D :		d	D	C4	84 (d)	C4 (D)	
11	4		4	:	F4	F4 (4)	7A (:)	
12	C		c	C	C3	83 (c)	C3 (C)	
13	R		r	R	D9	99 (r)	D9 (R)	
14	F %		f	F	C6	86 (f)	C6 (F)	
15	5	NO KEYFRONT ENGRAVING	5	%	F5	F5 (5)	6C (%)	
16	V		v	V	E5	A5 (v)	E5 (V)	
17	T		t	T	E3	A3 (t)	E3 (T)	
18	G '		g	G	C7	87 (g)	C7 (G)	
19	6		6	'	F6	F6 (6)	7D (')	
20	B		b	B	C2	82 (b)	C2 (B)	
21	Y		y	Y	E8	A8 (y)	E8 (Y)	
22	H >		h	H	C8	88 (h)	C8 (H)	
23	7		7	>	F7	F7 (7)	6E (>)	
24	N		n	N	D5	95 (n)	D5 (N)	
25	U		u	U	E4	A4 (u)	E4 (U)	

2980-2 CHARACTER SET/TRANSLATE TABLE

KEY No.	ENGRAVING Top(LC)	ENGRAVING Front(UC)	PRINTING Numeric(LC)	PRINTING Alpha(UC)	LINE Code	CPU CODE Numeric(LC)	CPU CODE Alpha(UC)	High Level Lang. ID
26	J		j	J	D1	91 (j)	D1 (J)	
27	* 8		8	8	F8	F8 (8)	5C (*)	
28	M		m	M	D4	94 (m)	D4 (M)	
29	I		i	I	C9	89 (i)	C9 (I)	
30	K		k	K	D2	92 (k)	D2 (K)	
31	(9		9	(	F9	F9 (9)	4D (()	
32	| ,		,	|	6B	6B (,)	4F (|)	
33	O		o	O	D6	96 (o)	D6 (O)	
34	L		l	L	D3	93 (l)	D3 (L)	
35	) 0		0	)	F0	F0 (0)	5D ())	
36	¬ .	NO KEYFRONT ENGRAVING	.	¬	4B	4B (.)	5F (¬)	
37	P		p	P	D7	97 (p)	D7 (P)	
38	! $		$	!	5B	5B ($)	5A (!)	
39	¯ -		-	_	60	60 (-)	6D (_)	
40	? /		/	?	61	61 (/)	6F (?)	
41	¢ @		@	¢	5C	7C (@)	4A (¢)	
42	= #	NO KEYFRONT ENGRAVING	#	"	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	16	16	BCKSPACE
48	RETURN				15	15	15	
49	SHIFT				06	06	06	
50	(SPACE)				40	40	40	
53	SEND				26-ETB 03-ETX			

2980-4 CHARACTER SET/TRANSLATE TABLE

KEY No.	ENGRAVING Top(LC)	ENGRAVING Front(UC)	PRINTING Numeric(LC)	PRINTING Alpha(UC)	LINE Code	CPU CODE Numeric(LC)	CPU CODE Alpha(UC)	High Level Lang. ID
0	CK $		¢		D9	BC	60	19
1		Q	L	Q	D3	D3	D8	
2		A	A	A	C1	C1	C1	
3	CK #	0	$\bar{C}$	0	C9	B7	C9	14
4		Z	.	Z	E9	4B	E9	
5		W	*	W	E6	5C	E6	
6		S	$	S	E2	5B	E2	
7	IMD 2	1	I	1	5B	4F	F1	
8		X	¥	X	E7	AE	E7	5
9		E	E	E	C5	C5	C5	
10		D	?	D	C4	6F	C4	
11	IMD 1	2	M	2	4B	D4	F2	
12		C	C	C	C3	C3	C3	
13		R		R	60	60	D9	
14		F	F	F	C6	C6	C6	
15	CODE	3	I	3	E8	BB	F3	18
16		V	▽	V	E5	A0	E5	22
17		T	△	T	E3	A1	E3	23

2980-4 CHARACTER SET/TRANSLATE TABLE

KEY No.	ENGRAVING Top(LC)	ENGRAVING Front(UC)	PRINTING Numeric(LC)	PRINTING Alpha(UC)	LINE Code	CPU CODE Numeric(LC)	CPU CODE Alpha(UC)	High Level Lang. ID
18		G	G	G	C7	C7	C7	
19	AMT	4	$	4	5C	BE	F4	21
20		B	B	B	C2	C2	C2	
21		Y	/	Y	61	61	E8	
22		H	P	H	D7	D7	C8	
23	OB	5	%	5	D8	B2	F5	9
24		N	N	N	D5	D5	D5	
25		U	¯	U	E4	AF	E4	6
26		J	J	J	D1	D1	D1	
27	ACCT #	6	#	6	C8	7B	F6	
28		N	X	M	D4	E7	D4	
29		I	0	I	D6	D6	C9	
30		K	K	K	D2	D2	D2	
31	7	7	7	7	F7	F7	F7	
32	---	---	'	'	6B	BLANK	6B	
33	4	0	4	0	F4	F4	D6	
34	1	L	1	L	F1	F1	D3	

2980-4 CHARACTER SET/TRANSLATE TABLE

KEY No.	ENGRAVING Top(LC)	ENGRAVING Front(UC)	PRINTING Numeric(LC)	PRINTING Alpha(UC)	LINE Code	CPU CODE Numeric(LC)	CPU CODE Alpha(UC)	High Level Lang. ID
35	8	8	8	8	F8	F8	F8	
36	0	.	0	.	F0	F0	4B	
37	5	P	5	P	F5	F5	D7	
38	2	$	2	$	F2	F2	5B	
39	9	9	9	9	F9	F9	F9	
40	---	---	‡	*	7B	B0	7B	7
41	6	*	6	*	F6	F6	5C	
42	3	#	3	#	F3	F3	7B	
43	VAL	&	A-K	&	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

1 PAGE 0035

SH20-1047-4

Customer Information Control System (CICS) APRM Printed in U.S.A. SH20-1047-4

International Business Machines Corporation
Data Processing Division
1133 Westchester Avenue, White Plains, New York 10604
(U.S.A. only)

IBM World Trade Corporation
821 United Nations Plaza, New York, New York 10017
(International)

IBM Technical Newsletter

This Newsletter No.	SN20-2983
Date	January 5, 1973
Base Publication No.	SH20-1047-4
File No.	
Previous Newsletters	None

Customer Information Control System (CICS)
Application Programmer's Reference Manual

This Technical Newsletter provides an index (pages 273-279) to the subject manual.

Please file this cover letter at the back of the manual.

IBM Corporation, Technical Publications Dept., 1133 Westchester Avenue, White Plains, N.Y. 10604

Printed in U.S.A.

IBM Technical Newsletter

This Newsletter No.	SN20-9012
Date	April 11, 1973
Base Publication No.	SH20-1047-4
Previous Newsletters	SN20-2983

Customer Information Control System (CICS)
Application Programmer's Reference Manual

This Technical Newsletter provides replacement pages for the subject manual. These replacement pages remain in effect for subsequent versions and modifications unless specifically altered. Pages to be inserted and/or removed are listed below.

Pages

Contents	48	198.1 (add)
1	61, 62	199, 200
1.1 (add)	83, 84	201, 202
2	85, 86	203
11, 12	89, 90	203.1 (add)
31	97, 98	204, 205
31.1 (add)	98.1 (add)	205.1 (add)
32	111, 112	206, 207
39	119, 120	208, 209
39.1 (add)	120.1 (add)	210
40	155	227, 228
47	155.1 (add)	New Reader's Comment Form
47.1 (add)	156	
47.2 (add)	197, 198	

Vertical rules in the left margin indicate changes.

Please file this cover at the back of the manual to provide a record of changes.

IBM Corporation, Department J04, 1501 California Avenue, Palo Alto, California 94304

Printed in U.S.A.

SH20-1047-4

(CICS) — Application Prog. Reference Manual Printed in U.S.A. SH20-1047-4

International Business Machines Corporation
Data Processing Division
1133 Westchester Avenue, White Plains, New York 10604
(U.S.A. only)

IBM World Trade Corporation
821 United Nations Plaza, New York, New York 10017
(International)

READER'S COMMENT FORM

Customer Information Control System (CICS)
Application Programmer's Reference Manual

SH20-1047-4

Please comment on the usefulness and readability of this publication, suggest additions and deletions, and list specific errors and omissions (give page numbers). All comments and suggestions become the property of IBM. If you wish a reply, be sure to include your name and address.

COMMENTS

fold fold

fold fold

- Thank you for your cooperation. No postage necessary if mailed in the U.S.A.
FOLD ON TWO LINES, STAPLE AND MAIL.

YOUR COMMENTS PLEASE...

Your comments on the other side of this form will help us improve future editions of this publication. Each reply will be carefully reviewed by the persons responsible for writing and publishing this material.

Please note that requests for copies of publications and for assistance in utilizing your IBM system should be directed to your IBM representative or the IBM branch office serving your locality.

fold fold

FIRST CLASS
PERMIT NO. 1359
WHITE PLAINS, N. Y.

BUSINESS REPLY MAIL
NO POSTAGE NECESSARY IF MAILED IN THE UNITED STATES

POSTAGE WILL BE PAID BY . . .

IBM Corporation
1133 Westchester Avenue
White Plains, N.Y. 10604

Attention: Technical Publications

fold fold

(CICS) — Application Prog. Reference Manual Printed in U.S.A. SH20-1047-4

IBM

International Business Machines Corporation
Data Processing Division
1133 Westchester Avenue, White Plains, New York 10604
[U.S.A. only]

IBM World Trade Corporation
821 United Nations Plaza, New York, New York 10017
[International]