IBM

Series/1

SL23-0090-0
File No. S1-14

LICENSED
PROGRAM

IBM Series/1 EDX Communications Facility
Work Session Controller High-Level Language Subroutines
Programmer's Guide

IBM

Series/1

SL23-0090-0
File No. S1-14

LICENSED
PROGRAM

IBM Series/1 EDX Communications Facility
Work Session Controller High-Level Language Subroutines
Programmer's Guide

First Edition (February 1982)

This edition applies to Version 1.0 of the Licensed Program IBM Series/1 Event Driven Executive Communications Facility, Program Number 5719-CF1, and to all subsequent releases of this program unless otherwise indicated in new editions or technical newsletters.

Information in this publication is subject to significant change. Any such changes will be reported in new editions or technical newsletters. Before using this publication, consult the latest *IBM Series/1 Publications Directory*, GC34-0361, and the technical newsletters that amend the directory, to be sure the edition you have is applicable and current. Use this publication only for the purposes mentioned under "About This Book."

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

Direct all requests for copies of IBM publications to your IBM representative or to the IBM branch office that serves your locality.

Any names of individuals, companies, brands, and products in the examples in this book are fictitious and any similarity to the name of an actual business enterprise is entirely coincidental.

This publication may contain technical or typographical errors. Forms for readers' comments are provided at the back of the book. If the forms have been removed, address your comments to IBM Corporation, Series/1 Program Product Development, PAS5, 1501 California Avenue, PO Box 10500, Palo Alto, CA 94304. IBM may use and distribute any of the information you supply in any way it believes appropriate without incurring any obligation whatever. You may, of course, continue to use the information you supply.

ABOUT THIS BOOK

This book is intended for programmers who are going to code programs to communicate with Event Driven Executive (EDX) terminals through calls to subroutines that are supplied with the IBM Series/1 Event Driven Executive Communications Facility.

The subroutines provide an easy-to-use, high-level interface to the functions of the Communications Facility's work session controller. Calls to the subroutines can come from programs written in COBOL and in the Event Driven Executive Language (EDL). The subroutine calls are compatible with corresponding Multiple Terminal Manager subroutine calls; Multiple Terminal Manager applications can be readily converted to run under the Communications Facility.

This book assumes that you already understand the work session controller's functions, know how a Communications Facility system is organized, and know the purpose of the program you plan to code. If you need introductory information about the Communications Facility, refer to *IBM Series/1 Event Driven Executive Communications Facility Introduction,* GL23-0071. For information about the work session controller, see *IBM Series/1 Event Driven Executive Communications Facility Programmer's Guide,* SL23-0074 (referred to in this manual simply as the *Programmer's Guide.* For detailed information about the structure of a Communications Facility system, see *IBM Series/1 Event Driven Executive Communications Facility Design and Installation Guide,* SL23-0073.

Other Communications Facility manuals you may want to refer to are *IBM Series/1 Event Driven Executive Communications Facility: Operator's Guide,* SL23-0075, which explains how to use the operator commands and utilities, and *IBM Series/1 Event Driven Executive Communications Facility: Debugging Guide,* LL23-0076, which gives the formats of control blocks and describes the EDL language extensions that are intended for internal system use.

This book assumes that you know either the Event Driven Executive language (EDL) which is presented in *IBM Series/1 Event Driven Executive Language Reference,* SC34-1706, or EDX COBOL, which is presented in *IBM Series/1 Event Driven Executive COBOL Language Reference,* GC34-0392.

This book also assumes you know how to use the facilities of the EDX system, Version 3, as explained in:

IBM Series/1 Event Driven Executive System Guide, SC34-1702

IBM Series/1 Event Driven Executive Messages and Codes, SC34-0403

IBM Event Driven Executive Program Preparation Guide, SC34-1704

IBM Event Driven Executive Communications and Terminal Applications Guide, SC34-1705.

This book will give you the information you need to code programs that use the work session controller high-level language subroutines.

To that end, it has these chapters:

- "Using the High-Level Language Subroutines" explains the functions you can perform, in a user program, through the high-level language subroutines. It explains what your program gets as input; what the various subroutines do; how to get your program installed and running; and what design considerations apply to a program that uses the subroutines.
- "Using the Subroutines in an EDL Program" explains what work session controller functions you can use from an EDL program, how to call each function, what parameters you get on entry, and what you get as output. It explains how to link-edit the required programs. It presents sample programs that make use of various subroutines.
- "Using the Subroutines in a COBOL Program" explains what work session controller functions you can use from a COBOL program, how to call each function, what parameters you get on entry, and what you get as output. It explains how to link-edit the required programs. It presents sample programs that make use of various subroutines.
- "Debugging Your Program" explains how to use the $DEBUG utility to debug an application program that uses the work session controller high-level language subroutines.
- "Glossary-Index" combines a glossary of technical Communications Facility terms with a conventional index to the publication.

CONTENTS

USING THE HIGH-LEVEL LANGUAGE SUBROUTINES

The Communications Facility work session controller high-level language subroutines allow you to write interactive application programs to communicate with EDX terminals anywhere in the network. Such programs can communicate with 4978, 4979, and 3101 terminals, and 4973, 4974, and 4975 printers. The interactive application program can communicate with multiple terminals, which can be attached to any Series/1 in the network. The terminals need not be defined to the Communications Facility, but they must be defined to EDX.

Introducing the Subroutines

This section presents an overview of the work session controller high-level language subroutines. It explains how the subroutines are related to the work session controller, other parts of the Communications Facility, and the multiple terminal manager.

Work Session Controller Functions

The Communications Facility program that allows you to communicate with EDX terminals is called the *work session controller.* You can communicate with the work session controller through calls to a set of subroutines from COBOL or EDX programs. You issue a call that indicates what you want done at the terminal—for example, reading data, writing data, or sounding a tone. The work session controller, running in the Series/1 to which the terminal is physically attached, uses EDX I/O instructions to perform the function you requested.

Alternatively, you can communicate with the work session controller by means of transactions. You send the work session controller a transaction specifying what you want done at the terminal. It sends another transaction as an acknowledgment. If you want to communicate with the work session controller through transactions, see the *Communications Facility Programmer's Guide.*

Multiple Terminal Manager Call Formats

The work session controller high-level language subroutines are named, and the subroutine calls are designed, to be compatible with the corresponding multiple terminal manager subroutine calls. The menu from which the terminal user selects an application is identical to the corresponding multiple terminal manager menu. Thus, if you have multiple terminal manager applications, you can readily adapt them to run under the Communications Facility.

Images in $.WSCIMG

Images displayed through the work session controller are built through $IMAGE and stored in the partitioned data set $.WSCIMG. "Storing Images in the Image Library" in the *Programmer's Guide* explains how to convert $IMAGE images to work session controller images.

Access through $.WSMENU

The Communications Facility includes a program, $.WSMENU, that starts communication between your application and its users. "Using $.WSMENU to Start Using the Work Session Controller" in the *Programmer's Guide* describes $.WSMENU.

Support Programs

Besides the subroutines themselves, the call interface to the work session controller includes:

- A *primary application load* program. This program presents terminal users with a *primary menu screen,* from which they select the application they want to run.
- Four *interface programs* (two each for COBOL and EDL: one for systems that use the Indexed Access Method and one for systems that don't), which process the various subroutine calls.

Calling the Subroutines

To make an I/O request, you call one of the work session controller high-level language subroutines. There are 11 different calls; you use them to specify what you want done. The chapters "Using the Subroutines in a COBOL Program" and "Using the Subroutines in an EDL Program" give the formats and parameters of the subroutine calls.

Terminal Management

Seven of the subroutine calls specify what is to be done at the terminal:

ACTION
Enables the application program to display a screen on the terminal and then obtain the operator's response to that display.

BEEP
Enables the application program to sound the tone, if the terminal has this feature, on the next output as a signal to the terminal operator.

CHGPAN
Enables the application program to modify the terminal screen image dynamically. It writes protected information.

SETPAN
Enables the application program to retrieve a specified screen from the $.WSCIMG data set and display it on the terminal.

FTAB
Sets up a table that describes the unprotected input fields placed in the input buffer after a SETPAN is issued. This function is useful in cursor positioning.

GETCUR
Enables the application program to obtain the current position of the cursor on 4978, 4979, and 3101 displays.

SETCUR
Enables the application program to set the position of the cursor on 4978, 4979, and 3101 displays.

Program Management

Five of the subroutine calls enable you to manage your program and allow it to run in synchronization with other interactive applications:

CYCLE
Enables an application program to suspend its execution to allow other applications to become active.

FAN
This must be coded as the first instruction in a COBOL application program to allow for COBOL environment initiatization.

LINK
Enables an application program to complete its own execution by loading and executing some other application program.

LINKON
Enables an application to request an operator action and, when this action is complete, load and execute some other application program.

MENU
Enables the application program to end its own operation and return control to the primary application load program. The program selection menu is then displayed on the terminal.

File Management

One subroutine call provides common support for all disk data transfer operations needed by the application programs. It supports both indexed and direct files under the control of a single callable function:

FILEIO
Enables the application program to perform read and write operations to disk or diskette using either indexed or direct accessing.

All requests for disk and diskette I/O are by means of a call to the FILEIO routine. FILEIO provides the following functions:

- Automatic open and close of the requested data set
- Support for direct-access files, where records are accessed by a relative record number (RRN)
- Support for Indexed Access Method files, providing a high-level language interface to most Indexed Access Method services. (If Indexed Access Method files are used, the Event Driven Executive Indexed Access Method (5719-AM3 or 5719-AM4) is required.)
- Data integrity, through automatic close and automatic writeback of data buffers.

Open/Close

Data sets are located and pre-bound from the PROGRAM statement of the EDL stub program when $.PD (the Communications Facility program dispatcher) is loaded. This process limits application programs to 9 data sets for EDL and 8 data sets for COBOL.

When you're using IAM data sets, the interface program automatically opens files requested by the first FILEIO operation performed in your application programs. It releases record locks across a CALL ACTION to prevent deadlocks from occurring.

Indexed File Support

Application programs can access indexed files by calling the FILEIO routine. The functions supported are listed in the description of the FILEIO format. You must create an Indexed Access Method file.

Some features of the indexed file support are:

- Records can be retrieved sequentially or by key.
- The key can be a generic key (the first *n* bytes of the actual key).
- Records can be added or deleted by key.
- It takes the same time to retrieve added records as original records.

If an application requires access to a file sequentially, and also directly by alphameric keys, indexed files are required.

Passing Parameters

You pass parameters to these subroutines just as you would to any other subroutine.

For example:

```
       CALL   SETPAN,(SCRNX),(RC)
        •
        •
        •
SCRNX  DC     CL8'SCRN10' SCREEN NAME
RC     DC     F'0'        RETURN CODE FIELD
        •
        •
        •
```

This example passes the addresses of the screen name and return code field to the SETPAN routine.

Note that you can't use the software registers (#1 and #2) in calls to these subroutines. However, the contents of #1 and #2 are maintained across calls in your program.

Structuring Your Application Program

You must code your application in two parts: an EDL stub program, and your application itself, which can be written in either COBOL or EDL.

The EDL stub program defines data sets and defines the sizes of input/output buffers. It sets up a list of five (for COBOL) or four (for EDL) parameters, which are passed to the application itself upon initiation:

- Input buffer address
- Output buffer address
- Terminal environment block address
- Interrupt information byte address
- Data save area address (COBOL only)

Input Buffer Address

The input buffer address is the address of a buffer used to contain the data input from the terminal after an ACTION. After the operator presses ENTER or a PF key, ACTION reads the data in the unprotected fields of the terminal into the input buffer. The input data fields are contiguous and start at the beginning of the buffer.

Output Buffer Address

The output buffer address is the address of a buffer that is used for two purposes.

First, your application program can place data in the output buffer. A subsequent call to ACTION writes the data from the buffer to the unprotected fields of the screen. If there are more characters are in the output buffer than unprotected positions on the screen, the excess characters are lost. The output buffer is set to blanks after a return from CALL ACTION.

Second, the output buffer is used for passing data between programs, when one links to another. Before a LINK to another program, your program may store data in the output buffer; The second program will find that data in its output buffer.

Terminal Environment Block (TEB) Address

This is the address of a word that will be 0 if the terminal is a 4978 or 4979, and 1 if it is a 3101. It is useful for systems with both type of terminals to be able to determine if FTAB (field table definitions) are needed or not.

Interrupt Information Byte (IIB) Address

For a 4978, 4979, or 3101, this is the address of a word containing a numeric value that represents the interrupting key which the operator pressed.

Data Save Area Address

This is the address, passed to a COBOL application program, of a save area defined in the EDL stub program for saving modified data between calls to the subroutines. "Creating Your Save Area" gives details about use of the save area.

EDL Stub Program Coding

The EDL stub program for a COBOL application is shown in Figure 1; the EDL stub program for an EDL application is shown in Figure 2.

The INPUT and OUTPUT statements define the input and output buffers for the application program. They need be only as large as the total of unprotected data bytes defined on the largest screen the application program uses.

There are two copy code modules on ASMLIB, one of which must be included in the EDL stub program for either COBOL or EDL applications:

HLSCCC—Copy code for a COBOL application
HLSCCE—Copy code for an EDL application

The copy code modules contain ENTRY and EXTRN statements as well as code necessary for application initialization.

```
CFMTM      PROGRAM START,DS=((COLTRDS,EDX002),(FILENAME,EDX003))
           COPY    HLSCCC
INPUT      DEFINE  BUFFER,SIZE=500
OUTPUT     DEFINE  BUFFER,SIZE=500
TEB        DATA    F'0'
IIB        DATA    F'0'
SAVEAREA   DATA    A(ENDSAVE-SAVEAREA)
           DATA    100F'0'
ENDSAVE    EQU     *
           ENDPROG
           END
```

Note that the data set "COLTRDS,EDX002" must be specified for COBOL stub programs.

Figure 1. EDL Stub Program for a COBOL Application

```
CFMTM  PROGRAM START,DS=((CUSTFILE,EDX003),(PAYFILE,EDX003))
       COPY    HLSCCE
INPUT  DEFINE  BUFFER,SIZE=500
OUTPUT DEFINE  BUFFER,SIZE=500
TEB    DATA    F'0'
IIB    DATA    F'0'
       ENDPROG
       END
```

Note that the compiler output module name of the EDL stub program must match the EDL stub name in the associated link control data set.

Figure 2. EDL Stub Program for an EDL Application

Your application program is compiled as a COBOL or EDL subroutine program and will be linked with the EDL stub program and other software modules for execution.

Using the Input and Output Buffers

The subroutines use the input and output buffers as a staging area to build work session controller transactions.

When defining the input and output buffer sizes in the EDL stub program, note these considerations:

- The input and output buffers must both be as large as the total number of protected characters defined on the largest screen image used by the application.
- The subroutines use the work session controller save data and restore data functions between calls (for example, calls to ACTION and CYCLE) to save the modified data in the program's save area. The total input/output buffer size must be as large as the save area, rounded up to a multiple of 256 bytes, plus 40 bytes.

  For example, if the save area is 400 bytes, the total buffer size must be 512 (the next higher multiple of 256) plus 40, or 552 bytes.

- If CALL CHGPAN commands are used, the total input/output buffer size must be as large as the length parameter on the call statement plus 40 bytes.

  For example, if your program includes this code:

```
CALL CHGPAN,(ROW),(COL),(LEN),DATA
   •
   •
   •
LEN  DATA F'500'
```

  then the total of the input and output buffers must be 540 bytes (500 + 40).

Figure 3 shows a programmer's view of the contents of the input and output buffers on entry to the application program.

Figure 4 shows the action the subroutines take on the buffer contents.

Entering Your Program into $.SYSPD

Your application program must be defined to the Communications Facility as a transaction-processing program. You must make an entry for it in the $.SYSPD data set.

Use an EDX editor to add the following TID (transaction identifier) statement to the data set:

TID *tranid pgm,vol,part* 32,P

tranid
is the 1- to 4-character transaction identifier.

pgm
is the 1- to 8-character name of the program that is to process the transaction.

Buffer Contents Upon Entry to Application Program	Input Buffer	Output Buffer
From CALL ACTION	Unprotected data read from screen	Blanks (X'40')
From CALL CYCLE	Blanks (X'40')	Unchanged
From CALL LINK	Blanks (X'40')	Unchanged from calling program
From CALL LINKON	Unprotected data read from screen	Blanks (X'40')

Figure 3. Buffer Contents on Entry to Application Program

Action Taken Upon Buffer Contents	Input Buffer	Output Buffer
By CALL ACTION	In all cases, written protected if CALL SETPAN has been issued; no action if CALL SETPAN has not been issued.	Written into unprotected fields on screen.
By CALL CYCLE		Saved.
By CALL LINK		Saved.
By CALL LINKON		Written into unprotected fields on screen.

Figure 4. Buffer Contents During Terminal I/O Operations

vol
is the 1- to 6-character volume name of the program. The default is the volume where the message dispatcher resides.

part
is the number of the partition where the program is to be loaded. You can specify any of the following:

1 to 8
means that the specified partition is used.

0
means that any available partition is used.

-1 to -8
means that any available partition is used except the one specified.

CF
means that the $.CF partition is used.

NCF
means that any partition is used except the $.CF partition.

32
is the transaction type, discussed under "Determining the Transaction's Type" in the *Programmer's Guide*.

P
indicates that the program may be stopped to make its storage available to other transaction-processing programs.

This is an example of a $.SYSPD to which three transaction identifiers (MTM2, MTM3, and MTM4) have been added:

```
TID HMU $.HMU 22
TID WSC $.WSC 42
TID USR4 $.WSCHLS 32,P
TID MTM2 CFMTM2,EDX003,4 32,P
TID MTM3 CFMTM3,,4 32,P
TID MTM4 CFMTM4 32,P
```

Note that the "TID USR4 $.WSCHLS" transaction identifier is required to allow users to gain access to your program from the $.WSMENU program.

How Users Access Your Program

A user can select your application programs by selecting a transaction identifier from the primary menu or through a program call to LINK or LINKON. The primary menu is used only for program selection. The terminal operator need only specify the transaction identifier name associated with the program selected for execution.

Program Loading

The subroutine interface responds to an interrupt from a terminal by loading the requested program specified by the transaction identifier. It routes subsequent operator entries to the associated program.

Multiple terminals using the same program use a single copy of the program. If different terminals are using different programs, the interface program will load separate copies for simultaneous program execution. When the terminal operator initially requests a program for execution, a copy of the program is loaded into any partition that meets the specifications of the program's TID entry.

Reserved Program Function Keys

Two program function keys are reserved:

- PF3 signals the interface program to terminate the current program and display the primary menu screen.
- PF6 signals the Event Driven Executive to print the contents of the current screen on the device specified by the HDCOPY parameter of the TERMINAL statement (for 4978/4979 terminals only). Normally, this device is the device specified for $SYSPRTR. Note that for the 3101 terminal, PF6 does not print the screen.

Application Program Design Considerations

Your applications are processed as independent Communications Facility transaction-processing programs, and loaded by the Communications Facility high-speed loader. Data sets are pre-bound as part of the initialization of $.PD during the startup of the Communications Facility.

Your application programs should adhere to the following conventions:

- No subtasks should be active across calls to the interface program.
- No system-wide resources should be enqueued across calls to the interface program.
- Application programs cannot use overlays or segmentation.
- Application programs must be written as subroutines named MTMSUB and designed to receive four parameters at initiation: input buffer, output buffer, TEB address, and IIB address. COBOL applications receive a fifth parameter, save area address.
- Application programs should use the interface program for all terminal I/O (other than spooled output) and disk I/O.
- All I/O should be complete before any call to the subroutines.
- Application programs should terminate only through calls to the interface program (CALL MENU) and should not issue any STOP RUN (COBOL), PROGSTOP, ENDTASK, or DETACH (EDL) instructions.
- Error exit routines should terminate with a CALL MENU.

USING THE SUBROUTINES IN AN EDL PROGRAM

This chapter explains EDL coding considerations, shows how to issue each subroutine call from an EDL program, and gives EDL sample programs.

EDL Programming Considerations

Your EDL application must be written as a subroutine, and must be defined to accept four parameters (input buffer, output buffer, TEB, and IIB). In addition, your program must use EXTRN statements to identify the subroutines. The subroutine name MTMSUB must also appear on the ENTRY statement. For example:

```
ENTRY      MTMSUB
EXTRN      ACTION,BEEP,CHGPAN,CYCLE,FAN
EXTRN      FILEIO,FTAB,GETCUR,LINK,LINKON,MENU
EXTRN      SETCUR,SETPAN
SUBROUT    MTMSUB,INPUT,OUTPUT,TEB,IIB
```

Format of Subroutine Calls

You use the EDL CALL statement to call the subroutines. For example, the statement to call SETPAN is:

CALL SETPAN,(*dsname*), (*code*)

This call passes the parameters *dsname* and *code* to the interface program.

Note that the buffer used for requests must be large enough to hold the largest possible record. The interface program does not truncate records if they are too large for the buffer. It reads or writes the requested size record regardless of the size of the buffer.

Creating Your Save Area

When your program requests a response from the terminal operator, your program is purged out of storage so other terminals may use the storage area while the operator is keying in new data. When the operator response is complete and storage is available, your program is reloaded into storage and given control at the next sequential instruction after the instruction that caused the program to be purged.

To maintain modified data for an application program that has been purged from storage, the interface program will save a specified data area in the program. You must collect the data you want saved in a save area defined in your EDL stub program. The save area must be large enough to contain all the data constants required for saving by the EDL application program.

You build the save area by specifying an ENTRY statement with the name SAVEAREA. The save area in the program would be coded as follows:

```
SAVEAREA   DATA   A(ENDSAVE-SAVEAREA)
COUNT      DATA   F'0'                    SAVED RECORD COUNT
PROGRAM    DATA   CL8'                    NAME OF PROGRAM
ENDSAVE    EQU    *
```

Note that the interface program reads and writes the save area to disk in the $.WSCIMG partitioned data set. The input and output buffers in the EDL stub program are used as temporary storage transfer areas. Therefore, be sure that the total size of the input and output buffers is greater than the size of the defined save area. Further, there is a restriction that the total save area can be no larger than 1920 bytes.

ACTION—Perform Terminal I/O

ACTION begins the cycle of writing prompts to the terminal and receiving responses from the user. If a CALL SETPAN has been executed previously during this session, it writes the screen image from the $.WSCIMG data set to the screen and scatter-writes the output buffer into the unprotected fields on the screen. If no SETPAN precedes the ACTION, ACTION writes only the output buffer. The terminal then waits for operator input and reenters your application (with operator input in the input buffer) at the next sequential instruction after CALL ACTION.

CALL ACTION Format

[*label*] CALL ACTION

BEEP—Sound Tone

BEEP sounds the tone (if the terminal has this feature) following the next output cycle. If the terminal doesn't have the feature, or if the terminal is a 4979 (which has no tone feature), this request is ignored. The current display and cursor position for the 4978, 4979, and 3101 are not affected.

CALL BEEP Format

[*label*] CALL BEEP

CHGPAN—Change Panel

After a CALL SETPAN, the protected characters of the screen panel specified have been displayed at the terminal. You can add data to the image before the next output cycle; the data is displayed as protected data. If you do add data, you must also use CALL CHGPAN to inform the interface program of the row, column, and data to be written in the protected area of the screen. This process allows applications to develop protected screen panels dynamically.

CALL CHGPAN Format

[*label*] CALL CHGPAN,(*row*),(*col*),(*len*),(*data*)

row
is the label of a word that contains the number of the row where the data is to be displayed on the terminal. Allowable row numbers are 0-23; row 0 is the top line of the screen.

col
is the label of a word that contains the number of the column where the data is to be displayed on the terminal. Allowable column numbers are 0-79; column 79 is the rightmost position of a row.

length
is the label of a word that contains the number of characters in the data field. Allowable lengths are 1-1920.

data
is the label of the data field to be displayed on the terminal.

CALL CHGPAN Coding Example

```
         CALL  CHGPAN,(ROW),(COL),(LEN),(SETERR)
          •
          •
          •
SETERR   DATA  CL12'SETPAN ERROR'
ROW      DATA  F'5'
COL      DATA  F'0'
LEN      DATA  F'12'
```

CYCLE—Swap Out

When CALL CYCLE is executed, the program may be made available to other terminals. The program save area is preserved. SETPAN or CHGPAN instructions will be executed to display written data.

After the program has processed input from all other terminals, control returns to the instruction after the CALL CYCLE.

CALL CYCLE Format

[*label*] CALL CYCLE

FILEIO—Perform Disk I/O

FILEIO performs disk I/O on direct and indexed files.

CALL FILEIO Format

[*label*] CALL FILEIO,(*fca*),(*buff*),(*rc*)

fca
: is the label of a file control area (FCA)—a table containing the parameters that describe the requested I/O operations. The meaning of some of the fields depends on the request type specified.

The FCA format for direct files is shown in Figure 5; the FCA format for indexed files is shown in Figure 6.

buff
: is the name of the user program I/O buffer. This buffer contains the record to be written or receives the record read.

rc
: is the name of the 2-byte field to contain the return code returned by FILEIO. This can be a FILEIO return code, a system error code, or a code passed from the Indexed Access Method.

Byte Displacement	Field Contents	Description
0	Request type	A 2-byte EBCDIC request (valid request types are shown in Figure 7).
4	Data Set Name	An 8-byte EBCDIC data set name, left-justified and padded with blanks.
12	Number of Records	A word specifying the number of 256-byte records to be read or written.
14	EOD Record	The 2-word system-maintained logical EOD record number passed back to the application after each direct file READ or WRITE.
18	Relative Record Number (RRN)	A 2-word value for the RRN. The first record is record number 1.
22	Volume Name	A 6-byte EBCDIC volume name, left-justified and padded with blanks.

Figure 5. FILEIO FCA Format for Direct Files

CALL FILEIO Indexed Access Method Considerations

FILEIO uses the parameters provided to create a parameter list for an Indexed Access Method supervisor call. Therefore, it is important to understand Indexed Access Method operation.

FILEIO executes a file cleanup routine after each call to ACTION, LINK, LINKON, or CYCLE. If any record locks have not been released, the cleanup routine releases these records to prevent any deadlock situations.

Byte Displacement	Field Contents	Description
0	Request type	A 2-byte EBCDIC request (valid request types are shown in Figure 7).
4	Data Set Name	An 8-byte EBCDIC data set name, left-justified and padded with blanks.
12	Key Relation	A 2-byte EBCDIC key relation operator, either GT, GE, or EQ (required only if request type is GETD, GETS, GTDU, or GTSU).
14	Key Length	A word specifying the length of the key to be used for retrieval. If the length specified is less than the actual key length, the first *n* bytes of the key are used.
16	Key Location	The address of the key to be used.
18	Reserved	Must be 0.
22	Volume Name	A 6-byte EBCDIC volume name, left-justified and padded with blanks.

Figure 6. FILEIO FCA Format for Indexed Files

Direct File Request Types	
READ	Read the record defined by the RRN field of the FCA into the user-provided buffer.
SEOD	Set the system-maintained EOD pointer to the record number provided in the RRN field of the FCA. This number should range from 1 to the EOF record of the file. This request is normally issued after the last record is written to the data set, but you may issue it any time you want to establish a logical end-of-file (EOF).
WRIT	Write the record defined by the RRN field of the FCA into the user-provided buffer.
Indexed File Request Types	
GETD	Get operation, direct read (GET)[1]
GETS	Get operation, sequential read (GETSEQ)
GTDU/GTRU	Direct get, update mode (GET)
GTSU	Sequential get, update mode (GETSEQ)
ICLS	Close an indexed data set (DISCONN)
IDEL	Delete operation (DELETE)
PUTD	Put operation, delete mode (PUTDE)
PUTN	Put operation, new mode, add a record to the file (PUT)
PUTU	Put operation, update mode (PUTUP)
RELR	Release a record held for update (RELEASE)
RELS	Release from sequential processing mode (ENDSEQ)
[1]Indexed file requests call the Indexed Access Method function shown in parentheses. Files are accessed in the PROCESS mode and are shared.	

Figure 7. File Request Types

This procedure will ensure data integrity on update:

1. Get record.
2. Save record contents.
3. Display to operator.
4. Get with update.
5. Ensure that record contents are unchanged.
6. Put with update.
7. Display to operator.

If sequential processing has been initiated on any indexed files, the FILEIO cleanup routine also releases those files from sequential processing mode. Thus, to continue sequential processing from the same key, the application should save the last key before calling ACTION, CYCLE, LINK, or LINKON. If you want to get sequential records and any of these CALL functions intervene, use GETD with the greater than key relation.

You can scan an indexed file from beginning to end by use of a sequence of "get sequential" (GETS) operations. The first GETS in a sequence should specify a key of all nulls (X'00') and a key relational operator of greater than (C'GT'). When executed, this initial GETS operation will receive the first record in the file (following the record, if any, for which the key is all nulls.) Subsequent GETS will retrieve the records following the first, in sequence.

FILEIO Return Codes

-1 Successful operation.

201 Data set not found.

203 No file table entries are available; all have updates outstanding.

206 Invalid function request type (this is returned for a valid Indexed Access Method function if the Indexed Access Method link module is not linked with the interface program).

207 Invalid key operator.

208 SEOD record number greater than data set length

Other return codes may be returned by the Indexed Access Method or by the system data management support.

CALL FILEIO Direct Access Coding Example (File Scan)

This example reads record number 1 to obtain the logical end-of-data. It then sets the record number to 1 and reads record 1, and successively higher records, until logical end-of-file is reached.

```
* GET EOD  (RETURNED BY READ OPERATION)
        MOVE RRN,1,DWORD
        CALL FILEIO,(FCA),(BUFFER),(RC)
* PROCESS FILE FROM RRN-1 TO EOD
        MOVE RRN,0,DWORD
LOOP    ADD  RRN,1,PREC-D
        IF   (RRN,GT,EOD,DWORD),GOTO,EXIT
          CALL FILEIO,(FCA),(BUFFER),(RC)
          •
          •
          •
          GOTO  LOOP
        ENDIF
EXIT    EQU   *
          •
          •
          •
* FILE CONTROL AREA
FCA     EQU  *
REQTYPE DATA   CL4'READ'
DSNAME  DATA   CL8'A'
NUMREC  DATA   F'1'
EOD     DATA   D'0'
RRN     DATA   D'0'
VOLNAME DATA   CL6'EDX002'
```

CALL FILEIO Indexed Access Coding Example

In this example, change *data set name*, *key length*, and *key location* if you are using secondary keys.

```
        CALL FILEIO,(FCA),(BUFFER),(RC)
          •
          •
          •
* FILE CONTROL AREA
FCA     EQU  *
REQTYPE DATA   CL4'GETD'
DSNAME  DATA   CL8'CUSTMAST'
RELOP   DATA   CL2'EQ'
KEYLEN  DATA   F'6'
KEYLOC  DATA   A(KEYFLD)
        DATA   2F'0'
VOLNAME DATA   CL6'EDX003'
          •
          •
          •
KEYFLD  DATA   CL6'069592'
BUFFER  DATA   256X'0'
RC      DATA   F'0'
```

FTAB—Build Unprotected Field Table

FTAB sets up a table that describes the unprotected action field areas in the input buffer following a CALL ACTION operation. You can use this table to format the output buffer before a CALL ACTION and to position the cursor to a specific field or to a precise location within a field.

FTAB is time-consuming; use it with care. If possible, perform the CALL SETPAN and CALL FTAB operations in the beginning of the application outside the normal looping operation of CALL ACTION and CALL FILEIO.

Note that you must define the FTAB data table in the program's save area to be saved between calls to ACTION, CYCLE, etc.

CALL FTAB Format

[*label*] CALL FTAB,(*table*),(*size*),(*code*)

table
The *table* operand is made up of a sequence of 3-word entries. Each 3-word entry describes an unprotected field of the screen image in the input buffer. The first word is the row position; the second word is the column position; and the third word is the length. The sequence begins at the location of the variable named in the table operand; it is repeated for each successive field of the screen.

This is an example of the table format:

TABLE	row	(word 1 of the first field)
	column	(word 2 of the first field)
	length	(word 3 of the first field)
TABLE+6	row	(second field)
	column	
	length	
TABLE+12	row	(third field)
	column	
	length	
•	•	
•	•	
•	•	
n		

where *n* is equal to the value of the *size* operand.

Unused fields in the FTAB table are always set to zero.

size
is 1 word long and contains the number of entries in the table. This decimal value can be in the range 1 to 32767.

code
is the name of a 1-word field reserved for a return code from FTAB.

FTAB Return Codes

-1 Successful return.

1 No data fields found.

2 Data table truncated.

CALL FTAB Coding Example

```
          CALL FTAB,(TABLE),(SIZE),(RC)
•
•
•
TABLE     DATA  30F'0'
SIZE      DATA  F'10'
RC        DATA  F'0'
•
•
•
```

GETCUR—Get Cursor Position

GETCUR gets the cursor position returned to the program after a CALL ACTION.

CALL GETCUR Format

[*label*] CALL GETCUR,(*row*),(*column*)

row
is the label of a word to contain the row number of the cursor. Possible row numbers are 0-23; row 0 is the top line of the screen.

column is the label of a word to contain the column number of the cursor. Possible column numbers are 0-79; column 79 is the rightmost position of a row.

CALL GETCUR Coding Example

```
          CALL     ACTION
          CALL     GETCUR,(ROW),(COLUMN)
*
ROW       DATA     F'0'               CURSOR ROW POSITION
COLUMN    DATA     F'0'               CURSOR COLUMN POSITION
```

LINK—Transfer Control to Another Program

A call to LINK causes the named application program, which uses the work session controller high-level language subroutines, to be loaded and executed (replacing the current program). If a SETPAN or CHGPAN precedes the LINK, the contents of the input buffer are displayed for 4978, 4979, or 3101 terminals and the buffer is freed. The output buffer is passed unchanged to the linked-to program.

If the transaction identifier is invalid or cannot be found, control returns to the caller; therefore, any return to your program from CALL LINK is an error condition.

CALL LINK Format

[*label*] CALL LINK,(*tid*)

tid
is the name of a variable that contains the 4-byte name of a transaction identifier in the $.SYSPD data set. The TID specifies the transaction-processing program to be linked to (right padded with blanks, if necessary).

CALL LINK Coding Example

```
         CALL    LINK,(PROG)
         GOTO    ERROR
          •
          •
          •
PROG     DATA    C'MTM4'
```

LINKON—Transfer Control to Another Program with Output Cycle

A call to LINKON provides the same function as CALL LINK, except that a screen is displayed and the interface program waits for an operator response. The named program is then entered at its entry point with the input buffer containing the unprotected characters from the screen.

If the transaction identifier is invalid or cannot be found, control returns to the caller; therefore, any return to your program from CALL LINK is an error condition.

CALL LINKON Format

[label] CALL LINKON,(*tid*)

tid
is the name of a variable that contains the 4-byte name of a transaction identifier in the $.SYSPD data set. The TID specifies the transaction-processing program to be linked to (right padded with blanks, if necessary).

CALL LINKON Coding Example

```
         CALL   LINKON,(LNKPGM)
          •
          •
          •
LNKPGM   DATA   'MTM4'
          •
          •
          •
```

MENU—Return to Primary Menu

CALL MENU immediately terminates the current program and causes the primary menu screen to be displayed. The operator can get back to the primary menu at any time by pressing PF3 on a 4979, 4978, or 3101.

CALL MENU Format

[label] CALL MENU

SETCUR—Set Cursor Position

SETCUR specifies the position at which the cursor is to be displayed for the next output cycle. The cursor position is expressed as a pair of row and column coordinates on the screen.

Each screen panel specifies a cursor position to be used while the screen is active (until the next SETPAN or CHGPAN). CALL SETCUR permits you to override the cursor position established by a previous SETPAN or CHGPAN. The cursor is moved on the next output cycle.

CALL SETCUR Format

[label] CALL SETCUR,(*row*),(*col*)

row
: is the label of a word that contains the number of the row at which the cursor is to be set. Allowable row numbers are 0-23; row 0 is the top line of the screen.

col
: is the label of a word that contains the number of the column at which the cursor is to be set. Allowable column numbers are 0-79; column 79 is the rightmost position of a row.

CALL SETCUR Coding Example

To set the cursor position to row 1, column 12 of a static-screen display:

```
         CALL    SETCUR,(ROW),(COLUMN)
*
ROW      DATA    F'1'               ROW 1
COLUMN   DATA    F'12'              COLUMN 12
```

SETPAN—Write Buffer to Screen

SETPAN causes the specified screen format name to be saved and sets a switch to cause the screen format to be written to the screen during the next output cycle. Any nulls in the screen image will be written unprotected; all other characters will be written protected. The cursor position for the next display after SETPAN will be set to the first unprotected character position. Unprotected defaults that were specified when the screen was built are not displayed by SETPAN.

CALL SETPAN Format

[label] CALL SETPAN,(*dsname*),(*code*)

dsname
is the name of a variable that contains the 8-byte data set name of the screen format in the $.WSCIMG data set.

code
is the label of a word in which SETPAN will place a return code.

SETPAN Return Codes

-1 Successful, new panel in buffer.

CALL SETPAN Coding Example

```
          CALL     SETPAN,(SCREEN01),(RC)
           •
           •
           •
SCREEN01  DATA     C'SCRNTST1'
RC        DATA     F'0'
```

Link Editing Your Programs

You need to link edit your EDL application program, your EDL stub program, and the interface program. You can choose from two versions of the interface program: O$HLSE if you don't use the indexed access method, and O$HLSEI if you do. If you choose O$HLSEI, your link control data set must include an INCLUDE IAM,ASMLIB statement.

Your link control data set must also include either an INCLUDE O$SEOD,ASMLIB statement (if your program uses the SEOD function) or an INCLUDE O$NOSEOD,ASMLIB statement (if it doesn't). O$SEOD requires approximately 3700 bytes of additional storage in the application program.

Figure 8 shows the link control data set for an EDL application without the indexed access method.

```
 INCLUDE   O$BASEE,EDX003            EDL STUB PROGRAM
 INCLUDE   O$MTMSUB,EDX003           EDL APPLICATION SUBROUTINE
 INCLUDE   O$HLSE,ASMLIB             INTERFACE MODULE
 INCLUDE   O$SEOD,ASMLIB             SEOD MODULE
*INCLUDE   O$NOSEOD,ASMLIB           NO SEOD MODULE
 LINK CFMTM4,EDX002 REPLACE END      UPDATE CONTROL STATEMENT
```

Figure 8. Link Control Data Set (No IAM)

Figure 9 shows the link control data set for an EDL application with the indexed access method.

```
 INCLUDE   O$BASEE,EDX003            EDL STUB PROGRAM
 INCLUDE   O$MTMSUB,EDX003           EDL APPLICATION SUBROUTINE
 INCLUDE   O$HLSEI,ASMLIB            INTERFACE MODULE
*INCLUDE   O$SEOD,ASMLIB             SEOD MODULE
 INCLUDE   O$NOSEOD,ASMLIB           NO SEOD MODULE
 INCLUDE   IAM,ASMLIB                IAM STUB
 LINK CFMTM4,EDX002 REPLACE END      UPDATE CONTROL STATEMENT
```

Figure 9. Link Control Data Set (IAM)

Sample Programs

This section presents sample EDL programs that use the work session controller high-level language subroutines.

Screen Display Techniques

This section shows EDL coding examples of some screen display techniques.

Screen Retrieval and Display

In this example, a program retrieves a screen created by $IMAGE, displays it, and reads operator input. It retrieves a screen image is retrieved through a CALL SETPAN.

A subsequent call to ACTION performs terminal output and input. ACTION displays the contents of the output buffer as unprotected data on the static screen from the call to SETPAN. After the operator presses ENTER or a program function key, ACTION reads the operator input into the input buffer.

ACTION also places a program function key identifier into the interrupt information byte (IIB).

The following example shows an application that retrieves a screen image, displays it, and manipulates the operator input data that has been read into the input buffer. It also checks whether the operator has pressed a program function key. In the example, a screen image MAINSCRN is displayed on the terminal from the data set $.WSCIMG. A return code of -1 indicates successful completion.

```
***********************************************************************
*       RETRIEVE AND DISPLAY SCREEN, ACCEPT OPERATOR INPUT, AND       *
*       CHECK IF PROGRAM FUNCTION KEY USED.                           *
***********************************************************************
          SUBROUT MTMSUB,INBUFF,OUTBUFF,TEB,PFKEY
          ENTRY   MTMSUB
          EXTRN   SETPAN,ACTION
           •
           •
           •
          CALL    SETPAN,(MAINSCRN),(RC)  DISPLAY SCREEN FROM $.WSCIMG
          IF      (RC,NE,-1)              IF SETPAN ERROR
           •
           •
           •
          ENDIF                           ENDIF
          CALL    ACTION                  GET RESPONSE
          MOVE    #1,PKFEY                GET PF KEY ADDRESS
          IF      ((0,#1),EQ,4)           IF PF KEY FOUR
          IF      (RC,NE,-1)              IF SETPAN ERROR
           •
           •
           •
          ENDIF                           ENDIF
           •
           •
           •
***********************************************************************
*       DATA AREAS AND EQUATES                                        *
***********************************************************************
MAINSCRN DATA     CL8'MAINSCRN'           SCREEN MEMBER NAME
RC        DC      F'0'                    RETURN CODE FIELD
           •
           •
           •
```

Dynamic Screen Creation

The following example shows how to modify a screen dynamically. This technique is especially useful for one-line error messages; you can put the message in the appropriate place on the screen with a CALL CHGPAN. For example, the following code displays the message "SETPAN ERROR" on the fifth line of a terminal.

```
************************************************************************
*       MODIFY SCREEN DYNAMICALLY                                      *
************************************************************************
        SUBROUT MTMSUB,INBUFF,OUTBUFF,TEB,PFKEY
         •
         •
         •
        CALL CHGPAN,(ROW),(COL),(LEN),(SETERR) CHANGE SCREEN
        CALL ACTION                            DISPLAY MSG, GET RESP
         •
         •
         •
************************************************************************
*       DATA AREAS AND EQUATES.                                        *
************************************************************************
SETERR  DATA    CL12'SETPAN ERROR'     MESSAGE TO BE DISPLAYED
ROW     DATA    F'5'                   DISP TO FIFTH LINE OF SCREEN
COL     DATA    F'0'
LEN     DATA    F'12'
         •
         •
         •
```

Updating an Indexed File

You might want an application to update indexed data sets based on operator input. This process requires a get for update (FILEIO GTDU), a screen write and read (SETPAN and ACTION), and then a put for update (FILEIO PUTU).

Applications must release data sets from update mode before an ACTION is performed; therefore, the program must perform a get direct without update before the ACTION, and a get direct for update after the ACTION.

A problem might arise because the ACTION can cause the issuing program to be swapped out. If another program were to update the record during that time, the update that program makes would be lost when the original program updates the file. To avoid this problem, each program should ensure that any record it modifies has not been modified during the ACTION, as shown in the following example.

```
**********************************************************************
*       UPDATE INDEXED FILE                                          *
**********************************************************************
        .
        .
        .
        MOVE    REQTYPE,GETD,(4,BYTES)        SET UP GET DIRECT
        CALL    FILEIO,(FCA),(OLDBUF),(RC)    GET RECORD
        CALL    SETPAN,...                    RETRIEVE SCREEN
        CALL    ACTION                        DISP SCREEN, ACCEPT INPUT
        MOVE    REQTYPE,GTDU,(4,BYTES)        SET UP GET DIR UPDATE MODE
        CALL    FILEIO,(FCA),(NEWBUF),(RC)    GET RECORD FOR UPDATE
        IF      (OLDBUF,NE,NEWBUF,80)         IF RECORD HAS CHANGED
          CALL  ALERT                         CALL ERROR ROUTINE
        ENDIF                                 ENDIF
        MOVE    #1,INBUF                      GET INPUT BUFFER ADDRESS
        MOVE    NEWBUF,(0,#1),(...,BYTES)     SAVE RECORD
        MOVE    REQTYPE,PUTU,(4,BYTES)        SET UP PUT UPDATE
        CALL    FILEIO,(FCA),(NEWBUF),(RC)    UPDATE RECORD
        .
        .
        .
**********************************************************************
*       DATA AREA.                                                   *
**********************************************************************
GETD    DATA    CL4'GETD'                 GET DIRECT FCA CODE
GTDU    DATA    CL4'GTRU'                 GET DIRECT FCA CODE
PUTU    DATA    CL4'PUTU'                 PUT UPDATE FCA CODE
RC      DATA    F'0'                      RETURN CODE FIELD
        .
        .
        .
```

File Maintenance Transaction Application

This example consists of a pair of programs that perform a simple file maintenance task. It reads or writes a single record, or sets an end of data (EOD) marker.

The first program displays a screen that requests the file parameters, which include data set name and relative record number. It then issues a CALL LINK to execute the second program, passing the file parameters.

The second program builds a file control area (FCA) from the file parameters and performs the requested file I/O operation. The results of the operation are displayed on the screen, and the program ends.

Note: The examples in this section operate only on data sets of less than 32K bytes.

The following is a detailed explanation of each program statement in the sample program listing and the effects of program execution of the application.

The first statements in the first program are declarations:

```
EXTRN    BEEP,SETPAN,MENU,ACTION,LINK
ENTRY    MTMSUB,SAVEAREA
SUBROUT  MTMSUB,INBADDR,OUTBADDR,TEBADDR,IIBADDR
```

EXTRN declares the subroutine functions as external, so the application can access them. ENTRY declares the application as an entry point and locates the user save area for the interface program. The application is a subroutine, as shown in the SUBROUT statement, called MTMSUB. It accepts four parameters, the addresses of the input buffer, output buffer, terminal environment block and interrupt information byte.

The next instructions put the buffer addresses into registers 1 and 2:

```
MOVE   #1,INBADDR
MOVE   #2,OUTBADDR
```

Initial data is loaded into the output buffer for display when the CALL ACTION is issued:

```
         MOVE (14,#2),INITDATA,(8,BYTES)
          •
          •
          •
INITDATA DATA CL8'READ0001'
```

The terminal is prepared to sound the audible alarm:

```
CALL BEEP
```

A screen image is retrieved from a disk data set and written to the terminal:

```
         CALL   SETPAN,(REQSCRN),(RC)
          •
          •
          •
RC       DATA   F'0'
REQSCRN  DATA   CL8'REQ'
```

A screen image consists of protected data, which may be considered a screen template or form. The protected data is a screen-sized (24 by 80) image consisting of character data which is displayed, and fields of nulls used for data entry. Default data in the output buffer is written by the ACTION call into these null fields, and operator input is read from them.

After the call to SETPAN, the terminal screen contains the screen as shown in SCREEN 1, with five null fields as shown by dollar signs. The $ is just to illustrate where the fields are on the screen; null fields are actually displayed as blanks.

SCREEN 1

```
          DATA SET, VOLUME NAME -->$$$$$$$$:,$$$$$$
          REQUEST (READ, WRIT, SEOD) -->$$$$
          RELATIVE RECORD NUMBER -->$$$$
          NUMBER OF RECORDS -->1
          DATA TO BE WRITTEN:
----------------------------------------------------------
$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$
----------------------------------------------------------
```

The output buffer contains data used to initialize unprotected input fields. It consists of 14 blanks, followed by READ0001, followed by 80 blanks. When written to the unprotected portion of the screen, the terminal appears as shown in SCREEN 2.

A test of the return code from SETPAN is done. If the return code does not indicate a successful return, the program ends by giving control to the primary menu routine:

```
IF     (RC,NE,-1)
     CALL MENU
ENDIF
```

The active terminal type can be determined by testing the TEB. Since 3101 terminals require that a CALL FTAB be issued and 4978 terminals do not because they support scatter write, the following code will make the program more efficient:

```
MOVE       #1,TEBADDR
IF         ((0,#1),EQ,1)
  CALL     FTAB,(TABLE),(SIZE),(RC)
ENDIF
```

The next instruction calls the ACTION routine to display the contents of the output buffers, and reads the operator response:

```
CALL ACTION
```

The effects of CALL ACTION are:

- Write the output buffer contents , if any, into the null fields as unprotected characters.
- Wait for the operator to enter data and press ENTER or a PF key.
- Read the contents of the unprotected fields (the operator input) into the input buffer.

This results in SCREEN 2 appearing on the terminal, where the default characters are highlighted (shown by underlining here).

SCREEN 2

```
DATA SET, VOLUME NAME ==>
REQUEST (READ, WRIT, SEOD) ==>READ
RELATIVE RECORD NUMBER ==>0001
NUMBER OF RECORDS ==>1
DATA TO BE WRITTEN:
----------------------------------------------------------

----------------------------------------------------------
```

The operator then enters data, changing the default data associated with relative record number. For example, to read the third record of data set 'K" ' on volume EDX013, the underlined data on SCREEN 3 would be entered.

SCREEN 3

```
DATA SET, VOLUME NAME ==>K    ,EDX013
REQUEST (READ, WRIT, SEOD) ==>READ
RELATIVE RECORD NUMBER ==>0003
NUMBER OF RECORDS ==>1
DATA TO BE WRITTEN:
----------------------------------------------------------

----------------------------------------------------------
```

The operator signals that the input is ready by pressing ENTER or a PF key. ACTION then completes the input cycle by reading the contents of the unprotected fields into the input buffer:

```
K        EDX013READ0003              (80 blanks)
```

For the second program to receive the file parameters, they must be passed through the output buffer. The next instruction moves the input data from the input buffer to the output buffer:

```
MOVE  (0,#2),(0,#1),(106,BYTES)
```

Finally, a CALL LINK to MTM3 is made. The transaction identifier is referened in the IOPROG statement.

```
          CALL   LINK,(IOPROG)
           •
           •
           •
IOPROG    DATA   CL8'PROG2'
```

A call to MENU to terminate the transaction is placed after the LINK, in case the LINK is unsuccessful:

```
CALL MENU
```

The first four lines of PROG2 are similar to those of PROG1, except that other functions are declared external, and only register 2 is assigned a buffer address:

```
EXTRN    FILEIO,SETPAN,MENU,ACTION
ENTRY    MTMSUB,SAVEAREA
SUBROUT  MTMSUB,INBADDR,OUTBADDR,TEBADDR,IIBADDR
MOVE     #2,OUTBADDR
```

At this point the output buffer (pointed to by register #2) contains various file parameters. A file control area (FCA) is constructed using these parameters. For example, the request type is moved from the output buffer to the FCA:

```
         MOVE FCAREQ,(REQTYPE,#2),(4,BYTES)
          •
          •
          •
FCAREQ   DATA    CL4' '
          •
          •
          •
REQTYPE  EQU     14
```

Similarly, other fields must be moved, and relative record number must be converted to numeric:

```
* SET UP FILE CONTROL AREA AND BUFFER.
         MOVE     FCAREQ,(REQTYPE,#2),(4,BYTES)    REQUEST TYPE
         MOVE     FCADSN,(DSNAME,#2),(8,BYTES)     DATA SET NAME
         MOVE     FCANUM,1                         NUMBER OF RECS
         CONVTD   FCARRN,(RRN,#2),FORMAT=(4,0,I)   CONVERT RRN
         MOVE     FCAVOL,(VOLNAME,#2),(6,BYTES)    VOLUME NAME
         MOVE     BUFFER,(BUFFDISP,#2),(80,BYTES) DATA BUFFER
          •
          •
          •
* FILE CONTROL AREA.
FCA      EQU      *
FCAREQ   DATA     CL4' '               REQUEST TYPE
FCADSN   DATA     CL8' '               DATA SET NAME
FCANUM   DATA     F'1'                 NUMBER OF RECORDS
         DATA     F'0'
FCAEOD   DATA     F'0'                 EOD RELATIVE RECORD NUMBER
         DATA     F'0'
FCARRN   DATA     F'0'                 RELATIVE RECORD NUMBER
FCAVOL   DATA     CL6' '               VOLUME NAME
          •
          •
          •
* EQUATES FOR OUTPUT BUFFER DATA.
DSNAME   EQU      0                    DATA SET NAME
VOLNAME  EQU      8                    VOLUME NAME
REQTYPE  EQU      14                   REQUEST TYPE
RRN      EQU      18                   RELATIVE RECORD NUMBER
BUFFDISP EQU      22                   BUFFER DISPLACEMENT
EODRRN   EQU      102                  EOD RRN DISPLACEMENT
RCDISP   EQU      106                  RETURN CODE DISPLACEMENT
```

A screen image with which to display the file data is retrieved, and the return code is checked. This screen is similar to the previous screens shown with the addition of two new fields.

```
          CALL SETPAN,(LISTSCRN),(RC)
          IF   (RC,NE,-1)
               CALL MENU
          ENDIF
           •
           •
           •
LISTSCRN  DATA  CL8'LST'
```

At this point the image depicted in SCREEN 4 is in the buffers. Since there is no default data, the output buffer is empty.

SCREEN 4

```
          DATA SET, VOLUME NAME ==>
          REQUEST (READ, WRIT, SEOD) ==>
          RELATIVE RECORD NUMBER ==>
          NUMBER OF RECORDS ==>1
          DATA TO BE WRITTEN:
----------------------------------------------------------

----------------------------------------------------------
          EOD RELATIVE RECORD NUMBER ==>
          RETURN CODE ==>
```

The actual FILEIO operation is performed, specifying the FCA, a buffer, and a return code:

```
          CALL     FILEIO,(FCA),(BUFFER),(RC)
           •
           •
           •
RC        DATA     F'0'
BUFFER    DATA     256X'0'
```

Note that the buffer is 256 bytes in length (the length of an Event Driven Executive record) even though only the first 80 bytes are used.

Now that all the file data is available, it is placed in the output buffer so that it can be displayed. The data is taken from the FCA, the buffer and return code, and concatenated so that it may be written into the unprotected fields of the screen image:

```
* PUT DATA INTO OUTPUT BUFFER SO IT WILL BE DISPLAYED.
         MOVE     (REQTYPE,#2),FCAREQ,(4,BYTES)      REQUEST TYPE
         MOVE     (DSNAME,#2),FCADSN,(8,BYTES)       DATA SET NAME
         CONVTB   (EODRRN,#2),FCAEOD,FORMAT=(4,0,I)  CONV EOD RRN
         CONVTB   (RRN,#2),FCARRN,FORMAT=(4,0,I)     CONVERT RRN
         MOVE     (VOLNAME,#2),FCAVOL,(6,BYTES)      VOLUME NAME
         MOVE     (BUFFDISP,#2),BUFFER,(80,BYTES)    DATA
         CONVTB   (RCDISP,#2),RC,FORMAT=(4,0,I)      CONV RET CODE
```

The output buffer now looks as follows:

```
K       EDX013READ0003RECORD 3(72 blanks)0005-001
```

Both input and output buffers are displayed on the screen by the following call:

```
CALL ACTION
```

SCREEN 5

```
          DATA SET, VOLUME NAME -->K      :,EDX013
          REQUEST (READ, WRIT, SEOD) -->:READ
          RELATIVE RECORD NUMBER -->0003
          NUMBER OF RECORDS -->1
          DATA TO BE WRITTEN:
----------------------------------------------------------
RECORD 3:
----------------------------------------------------------
          EOD RELATIVE RECORD NUMBER -->0005
          RETURN CODE -->-001
```

A call to ACTION waits for operator input followed by an ENTER or PF key. In this case, no input is solicited; however, the use of ACTION allows the user to view the screen and press ENTER after the contents have been read. At that point the program ends.

```
CALL MENU
```

The following pages contain the applications used to perform the example previously shown.

EDL Sample Program 1

```
         EXTRN     BEEP,SETPAN,MENU,ACTION,LINK,FTAB
         ENTRY     MTMSUB,SAVEAREA
         SUBROUT   MTMSUB,INBADDR,OUTBADDR,TEBADDR,IIBADDR
         MOVE      #1,INBADDR            GET INPUT BUFF ADDRESS
         MOVE      #2,OUTBADDR           GET OUTPUT BUFF ADDRESS
* MOVE INITIAL DATA TO OUTPUT BUFFER.
         MOVE      (14,#2),INITDATA,(8,BYTES)
* BEEP UPON TERMINAL I/O.
         CALL      BEEP
* RETRIEVE SCREEN IMAGE AND ABORT IF ERROR.
         CALL      SETPAN,(REQSCRN),(RC)   GET SCREEN IMAGE
         IF        (RC,NE,-1)             OK?
           CALL      MENU                 NO
         ENDIF
* TEST FOR TERMINAL TYPE, CALL FTAB IF 3101.
         MOVE      #1,TEBADDR
         IF        ((0,#1),EQ,1)
            CALL     FTAB,(TABLE),(SIZE),(RC)
         ENDIF
* DISPLAY SCREEN IMAGE, READ OPERATOR RESPONSE.
         CALL      ACTION
* MOVE DATA FROM INPUT BUFFER TO OUTPUT BUFFER (106 BYTES).
         MOVE      (0,#2),(0,#1),(106,BYTES)
* LINK TO PROGRAM WHICH WILL PERFORM FILE I/O.
         CALL      LINK,(IOPROG)
* ABORT IF LINK FAILS.
         CALL      MENU
*************************************************************
*                                                           *
*        DATA ITEMS                                         *
*                                                           *
*************************************************************
INITDATA DATA     CL8'READ0001'
REQSCRN  DATA     CL8'REQ'               NAME OF REQUEST SCREEN
IOPROG   DATA     CL8'MTM3'              NAME OF I/O PROG TID
RC       DATA     F'0'                   RETURN CODE
SAVEAREA DATA     A(ENDSAVE-SAVEAREA)
SIZE     DATA     F'10'
TABLE    BUFFER   30,WORDS
ENDSAVE  EQU      *
         ENDPROG
         END
```

```
          EXTRN   FILEIO,SETPAN,MENU,ACTION
          ENTRY   MTMSUB,SAVEAREA
          SUBROUT MTMSUB,INBADDR,OUTBADDR,TEBADDR,IIBADDR
          MOVE    #2,OUTBADDR              GET O/P BUFFER ADDR
* SET UP FILE CONTROL AREA AND BUFFER.
          MOVE    FCAREQ,(REQTYPE,#2),(4,BYTES)    REQST TYPE
          MOVE    FCADSN,(DSNAME,#2),(8,BYTES)     DATA SET NAME
          MOVE    FCANUM,1                         NUMBER OF RECS
          CONVTD  FCARRN,(RRN,#2),FORMAT=(4,0,I)   CONVERT RRN
          MOVE    FCAVOL,(VOLNAME,#2),(6,BYTES)    VOLUME NAME
          MOVE    BUFFER,(BUFFDISP,#2),(80,BYTES)  DATA BUFFER
* RETRIEVE LISTING SCREEN AND ABORT IF ERROR.
          CALL    SETPAN,(LISTSCRN),(RC)
          IF      (RC,NE,-1)               GOT SCREEN IMAGE OK?
            CALL   MENU                    NO
          ENDIF
* TEST FOR TERMINAL TYPE, CALL FTAB IF 3101.
          MOVE    #1,TEBADDR
          IF      ((0,#1),EQ,1)
             CALL    FTAB,(TABLE),(SIZE),(RC)
          ENDIF
* PERFORM FILE I/O.
          CALL    FILEIO,(FCA),(BUFFER),(RC)
* PUT DATA INTO OUTPUT BUFFER SO IT WILL BE DISPLAYED.
          MOVE    (REQTYPE,#2),FCAREQ,(4,BYTES)     REQUEST TYPE
          MOVE    (DSNAME,#2),FCADSN,(8,BYTES)      DATA SET NAME
          CONVTB  (EODRRN,#2),FCAEOD,FORMAT=(4,0,I) CONV EOD RRN
          CONVTB  (RRN,#2),FCARRN,FORMAT=(4,0,I)    CONVERT RRN
          MOVE    (VOLNAME,#2),FCAVOL,(6,BYTES)     VOLUME NAME
          MOVE    (BUFFDISP,#2),BUFFER,(80,BYTES)   DATA
          CONVTB  (RCDISP,#2),RC,FORMAT=(4,0,I)     CONV RET CODE
* DISPLAY SCREEN IMAGE AND DATA.
          CALL    ACTION
* END PROGRAM.
          CALL    MENU
*************************************************************
*                                                           *
*         DATA ITEMS                                        *
*                                                           *
*************************************************************
*
LISTSCRN DATA     CL8'LST'          NAME OF LISTING SCREEN
RC       DATA     F'0'              RETURN CODE
BUFFER   DATA     256X'0'           DATA BUFFER
* FILE CONTROL AREA.
FCA      EQU      *
FCAREQ   DATA     CL4' '            REQUEST TYPE
FCADSN   DATA     CL8' '            DATA SET NAME
FCANUM   DATA     F'1'              NUMBER OF RECORDS
         DATA     F'0'
FCAEOD   DATA     F'0'              EOD RELATIVE RECORD NUMBER
         DATA     F'0'
FCARRN   DATA     F'0'              RELATIVE RECORD NUMBER
FCAVOL   DATA     CL6' '            VOLUME NAME
```

```
* EQUATES FOR OUTPUT BUFFER DATA.
DSNAME   EQU      0                DATA SET NAME
VOLNAME  EQU      8                VOLUME NAME
REQTYPE  EQU      14               REQUEST TYPE
RRN      EQU      18               RELATIVE RECORD NUMBER
BUFFDISP EQU      22               BUFFER DISPLACEMENT
EODRRN   EQU      102              EOD RRN DISPLACEMENT
RCDISP   EQU      106              RETURN CODE DISPLACEMENT
SAVEAREA DATA     A(ENDSAVE-SAVEAREA)
SIZE     DATA     F'10'
TABLE    BUFFER   30,WORDS
ENDSAVE  EQU      *
         ENDPROG
         END
```

USING THE SUBROUTINES IN A COBOL PROGRAM

This chapter shows how to issue each subroutine call from a COBOL program; gives COBOL coding considerations; and includes a COBOL sample program.

COBOL Programming Considerations

The PROGRAM-ID for all COBOL applications must be "MTMSUB". All parameters passed to the interface program must be level 01 or 77. The five parameters passed to the application (input buffer, output buffer, TEB, IIB, and save area), must be defined in the program's LINKAGE SECTION. The PROCEDURE DIVISION must contain the USING clause followed by the names given to the input buffer, output buffer, TEB, IIB, and save area, in that order.

Format of Subroutine Calls

You use the COBOL CALL statement to call the subroutines. For example, the statement to call SETPAN is:

CALL "SETPAN" USING SCREEN, RC.

This call passes the addresses of SCREEN and RC to the interface program.

The WORKING-STORAGE SECTION would include:

```
77  SCREEN PICTURE X(8) VALUE "SCRNNAME".
77  RC PICTURE 99 COMP.
```

Note that the buffer used for requests must be large enough to hold the largest possible record. The interface program does not truncate records if they are too large for the buffer. It reads or writes the requested size record regardless of the size of the buffer.

Creating Your Save Area

When your program requests a response from the terminal operator, your program is purged out of storage so other terminals may use the storage area while the operator is keying in new data. When the operator response is complete and storage is available, your program is reloaded into storage and given control at the next sequential instruction after the instruction that caused the program to be purged.

To maintain modified data for an application program that has been purged from storage, the interface program will save a specified data area in the program. You must collect the data you want saved in a save area defined in your EDL stub program. The save area must be large enough to contain all the data constants required for saving by the COBOL program. The modified data values and constants must be defined in the LINKAGE SECTION of the COBOL program. The PROCEDURE DIVISION USING statement must contain the keyword SAVEAREA.

Figure 10 shows how to code your save area.

Note that the interface program reads and writes the save area to disk in the $.WSCIMG partitioned data set. The input and output buffers in the EDL stub program are used as temporary storage transfer areas. Therefore, be sure that

```
DATA DIVISION.
WORKING STORAGE SECTION.
•
•
•
LINKAGE SECTION.
01 INPUT-BUFFER.
•
•
•
01 OUTPUT-BUFFER.
•
•
•
77 TEB                PIC S99 COMP.
77 IIB                PIC S99 COMP.
01 SAVEAREA.
   05 COUNT           PIC S99 COMP.
   05 PROG-NAME       PIC X(8).
       •
       •
       •
PROCEDURE DIVISION
   USING INPUT-BUFFER, OUTPUT-BUFFER, TEB, IIB, SAVEAREA.
   CALL "FAN".
```

Figure 10. Save Area Coding

the total size of the input and output buffers is greater than the size of the defined save area. Further, there is a restriction that the total save area can be no larger than 1920 bytes.

ACTION—Perform Terminal I/O

ACTION begins the cycle of writing prompts to the terminal and receiving responses from the user. If a CALL SETPAN has been executed previously during this session, it writes the screen image from the $.WSCIMG data set to the screen and scatter-writes the output buffer into the unprotected fields on the screen. If no SETPAN precedes the ACTION, ACTION writes only the output buffer. The terminal then waits for operator input and reenters your application (with operator input in the input buffer) at the next sequential instruction after CALL ACTION.

CALL ACTION Format

[*label*] CALL "ACTION".

BEEP—Sound Tone

BEEP sounds the tone (if the terminal has this feature) following the next output cycle. If the terminal doesn't have the feature, or if the terminal is a 4979 (which has no tone feature), this request is ignored. The current display and cursor position for the 4978, 4979, and 3101 are not affected.

CALL BEEP Format

[*label*] CALL "BEEP".

CHGPAN—Change Panel

After a CALL SETPAN, the protected characters of the screen panel specified have been displayed at the terminal. You can add data to the image before the next output cycle; the data is displayed as protected data. If you do add data, you must also use CALL CHGPAN to inform the interface program of the row, column, and data to be written in the protected area of the screen. This process allows applications to develop protected screen panels dynamically.

CALL CHGPAN Format

[*label*] CALL "CHGPAN" USING *row, col, len, data.*

row
is the label of a word that contains the number of the row where the data is to be displayed on the terminal. Allowable row numbers are 0-23; row 0 is the top line of the screen.

col
is the label of a word that contains the number of the column where the data is to be displayed on the terminal. Allowable column numbers are 0-79; column 79 is the rightmost position of a row.

len
is the label of a word that contains the number of characters in the data field. Allowable lengths are 1-1920.

data
is the label of the data field to be displayed on the terminal.

CALL CHGPAN Coding Example

```
WORKING-STORAGE SECTION.
77  ROW              PIC S99 COMP VALUE 0.
77  COLUMN           PIC S99 COMP VALUE 1.
77  LENGTH           PIC S99 COMP VALUE 12.
77  DATA             PIC X(12) VALUE 'SETPAN ERROR'.
     •
     •
     •
CALL "CHGPAN" USING ROW, COLUMN, LENGTH, DATA.
```

CYCLE—Swap Out

When CALL CYCLE is executed, the program may be made available to other terminals. The program save area is preserved. SETPAN or CHGPAN instructions will be executed to display written data.

After the program has processed input from all other terminals, control returns to the instruction after the CALL CYCLE.

CALL CYCLE Format

[*label*] CALL "CYCLE".

FAN—COBOL Return Interface

In a COBOL application, CALL FAN must be coded as the first executable CALL statement in the PROCEDURE DIVISION. Upon a return from a CALL ACTION, CALL FAN will transfer control to the next sequential instruction following that call.

CALL FAN Format

CALL "FAN".

CALL FAN Coding Example

```
PROCEDURE DIVISION USING IN-REC, OUT-REC, TEB, IIB, SAVEAREA.
BEGIN PROCESSING.
CALL "FAN".
```

FILEIO—Perform Disk I/O

FILEIO performs disk I/O on direct and indexed files.

CALL FILEIO Format

[*label*] CALL "FILEIO" USING *fca, buff, rc.*

fca
is the label of a file control area (FCA)—a table containing the parameters that describe the requested I/O operations. The meaning of some of the fields depends on the request type specified.

The FCA format for direct files is shown in Figure 11; the FCA format for indexed files is shown in Figure 12.

buff
is the name of the user program I/O buffer. This buffer contains the record to be written or receives the record read.

rc
is the name of the 2-byte field to contain the return code returned by FILEIO. This can be a FILEIO return code, a system error code, or a code passed from the Indexed Access Method.

Byte Displacement	Field Contents	Description
0	Request type	A 2-byte EBCDIC request (valid request types are shown in Figure 13).
4	Data Set Name	An 8-byte EBCDIC data set name, left-justified and padded with blanks.
12	Number of Records	A word specifying the number of 256-byte records to be read or written.
14	EOD Record	The 2-word system-maintained logical EOD record number passed back to the application after each direct file READ or WRITE.
18	Relative Record Number (RRN)	A 2-word value for the RRN. The first record is record number 1.
22	Volume Name	A 6-byte EBCDIC volume name, left-justified and padded with blanks.

Figure 11. FILEIO FCA Format for Direct Files

This example shows an FCA for indexed files that would read a record associated with a 4-character key "XXXX".

```
01  FILE-CONTROL-AREA.
    05  REQUEST-TYPE  PIC X(4) VALUE "GETD".
    05  DATA-SET-NAME PIC X(8).
    05  KEY-REL-OP    PIC XX   VALUE "EQ".
    05  KEY-LENGTH    PIC S999 COMP VALUE 4.
    05  KEY-LOCATION  PIC S999 COMP VALUE 0.
    05  FILLER        PIC X(4).
    05  VOLUME-NAME   PIC X(6).
    05  KEY           PIC X(4) VALUE "XXXX".
```

Byte Displacement	Field Contents	Description
0	Request type	A 2-byte EBCDIC request (valid request types are shown in Figure 13).
4	Data Set Name	An 8-byte EBCDIC data set name, left-justified and padded with blanks.
12	Key Relation	A 2-byte EBCDIC key relation operator, either GT, GE, or EQ (required only if request type is GETD, GETS, GTDU, or GTSU).
14	Key Length	A word specifying the length of the key to be used for retrieval. If the length specified is less than the actual key length, the first *n* bytes of the key are used.
16	Reserved	Must be 0.
18	Reserved	Must be 0.
22	Volume Name	A 6-byte EBCDIC volume name, left-justified and padded with blanks.
28	Key Field	The location of the key.[1]

[1] If the key location equals 0 and key length is not equal to 0, the file manager assumes that the key immediately follows the FCA. This is primarily to facilitate COBOL programs, which cannot code addresses.

Figure 12. FILEIO FCA Format for Indexed Files

FILEIO Indexed Access Method Considerations

FILEIO uses the parameters provided to create a parameter list for an Indexed Access Method supervisor call. Therefore, it is important to understand Indexed Access Method operation.

FILEIO executes a file cleanup routine after each call to ACTION, LINK, LINKON, or CYCLE. If any record locks have not been released, the cleanup routine releases these records to prevent any deadlock situations.

This procedure will ensure data integrity on update:

1. Get record.
2. Save record contents.
3. Display to operator.
4. Get with update.
5. Ensure that record contents are unchanged.
6. Put with update.
7. Display to operator.

If sequential processing has been initiated on any indexed files, the FILEIO cleanup routine also releases those files from sequential processing mode. Thus, to continue sequential processing from the same key, the application should save the last key before calling ACTION, CYCLE, LINK, or LINKON. If you want to get sequential records and any of these CALL functions intervene, use GETD with the greater than key relation.

You can scan an indexed file from beginning to end by use of a sequence of "get sequential" (GETS) operations. The first GETS in a sequence should specify a key of all nulls (X'00') and a key relational operator of greater than (C'GT'). When executed, this initial GETS operation will receive the first record in the file (following the record, if any, for which the key is all nulls.) Subsequent GETS will retrieve the records following the first, in sequence.

Direct File Request Types	
READ	Read the record defined by the RRN field of the FCA into the user-provided buffer.
SEOD	Set the system-maintained EOD pointer to the record number provided in the RRN field of the FCA. This number should range from 1 to the EOF record of the file. This request is normally issued after the last record is written to the data set, but you may issue it any time you want to establish a logical end-of-file (EOF).
WRIT	Write the record defined by the RRN field of the FCA into the user-provided buffer.
Indexed File Request Types	
GETD	Get operation, direct read (GET)[1]
GETS	Get operation, sequential read (GETSEQ)
GTDU/GTRU	Direct get, update mode (GET)
GTSU	Sequential get, update mode (GETSEQ)
ICLS	Close an indexed data set (DISCONN)
IDEL	Delete operation (DELETE)
PUTD	Put operation, delete mode (PUTDE)
PUTN	Put operation, new mode, add a record to the file (PUT)
PUTU	Put operation, update mode (PUTUP)
RELR	Release a record held for update (RELEASE)
RELS	Release from sequential processing mode (ENDSEQ)
[1]Indexed file requests call the Indexed Access Method function shown in parentheses. Files are accessed in the PROCESS mode and are shared.	

Figure 13. File Request Types

FILEIO Return Codes

-1 Successful operation.

201 Data set not found.

203 No file table entries are available; all have updates outstanding.

206 Invalid function request type (this is returned for a valid Indexed Access Method function if the Indexed Access Method link module is not linked with the interface program).

207 Invalid key operator.

208 SEOD record number greater than data set length

Other return codes may be returned by the Indexed Access Method or by the system data management support.

CALL FILEIO Direct Access Coding Example

```
WORKING-STORAGE SECTION.
77  RC                  PIC S99 COMP.
77  BUFFER              PIC X(256).
01  FILE-CONTROL-AREA.
    05  REQUEST-TYPE        PIC X(4).
    05  DATA-SET-NAME       PIC X(8).
    05  NUMBER-OF-RECORDS   PIC 999 USAGE COMP VALUE 1.
    05  EOD                 PIC 99999 USAGE IS COMP.
    05  RRN                 PIC 99999 USAGE IS COMP.
    05  VOLUME-NAME         PIC X(6).
     •
     •
     •
    CALL "FILEIO" USING FILE-CONTROL-AREA, BUFFER, RC.
```

CALL FILEIO Indexed Access Coding Example

In this example, change *data set name*, *key length*, and *key location* if you are using secondary keys.

```
WORKING-STORAGE SECTION.
01  FILE-CONTROL-AREA.
    05  REQUEST-TYPE        PIC X(4) VALUE "GETD".
    05  DATA-SET-NAME       PIC X(8) VALUE "CUSTMAST".
    05  RELATIONAL-OP       PIC X(2) VALUE "EQ".
    05  KEY-LENGTH          PIC S999 COMP VALUE 6.
    05  KEY-LOCATION        PIC S999 COMP VALUE 0.
    05  FILLER              PIC X(4).
    05  VOLUME-NAME         PIC X(6) VALUE "EDX003".
    05  KEY-FIELD           PIC X(6) VALUE "069592".
01  BUFFER                  PIC X(256).
01  RC                      PIC S99 COMP.
    •
    •
    •
   CALL "FILEIO" USING FILE-CONTROL-AREA, BUFFER, RC.
```

FTAB—Build Unprotected Field Table

FTAB sets up a table that describes the unprotected action field areas in the input buffer following a CALL ACTION operation. You can use this table to format the output buffer before a CALL ACTION and to position the cursor to a specific field or to a precise location within a field.

FTAB is time-consuming; use it with care. If possible, perform the CALL SETPAN and CALL FTAB operations in the beginning of the application outside the normal looping operation of CALL ACTION and CALL FILEIO.

Note that you must define the FTAB data table in the program's save area to be saved between calls to ACTION, CYCLE, etc.

CALL FTAB Format

[*label*] CALL "FTAB" USING *table, size, code.*

table

The *table* operand is made up of a sequence of 3-word entries. Each 3-word entry describes an unprotected field of the screen image in the input buffer. The first word is the row position; the second word is the column position; and the third word is the length. The sequence begins at the location of the variable named in the table operand; it is repeated for each successive field of the screen.

This is an example of the table format:

TABLE	row	(word 1 of the first field)
	column	(word 2 of the first field)
	length	(word 3 of the first field)
TABLE+6	row	(second field)
	column	
	length	
TABLE+12	row	(third field)
	column	
	length	
•	•	
•	•	
•	•	
n		

where *n* is equal to the value of the *size* operand.

Unused fields in the FTAB table are always set to zero.

size

is 1 word long and contains the number of entries in the table. This decimal value can be in the range 1 to 32767.

code

is the name of a 1-word field reserved for a return code from FTAB.

FTAB Return Codes

-1 Successful return.

1 No data fields found.

2 Data table truncated.

CALL FTAB Coding Example

```
WORKING-STORAGE SECTION.
77   RETURN-CODE     PIC S9999 COMP.
77   FTAB-SIZE       PIC S9999 COMP VALUE 5.
LINKAGE SECTION.
01   INPUT-BUFFER    PIC X(250).
      •
      •
      •
01   PFKEY           PIC S99 COMP.
01   SAVEAREA.
     05   ROW1             PIC S9999 COMP.
     05   COL1             PIC S9999 COMP.
     05   LEN1             PIC S9999 COMP.
     05   ROW2             PIC S9999 COMP.
     05   COL2             PIC S9999 COMP.
     05   LEN2             PIC S9999 COMP.
     05   ROW3             PIC S9999 COMP.
     05   COL3             PIC S9999 COMP.
     05   LEN3             PIC S9999 COMP.
     05   ROW4             PIC S9999 COMP.
     05   COL4             PIC S9999 COMP.
     05   LEN4             PIC S9999 COMP.
     05   COL5             PIC S9999 COMP.
     05   LEN5             PIC S9999 COMP.
     05   ROW5             PIC S9999 COMP.
      •
      •
      •
     CALL "FTAB" USING SAVEAREA, FTAB-SIZE, RETURN-CODE.
```

GETCUR—Get Cursor Position

GETCUR gets the cursor position returned to the program after a CALL ACTION.

CALL GETCUR Format

[*label*] CALL "GETCUR" USING *row, column.*

row
is the label of a word to contain the row number of the cursor. Possible row numbers are 0-23; row 0 is the top line of the screen.

column is the label of a word to contain the column number of the cursor. Possible column numbers are 0-79; column 79 is the rightmost position of a row.

CALL GETCUR Coding Example

```
WORKING-STORAGE SECTION.
77  ROW               PIC S99 COMP.
77  COLUMN            PIC S99 COMP.
     •
     •
     •
CALL "GETCUR" USING ROW, COLUMN.
```

LINK—Transfer Control to Another Program

A call to LINK causes the named application program, which uses the work session controller high-level language subroutines, to be loaded and executed (replacing the current program). If a SETPAN or CHGPAN precedes the LINK, the contents of the input buffer are displayed for 4978, 4979, or 3101 terminals and the buffer is freed. The output buffer is passed unchanged to the linked-to program.

If the transaction identifier is invalid or cannot be found, control returns to the caller; therefore, any return to your program from CALL LINK is an error condition.

CALL LINK Format

[*label*] CALL "LINK" USING *tid*.

tid
is the name of a variable that contains the 4-byte name of a transaction identifier in the $.SYSPD data set. The TID specifies the transaction-processing program to be linked to (right padded with blanks, if necessary).

CALL LINK Coding Example

```
WORKING-STORAGE SECTION.
77  LINK-TO-PGM         PIC X(4) VALUE "MTM4".
•
•
•
CALL "LINK" USING LINK-TO-PGM.
```

LINKON—Transfer Control to Another Program with Output Cycle

A call to LINKON provides the same function as CALL LINK, except that a screen is displayed and the interface program waits for an operator response. The named program is then entered at its entry point with the input buffer containing the unprotected characters from the screen.

If the transaction identifier is invalid or cannot be found, control returns to the caller; therefore, any return to your program from CALL LINK is an error condition.

CALL LINKON Format

[label] CALL "LINKON" USING *tid*.

tid
: is the name of a variable that contains the 4-byte name of a transaction identifier in the $.SYSPD data set. The TID specifies the transaction-processing program to be linked to (right padded with blanks, if necessary).

CALL LINKON Coding Example

```
WORKING-STORAGE SECTION.
77  LINKON-TO-PGM       PIC X(4) VALUE "MTM4".
•
•
•
CALL "LINKON" USING LINKON-TO-PGM.
```

MENU—Return to Primary Menu

CALL MENU immediately terminates the current program and causes the primary menu screen to be displayed. The operator can get back to the primary menu at any time by pressing PF3 on an 4979, 4978, or 3101.

CALL MENU Format

[label] CALL "MENU".

SETCUR—Set Cursor Position

SETCUR specifies the position at which the cursor is to be displayed for the next output cycle. The cursor position is expressed as a pair of row and column coordinates on the screen.

Each screen panel specifies a cursor position to be used while the screen is active (until the next SETPAN or CHGPAN). CALL SETCUR permits you to override the cursor position established by a previous SETPAN or CHGPAN. The cursor is moved on the next output cycle.

CALL SETCUR Format

[label] CALL "SETCUR" USING *row*, *col*.

row
  is the label of a word that contains the number of the row at which the cursor is to be set. Allowable row numbers are 0-23; row 0 is the top line of the screen.

col
  is the label of a word that contains the number of the column at which the cursor is to be set. Allowable column numbers are 0-79; column 79 is the rightmost position of a row.

CALL SETCUR Coding Example

To set the cursor position to row 1, column 12 of a static-screen display:

```
WORKING-STORAGE SECTION.
77  ROW-ONE          PIC S99 COMP VALUE 1.
77  COLUMN-TWELVE    PIC S99 COMP VALUE 12.
•
•
•
CALL "SETCUR" USING ROW-ONE, COLUMN-TWELVE.
```

SETPAN—Write Buffer to Screen

SETPAN causes the specified screen format name to be saved and sets a switch to cause the screen format to be written to the screen during the next output cycle. Any nulls in the screen image will be written unprotected; all other characters will be written protected. The cursor position for the next display after SETPAN will be set to the first unprotected character position. Unprotected defaults that were specified when the screen was built are not displayed by SETPAN.

CALL SETPAN Format

[label] CALL "SETPAN" USING *dsname, code.*

dsname
is the name of a variable that contains the 8-byte data set name of the screen format in the $.WSCIMG data set.

code
is the label of a word in which SETPAN will place a return code.

SETPAN Return Codes

-1 Successful, new panel in buffer.

CALL SETPAN Coding Example

```
WORKING-STORAGE SECTION.
77  S1              PIC X(8) VALUE "SCRNTST1".
77  RC              PIC S99 COMP.
•
•
•
CALL "SETPAN" USING S1, RC.
```

Link Editing Your Programs

You need to link edit your COBOL application program, your EDL stub program, and the interface program. You can choose from two versions of the interface program: O$HLSC if you don't use the indexed access method, and O$HLSCI if you do. If you choose O$HLSCI, your link control data set must include an INCLUDE IAM,ASMLIB statement.

Your link control data set must also include either an INCLUDE O$SEOD,ASMLIB statement (if your program uses the SEOD function) or an INCLUDE O$NOSEOD,ASMLIB statement (if it doesn't). O$SEOD requires approximately 3700 bytes of additional storage in the application program.

Figure 14 shows the link control data set for a COBOL application without the indexed access method.

```
AUTOCALL COKAUTO,ASMLIB          COBOL AUTO LINK MODULE
 INCLUDE  O$BASEC,EDX003         EDL STUB PROGRAM
 INCLUDE  MTMSUB#1,edx003        COBOL APPLICATION SUBROUTINE
 INCLUDE  O$HLSC,ASMLIB          INTERFACE MODULE
 INCLUDE  O$SEOD,ASMLIB          SEOD MODULE
*INCLUDE  O$NOSEOD,ASMLIB        NO SEOD MODULE
 LINK CFMTM4,EDX002 REPLACE END  UPDATE CONTROL STATEMENT
```

Figure 14. Link Control Data Set (No IAM)

Figure 15 shows the link control data set for a COBOL application with the indexed access method.

```
AUTOCALL COKAUTO,ASMLIB          COBOL AUTO LINK MODULE
 INCLUDE  O$BASEC,EDX003         EDL STUB PROGRAM
 INCLUDE  MTMSUB#1,EDX003        COBOL APPLICATION SUBROUTINE
 INCLUDE  O$HLSCI,ASMLIB         INTERFACE MODULE
*INCLUDE  O$SEOD,ASMLIB          SEOD MODULE
 INCLUDE  O$NOSEOD,ASMLIB        NO SEOD MODULE
 INCLUDE  IAM,ASMLIB             IAM STUB
 LINK CFMTM4,EDX002 REPLACE END  UPDATE CONTROL STATEMENT
```

Figure 15. Link Control Data Set (IAM)

Sample Programs

This example consists of a pair of programs that perform a simple file maintenance task. It reads or writes a single record, or sets an end of data (EOD) marker.

The first program displays a screen that requests the file parameters, which include data set name and relative record number. It then issues a CALL LINK to execute the second program, passing the file parameters.

The second program builds a file control area (FCA) from the file parameters and performs the requested file I/O operation. The results of the operation are displayed on the screen, and the program ends.

Note: The examples in this section operate only on data sets of less than 32K bytes.

These sample programs perform the same functions as the sample file maintenance programs in the chapter "Using the Subroutines in an EDL Program." That chapter includes more detailed commentary about what each step of each program does.

COBOL Sample Program 1

```
IDENTIFICATION DIVISION.
PROGRAM-ID.
    MTMSUB.
*
ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
SOURCE-COMPUTER.
    IBM-S1.
OBJECT-COMPUTER.
    IBM-S1.
*
DATA DIVISION.
WORKING-STORAGE SECTION.
77  REQUEST-SCREEN     PIC X(8) VALUE "REQ    ".
77  IO-PROGRAM         PIC X(8) VALUE "PROG2 ".
77  RC                 PIC S99 USAGE IS COMPUTATIONAL.
77  FTAB-SIZE          PIC S9999 COMP VALUE 5.
LINKAGE SECTION.
01  INPUT-BUFFER.
    05  DATA-SET-NAME PIC X(8).
    05  VOLUME-NAME   PIC X(6).
    05  REQUEST-TYPE  PIC X(4).
    05  RELATIVE-RECORD-NUMBER PIC 9999.
    05  BUFFER-DATA   PIC X(80).
01  OUTPUT-BUFFER.
    05  DATA-SET-NAME PIC X(8).
    05  VOLUME-NAME   PIC X(6).
    05  REQUEST-TYPE  PIC X(4).
    05  RELATIVE-RECORD-NUMBER PIC 9999.
    05  BUFFER-DATA   PIC X(80).
    05  EOD-RRN       PIC 9999.
    05  RETURN-CODE   PIC -999.
01  TEB             PIC S99 COMP.
01  IIB             PIC S99 COMP.
01  SAVEAREA.
    05  ROW1        PIC S9999 COMP.
    05  COL1        PIC S9999 COMP.
    05  LEN1        PIC S9999 COMP.
         •
         •
         •
    05  ROW 5       PIC S9999 COMP.
    05  COL 5       PIC S9999 COMP.
    05  LEN 5       PIC S9999 COMP.
*
 PROCEDURE DIVISION
     USING INPUT-BUFFER, OUTPUT-BUFFER, TEB, IIB,
       SAVEAREA.
 BEGIN.
     CALL "FAN".
* BEEP UPON TERMINAL I/O.
     CALL "BEEP".
* RETRIEVE SCREEN IMAGE AND ABORT IF ERROR.
     CALL "SETPAN" USING REQUEST-SCREEN, RC.
```

```
    IF RC IS NOT EQUAL TO -1,
      CALL "MENU".
    IF TEB EQUAL 1,
      CALL "FTAB" USING SAVEAREA, FTAB-SIZE, RC.
    IF RC NOT EQUAL TO -1,
      CALL "MENU".
* DISPLAY SCREEN IMAGE, READ OPERATOR RESPONSE.
    CALL "ACTION".
* MOVE DATA FROM INPUT BUFFER TO OUTPUT BUFFER.
    MOVE CORRESPONDING INPUT-BUFFER TO OUTPUT-BUFFER.
* LINK TO PROGRAM WHICH WILL PERFORM FILE I/O.
    CALL "LINK" USING IO-PROGRAM.
* ABORT IF LINK FAILS.
    CALL "MENU".
```

COBOL Sample Program 2

```
IDENTIFICATION DIVISION.
PROGRAM-ID.
    MTMSUB.
*
ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
SOURCE-COMPUTER.
    IBM-S1.
OBJECT-COMPUTER.
    IBM-S1.
*
DATA DIVISION.
WORKING-STORAGE SECTION.
77  LIST-SCREEN       PIC X(8) VALUE "LST   ".
77  RC                PIC S99 USAGE IS COMP.
77  BUFFER            PIC X(256).
77  FTAB-SIZE         PIC S9999 COMP VALUE 10.
01  FILE-CONTROL-AREA.
    05  REQUEST-TYPE  PIC X(4).
    05  DATA-SET-NAME PIC X(8).
    05  NUMBER-OF-RECORDS PIC S999 USAGE COMP VALUE 1.
    05  EOD-RRN       PIC S99999 USAGE IS COMP.
    05  RELATIVE-RECORD-NUMBER PIC S99999 USAGE COMP.
    05  VOLUME-NAME   PIC X(6).
LINKAGE SECTION.
01  INPUT-BUFFER      PIC X(500).
01  OUTPUT-BUFFER.
    05  DATA-SET-NAME PIC X(8).
    05  VOLUME-NAME   PIC X(6).
    05  REQUEST-TYPE  PIC X(4).
    05  RELATIVE-RECORD-NUMBER PIC 9999.
    05  NUMBER-OF-RECORDS      PIC 9.
    05  BUFFER-DATA   PIC X(80).
    05  EOD-RRN       PIC 9999.
    05  RETURN-CODE   PIC -999.
01  TEB             PIC S99 COMP.
01  IIB             PIC S99 COMP.
01  SAVE AREA.
    05  ROW1          PIC S9999 COMP.
    05  COL1          PIC S9999 COMP.
    05  LEN1          PIC S9999 COMP.
      •
      •
      •
    05  ROW10         PIC S9999 COMP.
    05  COL10         PIC S9999 COMP.
    05  LEN10         PIC S9999 COMP.
*
 PROCEDURE DIVISION
     USING INPUT-BUFFER, OUTPUT-BUFFER, TEB, IIB,
       SAVEAREA.
 BEGIN.
     CALL "FAN".
* SET UP FILE CONTROL AREA.
```

```
     MOVE CORRESPONDING OUTPUT-BUFFER
       TO FILE-CONTROL-AREA.
     MOVE BUFFER-DATA TO BUFFER.
* RETRIEVE LISTING SCREEN AND ABORT IF ERROR.
     CALL "SETPAN" USING LIST-SCREEN, RC.
     IF RC IS NOT EQUAL TO -1.
       CALL "MENU".
     IF TEB EQUAL 1,
       CALL "FTAB" USING SAVEAREA, FTAB-SIZE, RC.
     IF RC NOT EQUAL TO -1,
       CALL "MENU".
* PERFORM FILE I/O.
     CALL "FILEIO" USING FILE-CONTROL-AREA, BUFFER, RC.
* PUT DATA INTO OUTPUT BUFFER SO IT WILL BE DISPLAYED.
     MOVE CORRESPONDING FILE-CONTROL-AREA
       TO OUTPUT-BUFFER.
     MOVE BUFFER TO BUFFER-DATA OF OUTPUT-BUFFER.
     MOVE RC TO RETURN-CODE OF OUTPUT-BUFFER.
* DISPLAY SCREEN IMAGE.
     CALL "ACTION".
* END PROGRAM.
     CALL "MENU".
```

DEBUGGING YOUR PROGRAM

You can use the EDX debug utility ($DEBUG) to interactively debug application programs that use the work session controller high-level language subroutines. This section explains special considerations that you should take into account when using $DEBUG with this type of application. This discussion assumes that you are familiar with the command structure and use of $DEBUG.

$DEBUG Usage Considerations

Programs that use the high-level language subroutines are standard Communications Facility transaction-processing programs and can be debugged using the program dispatcher test and trace techniques.

The PD command:

> PD F TID *xxxx* TE ON

must be used to stop execution of the program so that $DEBUG can be activated. (*xxxx* is the transaction identifier.) Refer to the section of the *Programmer's Guide* that explains tracing and testing transaction-processing programs.

Adjusting $DEBUG Addresses

Addresses used by $DEBUG are relative to the program load point (where the application stub is). Debugging is much easier if addresses are made relative to the beginning of the application code, so that the addresses appearing on the program listing can be used. This is done by use of the $DEBUG QUALIFY command, which adjusts the base address to which all $DEBUG addresses are relative. Add the program load point address (given by $DEBUG or the $A system command) to the offset address of the application code (given by the application link map). Enter their sum as the operand of the QUALIFY command. After this command is entered, $DEBUG can be used in the conventional manner.

For example, suppose that an application named TEST is to be debugged, and that it loaded and stopped by the PD test command. Assume that it is loaded in partition 2. Load $DEBUG into partition 2 and specify:

```
> $CP 2
> $L $DEBUG
  PGM NAME: TEST
TEST IS ALREADY LOADED AT 3800
SHOULD A NEW COPY BE LOADED?
NO
```

The load point of TEST is hexadecimal 3800. From the link map for TEST, you can get the offset to the application code. In the following example, assume that the offset is hexadecimal 570. The QUALIFY command can now be issued, specifying the sum of 3800 and 570.

```
> Q 3D70
```

You can now debug TEST using conventional $DEBUG techniques.

GLOSSARY-INDEX

This Glossary-Index combines a conventional index to this publication with a glossary of technical Communications Facility terms that appear in the book. Only terms unique to the Communications Facility are defined here. For definitions of Event Driven Executive terms, consult the *EDX System Guide;* for definitions of 3270 terms, see the *3270 Component Description* manual.

READER'S COMMENT FORM

SL23-0090-0

IBM Series/1 EDX Communications Facility
Work Session Controller High-Level Language Subroutines
Programmer's Guide

This form may be used to comment on the usefulness and readability of this publication, suggest additions and deletions, and list specific errors and omissions (give page numbers).

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

If you wish a reply, be sure to include your name and address.

COMMENTS

- Thank you for your cooperation. No postage necessary if mailed in the U.S.A.
  FOLD ON TWO LINES, SEAL AND MAIL

Fold and tape

Please do not staple

Fold and tape

No postage necessary if mailed in the U.S.A.

Business Reply Mail

First Class Permit 40 Armonk New York

Postage will be paid by:

International Business Machines Corporation
System Products Division
Dept 24W/037 - PAS5
1501 California Avenue
P. O. Box 10500
Palo Alto, CA 94304

Fold and tape

Please do not staple

Fold and tape

READER'S COMMENT FORM

SL23-0090-0

IBM Series/1 EDX Communications Facility
Work Session Controller High-Level Language Subroutines
Programmer's Guide

This form may be used to comment on the usefulness and readability of this publication, suggest additions and deletions, and list specific errors and omissions (give page numbers).

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

If you wish a reply, be sure to include your name and address.

COMMENTS

- Thank you for your cooperation. No postage necessary if mailed in the U.S.A. FOLD ON TWO LINES, SEAL AND MAIL

Fold and tape　　Please do not staple　　Fold and tape

No postage necessary if mailed in the U.S.A.

Business Reply Mail

First Class　Permit 40　Armonk　New York

Postage will be paid by:

International Business Machines Corporation
System Products Division
Dept 24W/037 - PAS5
1501 California Avenue
P. O. Box 10500
Palo Alto, CA 94304

Fold and tape　　Please do not staple　　Fold and tape

READER'S COMMENT FORM

SL23-0090-0

IBM Series/1 EDX Communications Facility
Work Session Controller High-Level Language Subroutines
Programmer's Guide

This form may be used to comment on the usefulness and readability of this publication, suggest additions and deletions, and list specific errors and omissions (give page numbers).

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

If you wish a reply, be sure to include your name and address.

COMMENTS

- Thank you for your cooperation. No postage necessary if mailed in the U.S.A.
  FOLD ON TWO LINES, SEAL AND MAIL

Fold and tape

Please do not staple

Fold and tape

No postage
necessary
if mailed
in the U.S.A.

Business Reply Mail

First Class Permit 40 Armonk New York

Postage will be paid by:

International Business Machines Corporation
System Products Division
Dept 24W/037 - PAS5
1501 California Avenue
P. O. Box 10500
Palo Alto, CA 94304

Fold and tape

Please do not staple

Fold and tape

READER'S COMMENT FORM

SL23-0090-0

IBM Series/1 EDX Communications Facility
Work Session Controller High-Level Language Subroutines
Programmer's Guide

This form may be used to comment on the usefulness and readability of this publication, suggest additions and deletions, and list specific errors and omissions (give page numbers).

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

If you wish a reply, be sure to include your name and address.

COMMENTS

- Thank you for your cooperation. No postage necessary if mailed in the U.S.A.
  FOLD ON TWO LINES, SEAL AND MAIL

Fold and tape | Please do not staple | Fold and tape

No postage necessary if mailed in the U.S.A.

Business Reply Mail

First Class Permit 40 Armonk New York

Postage will be paid by:

International Business Machines Corporation
System Products Division
Dept 24W/037 - PAS5
1501 California Avenue
P. O. Box 10500
Palo Alto, CA 94304

Fold and tape | Please do not staple | Fold and tape

READER'S COMMENT FORM

SL23-0090-0

IBM Series/1 EDX Communications Facility
Work Session Controller High-Level Language Subroutines
Programmer's Guide

This form may be used to comment on the usefulness and readability of this publication, suggest additions and deletions, and list specific errors and omissions (give page numbers).

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

If you wish a reply, be sure to include your name and address.

COMMENTS

- Thank you for your cooperation. No postage necessary if mailed in the U.S.A.
FOLD ON TWO LINES, SEAL AND MAIL

Fold and tape | Please do not staple | Fold and tape

No postage necessary if mailed in the U.S.A.

Business Reply Mail

First Class Permit 40 Armonk New York

Postage will be paid by:

International Business Machines Corporation
System Products Division
Dept 24W/037 - PAS5
1501 California Avenue
P. O. Box 10500
Palo Alto, CA 94304

Fold and tape | Please do not staple | Fold and tape

IBM

International Business Machines Corporation

General Systems Division
4111 Northside Parkway N.W.
P.O. Box 2150
Atlanta, Georgia 30301
(U.S.A. only)

General Business Group/International
44 South Broadway
White Plains, New York 10601
(International)

SL23-0090-0

Printed in U.S.A.