INGLENET
business solutions

TIP/ix Programming Reference

IP-622

This edition applies to TIP/ix 2.5 and revision levels of TIP/ix 2.5 until otherwise indicated in a new edition. Publications can be requested from the address given below.

TIP Studio 2.5 reserves the right to modify or revise this document without notice. Except where a Software Usage Agreement has been executed, no contractual obligation between Inglenet Business Solutions Inc and the recipient is either expressed or implied.

It is agreed and understood that the information contained herein is **Proprietary** and **Confidential** and that the recipient shall take all necessary precautions to ensure the confidentiality thereof.

If you have a license agreement for TIP Studio or TIP/ix with Inglenet Business Solutions Inc, you may make copies of this documentation for internal use. Otherwise, you may not copy or transmit this document, in whole or in part, in any form or by any means, electronic, mechanical, photocopying, or otherwise, without the prior written permission of Inglenet Business Solutions Inc.

Inglenet Business Solutions Inc

Toll Free:	1-800-387-9391
Website:	http://www.Inglenet.com
Sales:	Sales@Inglenet.com
Help Desk:	HelpDesk@Inglenet.com

TIP Studio, TIP/ix, and TIP/30, and are registered trade marks of Inglenet Business Solutions Inc:

This documentation occasionally makes reference to the products of other corporations. These product names may be trade marks, registered or otherwise, or service marks of these corporations. Where this is the case, they are hereby acknowledged as such by Inglenet Business Solutions Inc.

Contents

Program Control System (PCS) 7
Online Program Structure 7
Program Execution Stack 8
Fixed Order Parameter Passing 12
Linkage Items 13
Transaction End 26
PIB-LOCK-INDICATOR Action 27
PCS Subroutines 28
BATPEER - Peer-to-Peer from Batch 30
BATQUEUE - Queuing from Batch 31
TIPBITS - Convert Bytes to Bits 31
TIPBYTES - Convert Bits to Bytes 34
TIPDATE - Return Date 35
TIPDUMP - Force Program Dump 35
TIPDXC - Delayed Transfer Control 36
TIPJUMP - Direct Transfer Control 37
TIPFLAG - Flag Services 38
TIPFORK - Start Program at a Terminal 41
TIPFORK - Start Background Program 44
TIPFORKW - Start Program in New Window 46
TIPGRPS - Retrieve Elective Groups 47
TIPGRPST - Change Elective Groups 48
TIPMSG - Retrieving Error Messages 50
TIPPEER - Peer-to-Peer Processing 53
TIPQUEUE - Record Queuing 61
TIPRTN - End Online Program 70
TIPSNAP - Snap Dump Memory 71
TIPSUB - Perform Program 72
TIPSUBP - Call a Subprogram 75
TIPTIMER - Timer Services 76
TIPUSR - Where is User 80
TIPUSRID - User Information 81
TIPUSRST – Set new User Information 82
TIPWINAP - Run a DOS or Windows Program 83
TIPXCTL - Transfer Control 84

Message Control System (MCS) 86
Provided Interfaces 86
MCS Screen Formats 87
MCS Subroutines 89

Program Control after CALL 90
MCS Interface Packet 91
MCS Subroutine CALLS 95
TIPASK - Display One Line and Return Answer 95
TIPASKYN - Display One Line and Return Answer 97
TIPERASE - Erase Screen 99
TIPLIST - Pick From a List 100
TIPMENU - Display Menu Bar 107
TIPMSGE - Send Error Text To Screen 108
TIPMSGEO - Define Deferred Error Text 110
TIPMSGI - Read Data from Screen Format 110
TIPMSGO - Output Data to Screen Format 114
TIPMSGOV - Overlay Current Screen 117
TIPMSGPR - Print Current Screen 119
TIPMSGRS - Pop the Current Screen 120
TIPMSGRV - Force Full Screen Transmit 121
TIPTITLE - Display Title 122
FCC Modifications 122
Cursor Positioning 126
Context Sensitive Help 126
Help Text Definition 127
TSTWIN - Sample TIP Program 129
Line Oriented Terminal I/O 136
Function Key Input 137
BREAK - Check For Operator Break 137
PARAM - Parameterize Data 138
PROMPT - Prompt Terminal for Reply 140
PROMPTX8 - Prompt for Text 142
ROLL - Output Line & Roll Screen 143
ROLLPT - Set Terminal Roll Point 144
TEXT - Get One Line From Terminal 145
TEXT80 - Get One Line From Terminal 145
Direct Communications I/O 146
Direct Communications I/O 146
Message Formats 146
TC-DCOUT copybook 148
TIPTERM Functions 149
T-GET - Get Input 150
T-PUT - Output Message 152
Paging API 154
Introduction to Terminal Paging 154
TIPPAGE Paging API 155

Function Calls 157

File Control System (FCS) 162

FCS Overview 162

FCS and Program Access 162
Record Locking 163
Journaling Online Files 163
Before Images 163
After Images 164
Dynamic Files 164
Setting a File in Sequential Mode 164
Record Locking 164
Call TIPFCS - Common Parameters 167
FCS Miscellaneous Functions 170
Techniques for Deleting Records 175

TIPFCS for Indexed Files 176

FCS-ADD - Indexed: Add Record 177
FCS-CLOSE - Indexed: Close File 178
FCS-DELETE - Indexed: Delete Record 178
FCS-ESETL - Indexed: End Sequential Mode 179
FCS-FLUSH - Indexed: Flush File 179
FCS-GET - Indexed: Read by Key 180
FCS-GET - Indexed: Read Sequential Key 181
FCS-GET-INDEX - Indexed: Read for Key 182
FCS-GET-KEYED - Indexed: Read by Key 184
FCS-GET-SEQ-LOCK - Indexed: WORKAROUND 185
FCS-GET-SEQ-NEXT - Indexed: Read Next Record 186
FCS-GET-SEQ-PREV - Indexed: Read Previous Record 187
FCS-GETRN - Indexed: Read by Record Number 188
FCS-GETUP - Indexed: Read With Lock 189
FCS-LOCK - Indexed: Lock Record 191
FCS-NEXT - Indexed: Get Next Record 193
FCS-NOUP - Indexed: Cancel Update 194
FCS-OPEN - Indexed: Open File 195
FCS-PREV - Indexed: Get Previous Record 196
FCS-PUT - Indexed: Rewrite Record 198
FCS-SETL - Indexed: Set Sequential Mode 199
FCS-SETL-BOF - Indexed: Set Sequential Mode 200
FCS-SETL-EOF - Indexed: Set Sequential Mode 201
FCS-SETL-EQ - Indexed: Set Sequential Mode 201
FCS-SETL-GT - Indexed: Set Sequential Mode 203

FCS-SKIP - Indexed: Skip Sequentially 204
TIPFCS for Direct Files 205
FCS-ADD - Direct: Add Record 205
FCS-CLOSE - Direct: Close File 206
FCS-DELETE - Direct: Delete Record 207
FCS-FLUSH - Direct: Flush File 208
FCS-GET - Direct: Read Record 208
FCS-GETUP - Direct: Read With Lock 209
FCS-NOUP - Direct: Cancel Update 210
FCS-OPEN - Direct: Open File 211
FCS-PUT - Direct: Update Record 212
TIPFCS for Sequential Files 213
TIPFCS for Sequential Files 213
FCS-CLOSE - Sequential: Close File 213
FCS-GET - Sequential: Read Record 214
FCS-OPEN - Sequential: Open File 215
FCS-PUT - Sequential: Write A Record 216
TIPFCS for Dynamic Files 217
FCS-ACCESS - Dynamic: Access File 218
FCS-ASSIGN - Dynamic: Assign File 219
FCS-CLOSE - Dynamic: Close File 220
FCS-CREATE - Dynamic Create File 220
FCS-GET - Dynamic: Read Record(s) 221
FCS-OPEN - Dynamic: Open File 223
FCS-PUT - Dynamic: Write Record(s) 224
FCS-SCRATCH - Dynamic: Scratch File 225
TIPFCS for Edit Buffers 226
TIPFCS for Edit Buffers 226
FCS-ADD - Edit: Add/Insert Line 226
FCS-CLOSE - Edit: Close Buffer 227
FCS-DELETE - Edit: Delete Line 228
FCS-FLUSH - Edit: Flush Buffer 229
FCS-GET - Edit: Read Line 229
FCS-OPEN - Edit: Open Buffer 230
FCS-PUT - Edit: Replace Line 232
FCS-SCRATCH - Edit: Scratch Buffer 233
TIPFCS for Library Files 234
Library File Descriptor 235
FCS-CLOSE - Library: Close Element 236
FCS-GET - Library: Read Next Line 236
FCS-NOUP - Library: Close Element (No update) 237

FCS-OPEN - Library: Open Element 238
FCS-PUT - Library: Write Line 239
Transaction Suspend (TIPSUSPEND) 241
TIP Print Facility (TIPPRINT) 242
TIPPRINT Print Destinations 242
FCS-CLOSE - Close TIPPRINT Interface 245
FCS-FLUSH - Flush TIPPRINT Buffer 246
FCS-OPEN - Open TIPPRINT Interface 247
FCS-PUT - Output Print Line 253
Accessing TIP Journal Files 258
Journal and QBL File Record Format 258
Batch Journal File Access 265
FCS Batch Interface 266
Prepare to use batch Interface Routine 267
tipbatpi.o Interface Subroutine 267
Batch Commit and Rollback 268
PCXFER - PC File Transfer 270
File Transfer Interface copybooks 270
PCXFER Masking 274
Transfer from/to MS-DOS File 275
PCXFER Compression 275
FCS-OPEN - Open PCXFER Interface 275
FCS-GET - Input Record from computer 277
FCS-PUT - Output Record to computer 278
FCS-FLUSH - Flush PCXFER Buffer 279
FCS-CLOSE - Close PCXFER Interface 280
Compiling and Testing Application Programs 282
Supported COBOL Compilers 282
Micro Focus COBOL 282
COBOL Makefiles 284
Debugging on-line programs 287
Embedded Debugging Statements 289
With Environment Variables from TIP Command Line: ... 304
Using Micro Focus cobanimsrv 305

Reference Tables 306
Hexadecimal - Decimal Conversion 306
Powers of 2 307
Powers of 16 307
ASCII Code Chart. 308
Standard Windows Character Set 308

National Replacement Character (NRC) Mappings 309
EBCDIC Code Chart 311
EBCDIC NRC Chart 311
Error Codes 312
Unix Shell Error 312
Micro Focus Cobol 312
Return Status from Unix System Calls 313
D-ISAM Error Codes 313
Information Management System(IMS) 314
TIP and IMS Interaction 315
Output for Input Queuing, from IMS Programs 316
IMS Status Codes 317
Known Differences between IMS and TIP 317

Index 318

Program Control System (PCS)

This chapter describes the facilities of the Program Control System (PCS). All TIP facilities that provide program control are included in this classification.

PCS, as a component of TIP, controls the execution of all transaction programs and provides monitor-level functions for transaction programs. Services are provided to support inter-program transfer of control and to permit transaction programs to access timer facilities.

The facilities of PCS are available to transaction programs by issuing standard programming language CALLs to subroutines provided with the TIP system.

When transaction programs are linked, the appropriate subroutine object modules are automatically included. In almost all cases, the subroutines are very small interface routines that transfer control to the resident TIP PCS routines.

When writing online programs, these facilities (especially those allowing transfer of control from one program to another) permit the programmer to use familiar control structures that are taken for granted in batch programs.

All TIP programs, regardless of the manner in which they were actually invoked, return control to the calling program by issuing a call to the subroutine TIPRTN.

This standardized return mechanism means that all TIP programs may operate either as a sub function or as a main function without the need for special code in the program. This powerful feature facilitates the creation of modular application systems.

Online Program Structure

TIP provides an environment for transaction programs. TIP provides several areas of main storage for each transaction program. Some areas are used to communicate information to the TIP system; other areas are used as external work areas by the transaction program.

A transaction program may be servicing a number of users at one time. In order to accomplish this, the program must have separate working areas for each instance of the program.

TIP calls a transaction program exactly as if the program was a subroutine of TIP. The addresses of the fixed areas of storage that are allocated for use by the transaction program are passed as parameters to the transaction program.

Online programs that operate in TIP native mode must be aware of the parameters that are automatically passed by TIP. All transaction programs are called either by TIP (if executed from the command line) or another program (if called via the TIPSUB mechanism for example).

The following discussion illustrates the general structure of a TIP native mode program. For convenience, the examples use COBOL syntax.

Program Execution Stack

Program Stack

TIP transaction programs operate in a stack oriented environment. The standard system prompt is displayed by the TIP command line processor to allow the terminal operator to enter a transaction name and any initial command line parameters that may be required by the transaction. When the program begins execution, it is considered to be executing on stack level one - the initial TIP prompt is regarded as stack level zero.

If the initial program transfers control to another program without an implied return of control (using TIPDXC or TIPXCTL), the called program simply replaces the initial program on the current stack level.

Activation Record

On the other hand, if the initial program transfers control to another program *with* an implied return of control, TIP does the following:

- Suspends execution of the calling program
- Saves the calling program's "activation record" (PIB, CDA, MCS, and WORK-AREA).
- Allocates and initializes (to low values) the called program's activation record
- Copies the calling program's CDA contents into the called program's CDA (for a length of the shorter of the two CDA areas)
- Establishes the PIB, MCS, WORK-AREA for the called program and initializes these areas
- Begins execution of the called program.

The called program is now running at the next higher stack level (level "two" in this case).

Climbing the Stack

This process of "climbing" the stack may proceed up to 16 levels. When any program issues a call to the TIPRTN subroutine, TIP does the following:

- Loads the saved "activation record" of the program that preceded the terminating program on the execution stack

- Copies the contents of the CDA of the terminating program to the CDA of the previous program on the stack (for a length of the shorter of the two CDA areas)
- De-allocates the PIB, MCS, and WORK-AREA of the terminating program
- Resumes execution of the program that invoked the terminating program.

Transferring Data to Another Program

The only information that is passed from stack level to stack level in either direction is the contents of the CDA. Since different programs have different CDA sizes, TIP only copies information between CDA areas for a length of the shorter CDA; therefore, programs can invoke other programs that may represent entire applications as if they were subroutines.

The calling program's environment is restored intact (with the exception of the CDA) whenever the called program (or any descendants of it) terminates back down the stack. A program that is suspended in this manner is not resumed until the stack returns to that point - this may be minutes, hours, or days later!

Record-Oriented Program-to-Program Communications

TIP provides a number of functions that allow one program to exchange data with another. These functions (TIPXCTL, TIPSUB, TIPFORK, etc) are described in the PCS section of the Programming Reference. They are similar in that they pass both control and data from program to program. For example, when program A transfers control to program B by using the TIPXCTL function, program A puts the data into its own CDA (Continuity Data Area). The TIP system copies A's CDA to B's CDA. Thus, when program B begins execution, its CDA will contain a copy of A's CDA.

In addition to the above mentioned functions, TIP also provides two record-passing techniques that also allow programs to exchange data. These two functions are TIPPEER and TIPQUEUE. Both of these functions allow one transaction program to send records to another. Unlike the PCS functions, these functions are record oriented and do not involve the transfer of control to another program.

TIPPEER

provides a real-time link between two transaction programs. The programs may execute within the same TIP system, or execute on different TIP systems that may be on different computer systems. You use the TIPPEER interface the same way you use the TIPFCS interface. Your program OPENs the TIPPEER connection, issues a series of GETs and PUTs to it, and then CLOSEs the connection when it has finished. Your program would use TIPPEER if it needs to exchange information in a real-time or immediate fashion.

However, it may not be possible to establish a connection. For example, the other computer may not be running, the other TIP system may not be up, or the network connection may not be available, etc. The initiating program should take appropriate action in situations when it cannot get a connection. Establishing a TIPPEER connection to another program, is similar to making a phone call.

TIPQUEUE

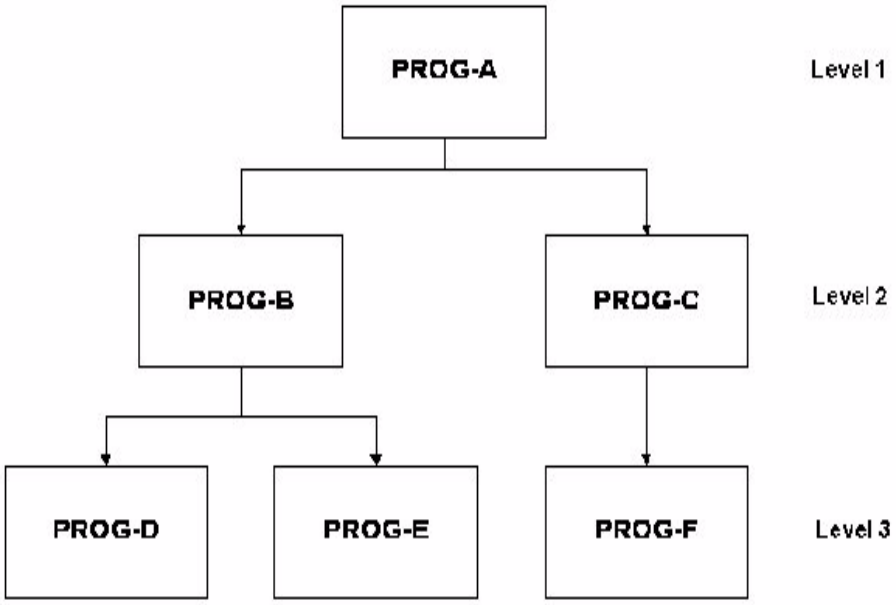
provides a store-and-forward capability, which allows transaction programs to reliably deliver records to other transaction programs. You control the TIPQUEUE interface much like the TIPFCS file interface. That is, a program OPENs a TIPQUEUE file, issues a series of GETs or PUTs to it, and CLOSEs the queue when it has finished. Whereas TIPPEER is a bi-directional (conversational) function, TIPQUEUE is a unidirectional function. That is, a transaction program can write records to a TIP queue, or read records from it, but cannot do both with the same TIP queue. Programs that write to a TIP queue are *client transactions*. Programs that read from the TIP queue are *server transactions*.

TIPQUEUE is transaction oriented. If a program writes to a TIP queue, then issues a commit request, it secures the data in the TIP queue. Likewise, if a program writes to a TIP queue, then subsequently issues a rollback request, all records written to the TIP queue since the last commit point are backed out.

Finally, if TIPPEER is like talking on the telephone, then TIPQUEUE is like leaving a message on a telephone answering computer. The important thing is that the message is heard *eventually*.

The TIP WHOSON Utility

The TIP utility program WHOSON displays the execution stack level of a program. The ability to stack or nest program execution is illustrated by the following hierarchy of programs:



In this example, PROG-A offers a choice of "performing" function B or C. Instead of transferring control (permanently) to either of those programs, PROG-A performs a TIPSUB operation that "performs" (in a sense similar to the COBOL PERFORM verb) the *transaction* B or C. When B or C terminates, control returns to PROG-A immediately following the call to TIPSUB.

PROG-A must ensure that PROG-B or PROG-C (or PROG-D, PROG-E or PROG-F) does not destroy any necessary information in the CDA, although, generally, the CDA is only used for passing information to such subordinate programs and all of the programs involved agree on the layout of the CDA area.

The advantage of this scheme is that PROG-B does not know how it was invoked. PROG-B performs its function and issues a call to TIPRTN. The TIP system determines the return point.

This example must not be interpreted to mean that TIPSUB is preferable to TIPXCTL. The programmer must choose between the two classic techniques to transfer control:
GO TO (TIPXCTL or TIPDXC) or PERFORM (TIPSUB).

Issuing a call to TIPSUB involves TIP system overhead - this overhead is somewhat more than that required for TIPXCTL or TIPDXC.

Coding Suggestions

- Avoid partitioning an application system into modules that are too small. A reasonable rule of thumb is to place code that is related by *use* in one transaction program. For example, use TIPSUB to "PERFORM" infrequently used functions that are not worth permanently imbedding in the load module.
- Avoid writing programs that are either excessively fragmented *or* are monolithic monsters.

- Avoid using a transfer of control to execute a relatively minor task.
- A particularly poor idea is designing a system that uses TIPSUB to "perform" a routine that issues file I/O. In this case, the relatively high overhead involved in a TIPSUB call (which almost always causes the TIP system to perform input/output operations) is incurred *just to perform I/O for the application program*. It is more efficient to perform the I/O directly inline.

Fixed Order Parameter Passing

TIP passes five parameters to a transaction program, in the following **fixed order:**

1. PIB Process Information Block
2. CDA Continuity Data Area order:
3. MCS Message Control System work area
4. WRK Work area
5. GDA Global Data Area

Each of these areas represents main storage, established by TIP, that the transaction program may use.

Example:

```
DATA DIVISION.
LINKAGE SECTION.
01  PIB.          COPY TC-PIB.
01  MCS.          COPY TC-MCS.
01  WORK-AREA.
. . .
01  CDA.          COPY TC-CDA.
01 GDA.
. . .
PROCEDURE DIVISION USING     PIB
                             CDA
                             MCS
                             WORK-AREA
                             GDA
```

The order of appearance of the "01" levels in the LINKAGE SECTION is not important, *but* the order of the areas specified in the PROCEDURE DIVISION USING statement is critical, and fixed.

The names of the "01" level items are not important (although the names illustrated in the example above have become somewhat of a tradition). What **is** very crucial, however, is the rule that each name in the USING list must refer to a corresponding named "01" level in the LINKAGE section.

Linkage Items

The PIB and the CDA must be present and are required. The MCS, WORK-AREA and GDA are optional areas. If an individual program does not use one or more of these areas, using a "dummy" linkage item to maintain the correct USING list order is recommended.

COBOL does not permit the programmer to omit items from the USING clause with one exception: trailing items may be omitted. If a program does not intend to reference the Global Data Area (for example) the fifth parameter may be omitted.

PIB - Process Information Block

The Process Information Block (PIB) is a fixed size and fixed format area that contains information about the transaction that is executing. TIP establishes a PIB area for each execution of a transaction program. Most of the fields in the PIB are read-only in the sense that the transaction program is never required to alter the field. A few fields, however, are occasionally modified by the transaction program as a preliminary step to calling a TIP subroutine.

TC-PIB Copy Book

The layout of the PIB is contained in the supplied COBOL copy book "TC-PIB":

```
*****************************************************
 * TIP - PROCESS INFORMATION BLOCK                   *
*****************************************************
05  PIB-TRID                    PICTURE X(8).
05  PIB-UID                     PICTURE X(8).
05  PIB-TID                     PICTURE X(4).
05  PIB-STATUS                  PICTURE X(1).
    88 PIB-GOOD                    VALUE " ".
    88 PIB-PROG-ABEND              VALUE "A".
    88 PIB-BREAK                   VALUE "B".
    88 PIB-DUP-AFT-NAME            VALUE "C".
    88 PIB-DUP-KEY                 VALUE "D".
    88 PIB-EOF                     VALUE "E".
    88 PIB-IO-ERROR                VALUE "F".
    88 PIB-FUNCTION                VALUE "G".
    88 PIB-ACTIVE                  VALUE "H".
    88 PIB-SECURITY                VALUE "K".
    88 PIB-LOCKED                  VALUE "L".
    88 PIB-MSG-AVAIL               VALUE "M".
    88 PIB-NO-MEM                  VALUE "M".
    88 PIB-NOT-FOUND               VALUE "N".
    88 PIB-OVERFLOW                VALUE "O".
    88 PIB-MISSING-PARAMS          VALUE "P".
    88 PIB-TIMED-OUT               VALUE "T".
    88 PIB-WRONG-MODE              VALUE "W".
    88 PIB-NOT-HELD                VALUE "X".
    88 PIB-HELD                    VALUE "Y".
    88 PIB-FULL                    VALUE "Z".
05  PIB-SYSTEM                  PICTURE X(1).
```

```
    88 PIB-EOJ-PENDING              VALUE "E".
05  PIB-GROUP-1                  PICTURE X(8).
05  PIB-GROUP-2                  PICTURE X(8).
05  PIB-DATE                     PICTURE 9(6) COMP-3.
05  PIB-TIME                     PICTURE 9(6) COMP-3.
05  PIB-JULIAN-DATE.
    10 PIB-YEAR                  PICTURE 9(3) COMP-4.
    10 PIB-DAY-OF-YEAR           PICTURE 9(3) COMP-4.
05  PIB-SITE-NAME                PICTURE X(12).
05  PIB-SECURITY-CODE            PICTURE 9(3) COMP-4.
    88 PIB-TECH-USER                 VALUE 1.
    88 PIB-MASTER-USER               VALUE 1 THRU 9.
    88 PIB-SYSTEM-USER               VALUE 10 THRU 19.
    88 PIB-SYSTEM-OR-HIGHER          VALUE 1 THRU 19.
    88 PIB-PROGRAMMER-USER           VALUE 20 THRU 29.
    88 PIB-PROGRAMMER-OR-HIGHER      VALUE 1 THRU 29.
    88 PIB-APPLICATION-USER          VALUE 30 THRU 255.
    88 PIB-APPLICATION-OR-HIGHER     VALUE 1 THRU 255.
05  PIB-ACCOUNT-NUMBER           PICTURE X(4).
05  PIB-LAST-MCS-NAME            PICTURE X(8).
05  PIB-LOCAP                    PICTURE X(8).
05  PIB-WAIT-TIME                PICTURE S9(4) COMP-4.
05  PIB-DETAIL-STATUS            PICTURE 9(4) COMP-4.
    88 PIB-DUPS-AHEAD                VALUE 1.
    88 PIB-LOAD-MODULE-NOT-FOUND     VALUE 56.
    88 PIB-LOAD-MODULE-SIZE-ZERO     VALUE 57.
    88 PIB-LOAD-MODULE-TOO-LARGE     VALUE 58.
    88 PIB-NO-FREEMEM-TO-LOAD-PROGRAM VALUE 59.
    88 PIB-NO-BACKGROUND-TABLES      VALUE 60.
    88 PIB-ERROR-DURING-PROGRAM-LOAD VALUE 61.
    88 PIB-NOT-FOUND-IN-TIP-CAT      VALUE 62.
    88 PIB-NOT-ALLOWED-BACKGROUND    VALUE 63.
    88 PIB-TERM-LOCAP-NAME-INVALID   VALUE 64.
05  PIB-LOCK-INDICATOR           PICTURE X(1).
    88 PIB-ROLLBACK                  VALUE "O".
    88 PIB-RELEASE                   VALUE "R".
    88 PIB-HOLD                      VALUE "H".
    88 PIB-COMMIT                    VALUE " ".
05  PIB-RPG-UPSI                 PICTURE X(1).
05  PIB-ALT-MCS-ROW              PICTURE 9(3) COMP-4.
05  PIB-CDA-I                    PICTURE 9(8) COMP-4.
05  PIB-WRK-I                    PICTURE 9(8) COMP-4.
05  PIB-LEVEL                    PICTURE 9(3) COMP-4.
05  PIB-TERM-TYPE                PICTURE 9(4) COMP-4.
    88 PIB-UTS-20                    VALUE 2.
    88 PIB-UTS-40                    VALUE 4.
    88 PIB-UTS-60                    VALUE 6.
    88 PIB-TELETYPE                  VALUE 11.
    88 PIB-OFIS-PC                   VALUE 13.
    88 PIB-TIPFE                     VALUE 20.
    88  PIB-TIPWEB                   VALUE 30.
    88  PIB-TIPWEBSERVICE            VALUE 32.
05  PIB-MIRAM-REL-REC-NUM        PICTURE 9(9) COMP-4.
05  PIB-CDA-SIZE                 PICTURE 9(8) COMP-4.
05  PIB-MCS-SIZE                 PICTURE 9(8) COMP-4.
05  PIB-WRK-SIZE                 PICTURE 9(8) COMP-4.
05  PIB-CDA-LENGTH               PICTURE 9(8) COMP-4.
```

```
05  PIB-LANGUAGE                PICTURE X(1).
05  PIB-WDEL-INDICATOR          PICTURE X(1).
    88 PIB-WAIT-DELIVERY            VALUE "Y".
    88 PIB-NO-WAIT-DELIVERY         VALUE "N".
05  PIB-ALT-MCS-COL             PICTURE 9(3) COMP-4.
05  PIB-MAX-MCS-ROW             PICTURE 9(3) COMP-4.
05  PIB-MAX-MCS-COL             PICTURE 9(3) COMP-4.
05  PIB-MCS-FIELD               PICTURE 9(4) COMP-4.
05  PIB-MCS-OVERLAY             PICTURE 9(3) COMP-4.
05  PIB-MCS-KEY                 PICTURE X.
    88 PIB-XMIT                     VALUE " ".
    88 PIB-MSG-WAIT                 VALUE "0".
    88 PIB-FKEY1                    VALUE "1".
    88 PIB-FKEY2                    VALUE "2".
    88 PIB-FKEY3                    VALUE "3".
    88 PIB-FKEY4                    VALUE "4".
    88 PIB-FKEY5                    VALUE "5".
    88 PIB-FKEY6                    VALUE "6".
    88 PIB-FKEY7                    VALUE "7".
    88 PIB-FKEY8                    VALUE "8".
    88 PIB-FKEY9                    VALUE "9".
    88 PIB-FKEY10                   VALUE "A".
    88 PIB-FKEY11                   VALUE "B".
    88 PIB-FKEY12                   VALUE "C".
    88 PIB-FKEY13                   VALUE "D".
    88 PIB-FKEY14                   VALUE "E".
    88 PIB-FKEY15                   VALUE "F".
    88 PIB-FKEY16                   VALUE "G".
    88 PIB-FKEY17                   VALUE "H".
    88 PIB-FKEY18                   VALUE "I".
    88 PIB-FKEY19                   VALUE "J".
    88 PIB-FKEY20                   VALUE "K".
    88 PIB-FKEY21                   VALUE "L".
    88 PIB-FKEY22                   VALUE "M".
    88 PIB-F-REBUILD                VALUE "1" "5" "N".
    88 PIB-F-NEXT                   VALUE "2" "6".
    88 PIB-F-UPDATE                 VALUE "4" "8".
    88 PIB-F-FIELD                  VALUE "<".
    88 PIB-F-MENU                   VALUE ">".
05  PIB-TEST-MODE-INDICATOR     PICTURE X.
    88 PIB-TEST-MODE-ON             VALUE "Y".
    88 PIB-TEST-MODE-OFF            VALUE "N".
05  PIB-HOST-NAME               PICTURE X(12).
05  PIB-TERM-NAME               PICTURE X(8).
05  PIB-CUR-MCS-ROW             PICTURE 9(3) COMP-4.
05  PIB-CUR-MCS-COL             PICTURE 9(3) COMP-4.
05  PIB-SYSTEM-TYPE             PICTURE X.
    88 PIB-TIP30                    VALUE "U".
    88 PIB-TIPIX                    VALUE "X".
05  PIB-K-INTERFACE             PICTURE X(17).
05  PIB-TYPE                    PICTURE X.
    88 PIB-TYPE-PEER                VALUE "P".
    88 PIB-TYPE-QUEUE               VALUE "Q".
    88 PIB-TYPE-SUB                 VALUE "S".
    88 PIB-TYPE-SUBP                VALUE "R".
    88 PIB-TYPE-FORK                VALUE "F".
    88 PIB-TYPE-MSG                 VALUE "M".
```

```
           88 PIB-TYPE-TIP                  VALUE "T".
           88 PIB-TYPE-WEBSERVICE          VALUE "W".
       05  PIB-CENTURY                    PICTURE 99.
       05  FILLER                         PICTURE X.
       05  PIB-FCS-WAIT-TIME              PICTURE S9(4) COMP-4.
    * Long Date format   YYYYMMDD
       05  PIB-LONG-DATE                        PICTURE 9(8).
       05  FILLER REDEFINES PIB-LONG-DATE.
           10  PIB-L-YEAR                       PICTURE 9(4).
           10  FILLER REDEFINES PIB-L-YEAR.
               15  PIB-L-CENTURY                PICTURE 9(2).
               15  PIB-L-YY                     PICTURE 9(2).
           10  PIB-L-MM                         PICTURE 9(2).
           10  PIB-L-DD                         PICTURE 9(2).
       05  PIB-IN-LOCAP                         PICTURE X(8).
       05  PIB-DBI-STS                          PICTURE 9(5).
       05  PIB-ODBC-STS                         PICTURE X(5).
       05  FILLER                               PICTURE X(16).
     * Last 4 bytes holds '$PiB' for overrun detection
       05  FILLER                               PICTURE X(04).
       * Keep FILLER so the whole PIB is 232 bytes.
```

The following is a description of the fields that make up the PIB.:

PIB-TRID
: This eight byte field contains the name of the transaction that is currently executing. The program may interrogate this field to determine the transaction name by which the program was called. Certain TIP subroutine calls (for example: TIPSUB) require the program to move information into this field. The field is reset to the original value after a call to a TIP subroutine that required modification of this field (example: TIPSUB, TIPSUBP).

PIB-UID
: This eight byte field contains one of the following values:

  user id
  : The user id of the user that is executing the program.

  BACK$nnn
  : The executing program is running as a background process. "nnn" is 3 digits representing the assigned background process number.

PIB-TID
: This four byte field is set to the name of the executing terminal. The program may interrogate this field to determine the name of the terminal running the program.

  For background processes, this field contains the terminal name of the originating process (the parent process).

  The value inserted here by TIP is often the last four characters of the user's terminal identifier (for example, the

terminal may be /dev/ttyx18, in which case the PIB-TID field will contain "YX18". This is based on the assumption that the last four characters are more likely to be unique

The environment variable TIPTERM may be set to a particular terminal name if the user wishes to force a specific value.

PIB-STATUS

This one byte field contains the status returned as a result of a call to a TIP subroutine. A number of 88 level items are defined in the copy book for your convenience.

It is *strongly recommended* that programs interrogate this status field after a call to a TIP subroutine. Subroutine calls that work one day may fail miserably the next due to unforeseen external influences.

The TIP Message Control System (MCS) also uses an additional status field in the MCS area (MCS-STATUS). The documentation of the various calls to MCS describes the status that may be set for each of those calls.

A value of PIB-GOOD indicates a successful call to the subroutine as far as TIP is concerned. Any other value may be an error - although it may be only a warning.

PIB-SYSTEM

This one-byte field is set to the value "PIB-EOJ-PENDING" if and only if TIP has been given the shutdown command "EOJ".

This mechanism allows TIP native mode programs to detect EOJ requests. When a program detects this condition, it is good practice to terminate the program as soon as possible to expedite system shutdown procedures.

At the very least, the program should attempt to inform the terminal operator that system shutdown has been requested.

PIB-GROUP-1

This field contains the name of the first elective group to which the user belongs.

If the user is not a member of a user group, this field contains spaces

PIB-GROUP-2

This field contains the name of the second elective group to which the user belongs.

If the user is not a member of a user group, this field contains spaces.

The TIP system permits up to 16 elective groups for each user. Only the first two elective group names are available in the PIB. The names of all elective groups can be obtained by using the subroutine call TIPGRPS.

PIB-DATE
: This field contains the current date (in YYMMDD format - year, month, day sequence).

PIB-TIME
: This field contains the current time of day (in HHMMSS format - hour, minute, second sequence).

  Note: Due to the way TIP operates internally, this field may not be accurate. The best resolution is approximately 1 second (this field is updated by TIP as a side effect of calling some of the TIP routines; between calls to TIP service routines, the contents of this field will not change). Programs that require an accurate time of day (for example to time stamp records or to generate a unique value) should obtain the current time from the operating system; COBOL provides the ACCEPT verb for this purpose.

PIB-JULIAN-DATE
: This group item contains the current date in the Julian format (day of the year, example: 88 109).

PIB-SITE-NAME
: This field contains the site name as retrieved from the Unix uname system call.

PIB-SECURITY-CODE
: This field contains the security level of the user running the program.

  The security level is represented by a number between 1 and 255 (inclusive).

  In the TC-PIB copy book, various popular values are indicated by 88 level items for this field.

PIB-ACCOUNT-NUMBER
: This field contains the account code specified when the user logged on TIP.

PIB-LAST-MCS-NAME
: This field contains the name of the last TIP screen format used at this terminal.

  If the last message output to the terminal was not issued via the TIP Message Control System (MCS) this field contains low-values.

PIB-LOCAP

This field contains the network name of the computer where the program is running.

PIB-WAIT-TIME

This field may be set by a program before soliciting terminal input (via calls to TIPMSGI, PROMPT, or TIPTERM). The system waits for an input message for only the specified wait-time (expressed in seconds).

If an input message does not arrive within the expected time interval the PIB-STATUS for the corresponding input request (TIPMSGI, PROMPT, etc.) is set to PIB-TIMED-OUT.

This field is reset to zero after each input message.

If this field is set to a value greater than zero, the system waits for the specified number of seconds for an input message.

If this field is set to a negative value (the sign is important - not the magnitude of the number), the system waits for the amount of time defined by the TIP definition parameter TIMEOUT= in the tipix.conf file.

If this field contains a zero, the system will not impose a time limit on the arrival of the next input message.

PIB-DETAIL-STATUS

Some TIP subroutines set this field to provide additional information about the status after a call to the subroutine.

The value denoted by the 88-level item "PIB-DUPS-AHEAD" is set by TIPFCS after a record read request (FCS-GET, FCS-GETUP, FCS-NEXT) if there are records with a duplicate key following the record that was read.

Note: MBP ISAM does not provide DUPS AHEAD status information, so it cannot be passed to the application.

PIB-LOCK-INDICATOR

A program sets this field whenever transaction end occurs to indicate to the system the type of record lock handling desired. See the discussion in Transaction End on page 26.

TIP examines this field whenever the program calls TIPRTN, TIPSUB, TIPDXC, TIPFORK, and TIPXCTL. Or calls TIPFCS with a function code of FCS-TREN or solicits terminal input (by calling TIPMSGI, PROMPT, etc.), or calls FCS-CLOSE for a recoverable file. If this field is set to:

Space
The default value. All record locks are released and a TREN (transaction end) record is written to the TIPIX.QBL file.

PIB-ROLLBACK (O)
All updates that were made to files that were defined as "hold for transaction (HOLD=TR)" are rolled back and a TREN (transaction end) record is written to the TIPIX.QBL file.

PIB-RELEASE (R)
All records that are held (via FCS-GETUP) and have not been updated by a corresponding PUT are released. Record locks acquired by updating or adding records are retained.

PIB-HOLD (H)
All record locks are maintained and transaction end is not recognized at this time.

Example:
PROGRAM-A holds a record, moves an "H" to this field, and TIPSUBs to PROGRAM-B. The transaction end that normally would take place when TIPSUB is called is suppressed - PROGRAM-B will find that the record is still held for update.

This field is reset to a space only after it is examined by TIP. The recommended technique is to move the appropriate value to this field before calling a TIP subroutine.

PIB-RPG-UPSI
User programs may use this field to communicate one byte of information from one program stack level to the next level. This field is cleared to low values when a transaction begins. Thereafter, the program(s) control the contents of this field.

The field is named "RPG-UPSI" because TIP RPG programs often use this field.

A program could move a particular value to this field to signal some sort of action to the next program that is called.

PIB-ALT-MCS-ROW
Place a row number (between 1 and 24 inclusive) in this field to override the starting row number for screen formats that are used by the program.

This field is cleared to zero when the transaction begins; thereafter, TIP does not modify this field.

Row numbers placed in this field override the starting row number for screen formats that are subsequently used by the transaction.

May be altered by the application program and define the upper left hand corner of an MCS window.

PIB-CDA-I

CDA area size increment. This field may be set to a value between 0 and 32,767 (inclusive) before transferring control to another program.

The CDA of the called program is increased in size by the specified number of bytes. The increase represents an amount in addition to the CDA= size specified in the called program's definition record.

PIB-WRK-I

WORK-AREA size increment. This field may be set to a value between 0 and 32,767 (inclusive) before transferring control to another program.

The WORK-AREA of the called program is increased in size by the specified number of bytes. The increase represents an amount in addition to the WORK= size specified in the called program's definition record.

PIB-LEVEL

This field contains the current program execution stack level. See the description of the program stack in the previous section – “Program Execution Stack” on page 8.

This value is the same value that is reported by the WHOSON utility program under the heading "Lvl".

PIB-TERM-TYPE

This field is set by the TIP system to identify the type of terminal that is associated with the executing program. A number of COBOL 88-level items are supplied for various terminal types.

PIB-MIRAM-REL-REC-NUM

When the TIP File Control System reads a record from a MIRAM file, this binary full-word is set to the relative record number of that record. The TIPFCS function FCS-GETRN can be used to read an indexed MIRAM file via a specified relative record number. See the description of FCS-GETRN in the documentation for accessing Indexed Files

PIB-CDA-SIZE

The TIP system sets this field to the size of the program's CDA (Continuity Data Area). This value represents the number of bytes in the CDA

PIB-MCS-SIZE
: This field is set by the TIP system to the size of the program's MCS (Message Control System Area). This value represents the number of bytes in the MCS area.

PIB-CDA-LENGTH
: This field may be set by a program to control the number of bytes of data in the CDA that are to be passed to or received from another program. If the program places a value in this field that is greater than the size of the program's CDA, the value is reduced to the size of the CDA.

  A program which is transferring control may place a count in this field to specify the maximum number of bytes to be transferred to the called program and to limit the amount of data that may be returned in the CDA when control returns to this program.

  Data is copied from the calling program CDA to the called program CDA for a length, which is computed as the least of the values in the PIB-CDA-LENGTH field in the PIB for both programs.

  Upon entry to a program, this field contains the same value as the field PIB-CDA-SIZE.

PIB-LANGUAGE
: This field is set to a one character code, which is the assigned language code for the user. The language code is specified in the TIP definition USER record for the user id.

PIB-ALT-MCS-CO
: Defines the column to which a screen format is to be displayed.

  May be altered by the application program and define the upper left hand corner of an MCS window.

PIB-MAX-MCS-ROW
: May be read by the application program and define the bottom right hand corner of an MCS window

PIB-MAX-MCS-COL
: May be read by the application program and define the bottom right hand corner of an MCS window.

PIB-MCS-FIELD
: Returns the relative field number in which the cursor was on the most recent transmit and/or function key.

PIB-MCS-OVERLAY
: Holds the current MCS overlay number. A value of ZERO indicates that nothing is overlaid.

CDA - Continuity Data Area

The Continuity Data Area (CDA) is an area of storage that TIP provides for transaction programs. It is the only area that is copied to and from programs during inter-program linkage - hence the name "continuity". The programmer determines the size and format of this area.

The TIP definition entry for the transaction contains the size (in bytes) of the area.

If a program transfers control to another program, the program initiating the transfer of control can specify the number of bytes in the CDA that are to be transferred to the called program's CDA.

The actual size of the CDA is not limited (other than by the obvious constraint of available memory). All transactions are automatically assigned a minimum CDA area of 256 bytes.

If a transaction program is called from the TIP command line *and* the transaction is defined with CML=YES, the TIP Command Line Processor will place data from the command line into the program's CDA.

TC-CDA copybook

The COBOL copybook TC-CDA defines the format for this particular use of the CDA:

```
*****************************************************
 * TIP - COMMAND LINE FORMAT OF CDA                  *
*****************************************************
05  CDA-PARAMETERS.
    10  CDA-PARAM OCCURS 8 TIMES        PICTURE X(8).
05  CDA-PARAMETERS-9 REDEFINES CDA-PARAMETERS.
    10  CDA-PARAM-9 OCCURS 8 TIMES      PICTURE 9(8).
05  CDA-OPTIONS.
    10  CDA-OPTION OCCURS 8 TIMES       PICTURE X.
05  CDA-OPTIONS-9 REDEFINES CDA-OPTIONS.
    10  CDA-OPTION-9 OCCURS 8 TIMES     PICTURE 9.
05 CDA-TEXT PICTURE X(80).
```

The following is a description of the command line fields that make up the CDA of a TIP program:

CDA-PARAMETERS
: Up to eight positional command line parameters are parameterized into these fields. Strictly numeric parameters (parameters consisting of only digits "0" through "9") are right justified and leading zero filled. Non-numeric parameters are left justified and trailing space filled.

Alphabetic characters in this field are forced to uppercase by the TIP command line processor (TCP).

CDA-OPTIONS
: This field contains the command line option information. Options immediately follow the transaction name and are concatenated with the transaction name by a comma or a slash.

  If no options are supplied, this field contains spaces. Alphabetic characters in this field are forced to uppercase by TIP.

CDA-TEXT
: This field contains the command line *parameters* (not the transaction name or options!) in exactly the format they were entered.

  TIP forces alphabetic characters in the CDA-TEXT area to uppercase.

Additional Considerations:

If the program was *not* called from the TIP command line, the layout and contents of the CDA are entirely at the discretion of the calling program.

MCS - MCS Area

The Message Control System Area (MCS) is an optional area that TIP reserves for the transaction program. The transaction program normally uses this area as a screen format I/O area although it may be used as a work area for any purpose. The size of this area (in bytes) must be correctly specified in the TIP definition for the transaction.

The MCS area is initially set to low values (X'00') by TIP.

TC-MCS copybook

The COBOL copybook TC-MCS defines the layout of the MCS packet prefix that is required to interface with the Message Control System.

The fields in the MCS packet prefix are described in a separate section of this document describing the Message Control System (MCS).

Work-Area

The WORK-AREA is an optional area that TIP reserves for the transaction program. The size and layout of the work area is entirely at the discretion of the programmer. Specify the size of the work area in the TIP definition entry for the transaction program.

The normal practice is for the programmer to simply define any work fields or areas that are needed by the program in this LINKAGE section item.

The COBOL compiler displays a DATA DIVISION MAP, which provides information about all of the fields defined in the program's DATA DIVISION. On the line where the "01" level item is defined, there appears a length (as a number of bytes).

TIP programs use the work area as an area containing fields that are modified during execution. The work area is the proper place for the various record areas for files that are manipulated online.

TIP sets the work area to low values (all X'00') before the transaction program is entered.

GDA - Global Data Area

The Global Data Area (GDA) is an optional area that may be configured when TIP is defined (for more information, see the -G parameter for utility TIPINSTALL in ARP-617, TIP Utilities).

If the GDA is defined in the TIP system, it is an area of fixed (specified) size that can be accessed by all TIP programs that have access permission. (You use the smprog utility to give a program permission to access the GDA.)

The first full-word of the GDA is set to the length of the GDA in bytes. The remainder of the GDA is cleared to low values (X'00') when the TIP system starts.

Common Storage

One possible use of the GDA is to store a common table that is referenced by many online programs. Instead of having each program explicitly read the table into the program's work area, the GDA can be initialized once with the desired data. Thereafter, all programs refer to the table contained in the GDA.

GDA as Serial Resource

The GDA is a serial resource! Modification of this area might involve race conditions. Some convention must be established and followed by programs which intend to update the GDA.

Some techniques that may be used to queue access to the GDA are:

- use of the TIPFLAG subroutine
- locking a record (via a call to TIPFCS using FCS-GETUP) that is designated as a control record for this purpose

TIP installations, that make use of the Global Data Area should consider creating a local copybook that user-written programs can use **to define the layout of the GDA**

Transaction End

When do Transactions Begin and End

Transaction Initiation

In TIP terms a transaction normally begins with the initiation of a program. Since a number of activities take place at transaction end, it is important to establish the conditions that cause TIP to consider that the transaction has terminated.

TIP Transaction Termination

Transaction termination occurs as a result of one of the following events:

6. TIP or the hardware aborts.
7. The transaction program ABORTS and does not contain specific coding to trap such errors.
8. The transaction program calls TIPFCS (the TIP File Control System) with a function code of: FCS-TREN or FCS-CLOSE. However, FCS-CLOSE for edit buffers, library files, dynamic files, and TIPPRINT does not cause transaction termination – as they are non-recoverable files.
9. The transaction program calls TIPRTN (end of program).
10. The transaction program calls TIPSUB, TIPXCTL, TIPDXC or TIPFORK (various transfers of control).
11. The transaction program solicits terminal input (via TIPMSGI, PROMPT, TIPTERM, etc.) without previously specifying that record locks are to be maintained across terminal input.
12. The transaction program calls TIPPEER with a function code of FCS-CLOSE.

In cases (1) and (2), the system always *rolls back* any updates since the last commit point (transaction end) for files that were defined with Record Hold set to T (transaction) and *releases* any outstanding record locks for the transaction.

In cases (3) through (7) the action of the system at transaction end depends on the setting of the PIB-LOCK-INDICATOR (described in the section "PIB-LOCK-INDICATOR ACTION").

The following table summarizes calls that can result in a transaction end.

CALL	CALL Type
TIPFCS	FCS-TREN for transaction end, or FCS-CLOSE for file closing. (However, FCS-CLOSE for edit buffers, library files, dynamic files, and TIPPRINT does not cause transaction end.)
TIPRTN	Program termination

CALL	CALL Type
TIPDXC TIPFORK TIPFORKW TIPSUB TIPSUBP TIPXCTL	Transfer of Control
PROMPT PROMPTX PROMPTX8 TEXT TEXT80 TIPASK TIPASKYN TIPLIST TIPMSGI TIPMSGRV TIPTERM (T-GET)	Terminal Input
PARAM	Potential Terminal Input

In general, transaction end causes the release of record locks and the writing of a "TREN" (mark transaction end) record to the TIP QBL file, if records were updated in a file that is defined as HOLD=TR.

Deferring Transaction End

A program may defer transaction end and link to another program to continue processing (see the description of the PIB field PIB-LOCK-INDICATOR).

Explicit Transaction End

A program may choose to signal an explicit transaction end to occur in those cases where the program must ensure that all updates made thus far are committed. See the description of the call to TIPFCS with the FCS-TREN function.

PIB-LOCK-INDICATOR Action

The following table summarizes the action of the TIP system when it examines the PIB-LOCK-INDICATOR field:

PIB-LOCK-INDICATOR	Transaction End?	GETUP LOCKS	UPDATE LOCKS	ROLLBACK UPDATES?
space / X'00'	Yes	Released	Released	No
O (roll back)	Yes	Released	Released	Yes

PIB-LOCK-INDICATOR	Transaction End?	GETUP LOCKS	UPDATE LOCKS	ROLLBACK UPDATES?
R (release)	No	Released	Kept	No
H (hold)	No	Kept	Kept	No

Event	Action Taken
TRANSACTION END	Marks a new commit point. File updates are either committed or rolled back to the previous commit point.
GETUP LOCK	A record lock that is currently imposed because the program has issued a GETUP on a record but has not yet updated the record.
UPDATE LOCK	A record lock that is currently imposed because the record has been updated by the program and the record is still held if the file is defined with Record Hold set to T (transaction).
ROLLBACK UPDATES	Reverse any file updates since the last commit point (Transaction End) using information in the QBL (quick before look) file(s).

PCS Subroutines

PCS subroutine CALLs are summarized here to provide an overview of the type of facilities that are available through the PCS. The individual subroutines are described in detail in subsequent sections.

Subroutine	Description
BATPEER	Have a peer-to-peer conversation with a transaction (from batch).
BATQUEUE	Send a record to a server transaction (from batch).
TIPBITS	Convert a series of 32 bytes to 32 bits. The TIPBITS subroutine enables COBOL programs to manipulate bit values.
TIPBYTES	Convert a series of 32 bits to 32 bytes. The TIPBYTES subroutine enables COBOL language programs to manipulate bit values.

Subroutine	Description
TIPDATE	Return date in readable format (example: TUESDAY OCTOBER 18 1988). The TIPDATE subroutine returns the date in expanded format (including day of the week).
TIPDUMP	Cause deliberate dump of transaction linkage areas. After the dump, the transaction terminates.
TIPDXC	Transfer control to another transaction program *after* the arrival of an input message from the terminal. TIPDXC enables a program to transfer control to another program after XMIT or a function key is pressed.
TIPFLAG	Provide capability to test and/or set up to 32 "flag" bits (switches). The TIPFLAG subroutine enables transaction programs to manipulate internal TIP flag bits and use these flags as semaphores to implement queuing schemes.
TIPFORK.	Start a transaction program running as an asynchronous process. TIPFORK enables a program to initiate another program as an asynchronous task, thus creating an independently executing process. The independent process runs either: 1) with a terminal, or 2) as a "background" process (without a connected terminal).
TIPFORKW	Open a new sub-window under TIP/fe and start the transaction program running as an asynchronous process in the new sub-window.
TIPGRPS	Retrieve elective group membership. The TIPGRPS subroutine retrieves the names of the application groups to which the user has membership.
TIPGRPST	Set elective group membership. The TIPGRPST subroutine sets the names of the application groups to which the user belongs.
TIPMSG	Retrieve pre-processed error messages from the TIP error message file.
TIPPEER	Have a peer-to-peer conversation with another transaction (like a phone call).

Subroutine	Description
TIPQUEUE	Send a record to a server transaction (like leaving a message on a telephone answering computer).
TIPRTN	Terminate transaction program and return control to calling program. All TIP programs use TIPRTN to terminate and return control to the calling program.
TIPSNAP	"Snap" dump selected portions of program's memory. The TIPSNAP subroutine is used to generate memory-image "snap" dumps of selective portions of a transaction program's memory areas. This subroutine is primarily used for debugging.
TIPSUB	Invoke a transaction program as a sub-function. TIPSUB allows a program to "PERFORM" another program and receive control when that program is finished.
TIPSUBP	Call a subprogram.
TIPTIMER	Delay program execution for a specified number of seconds.
TIPUSR	Retrieve terminal name where a specified user is using TIP system.
TIPUSRID	Retrieve information about TIP user.
TIPWINAP	From TIP/fe, start up a Windows application.
TIPXCTL	"GOTO" another program. Using TIPXCTL, a program can "GO TO" another program without any return of control.

BATPEER - Peer-to-Peer from Batch

BATPEER has not been implemented in TIP Studio. For the functionality found with BATPEER please use the TipAsActiveDTP control to establish PEER sessions with an application server from an external process.

BATPEER is like TIPPEER but it is invoked by a batch client program. It implements synchronous two-way communication between a batch program and a TIP transaction program. The transaction may be executing on the same or a different TIP system.

A *batch client* program uses *BATPEER* for peer-to-peer communication. (Just as an *on-line client* transaction would use *TIPPEER*.)

A *server transaction* always uses *TIPPEER* for peer-to-peer communication. You code the server transaction ***exactly*** the same way as you would with TIPPEER. In fact, you can call the same server transaction from a batch program or an on-line transaction.

For a discussion of peer-to-peer processing, see TIPPEER.

BATQUEUE - Queuing from Batch

BATQUEUE has not been implemented in TIP Studio.

BATQUEUE is like TIPQUEUE but it is invoked by a batch program. It implements queuing between a batch program and a TIP transaction program. The transaction may be executing on the same or a different TIP system.

A *batch* program uses *BATQUEUE* for queuing. (Just as an *on-line* transaction would use *TIPQUEUE*.)

For a discussion of queuing, see TIPQUEUE.

TIPBITS - Convert Bytes to Bits

This subroutine is supplied as a utility for COBOL language programmers that need to manipulate bits. TIPBITS converts a string of 32 bytes (each containing a value of 0 or 1) into a full-word (defined in COBOL as 9(9) BINARY) with the corresponding bits in the full-word set to a zero or one (X'F0' or X'F1').

The bits in the full-word are numbered from 31 to 0 from **LEFT to RIGHT**.

Syntax:

```
CALL "TIPBITS" USING    bit-switches
                        byte-switches
```

bit-switches
: The receiving field defined as a binary full-word - PIC 9(9) COMP SYNC.

byte-switches
: The 32 bytes that are to be mapped into bits in the receiving field. Each byte must contain a zero or one.

Example:

```
MOVE "11001100110011001100110011001100"
```

```
                                    TO BYTE-SWITCHES
     CALL "TIPBITS" USING           BIT-SWITCHES
                                    BYTE-SWITCHES
```

The field "BIT-SWITCHES" would contain:

Binary '11001100110011001100110011001100'

Hex 'CCCCCCCC'

TC-BITS

A supplied copy book named TC-BITS defines the two parameters in the above syntax description. See the description of the TIPFLAG subroutine.

```
*****************************************************
* Define 32 "Bit" Switches                          *
*****************************************************
*
05  BIT-SWITCHES PICTURE 9(9) BINARY SYNC.
*
05  BYTE-SWITCHES.
    10 SWITCH-31                PICTURE 9.
        88 SWITCH-31-OFF            VALUE 0.
        88 SWITCH-31-ON             VALUE 1.
    10 SWITCH-30                PICTURE 9.
        88 SWITCH-30-OFF            VALUE 0.
        88 SWITCH-30-ON             VALUE 1.
    10 SWITCH-29                PICTURE 9.
        88 SWITCH-29-OFF            VALUE 0.
        88 SWITCH-29-ON             VALUE 1.
    10 SWITCH-28                PICTURE 9.
        88 SWITCH-28-OFF            VALUE 0.
        88 SWITCH-28-ON             VALUE 1.
    10 SWITCH-27                PICTURE 9.
        88 SWITCH-27-OFF            VALUE 0.
        88 SWITCH-27-ON             VALUE 1.
    10 SWITCH-26                PICTURE 9.
        88 SWITCH-26-OFF            VALUE 0.
        88 SWITCH-26-ON             VALUE 1.
    10 SWITCH-25                PICTURE 9.
        88 SWITCH-25-OFF            VALUE 0.
        88 SWITCH-25-ON             VALUE 1.
    10 SWITCH-24                PICTURE 9.
        88 SWITCH-24-OFF            VALUE 0.
        88 SWITCH-24-ON             VALUE 1.
    10 SWITCH-23                PICTURE 9.
        88 SWITCH-23-OFF            VALUE 0.
        88 SWITCH-23-ON             VALUE 1.
    10 SWITCH-22                PICTURE 9.
        88 SWITCH-22-OFF            VALUE 0.
        88 SWITCH-22-ON             VALUE 1.
    10 SWITCH-21                PICTURE 9.
        88 SWITCH-21-OFF            VALUE 0.
        88 SWITCH-21-ON             VALUE 1.
    10 SWITCH-20                PICTURE 9.
        88 SWITCH-20-OFF            VALUE 0.
```

```
    88 SWITCH-20-ON          VALUE 1.
10 SWITCH-19                PICTURE 9.
    88 SWITCH-19-OFF         VALUE 0.
    88 SWITCH-19-ON          VALUE 1.
10 SWITCH-18                PICTURE 9.
    88 SWITCH-18-OFF         VALUE 0.
    88 SWITCH-18-ON          VALUE 1.
10 SWITCH-17                PICTURE 9.
    88 SWITCH-17-OFF         VALUE 0.
    88 SWITCH-17-ON          VALUE 1.
10 SWITCH-16                PICTURE 9.
    88 SWITCH-16-OFF         VALUE 0.
    88 SWITCH-16-ON          VALUE 1.
10 SWITCH-15                PICTURE 9.
    88 SWITCH-15-OFF         VALUE 0.
    88 SWITCH-15-ON          VALUE 1.
10 SWITCH-14                PICTURE 9.
    88 SWITCH-14-OFF         VALUE 0.
    88 SWITCH-14-ON          VALUE 1.
10 SWITCH-13                PICTURE 9.
    88 SWITCH-13-OFF         VALUE 0.
    88 SWITCH-13-ON          VALUE 1.
10 SWITCH-12                PICTURE 9.
    88 SWITCH-12-OFF         VALUE 0.
    88 SWITCH-12-ON          VALUE 1.
10 SWITCH-11                PICTURE 9.
    88 SWITCH-11-OFF         VALUE 0.
    88 SWITCH-11-ON          VALUE 1.
10 SWITCH-10                PICTURE 9.
    88 SWITCH-10-OFF         VALUE 0.
    88 SWITCH-10-ON          VALUE 1.
10 SWITCH-09                PICTURE 9.
    88 SWITCH-09-OFF         VALUE 0.
    88 SWITCH-09-ON          VALUE 1.
10 SWITCH-08                PICTURE 9.
    88 SWITCH-08-OFF         VALUE 0.
    88 SWITCH-08-ON          VALUE 1.
10 SWITCH-07                PICTURE 9.
    88 SWITCH-07-OFF         VALUE 0.
    88 SWITCH-07-ON          VALUE 1.
10 SWITCH-06                PICTURE 9.
    88 SWITCH-06-OFF         VALUE 0.
    88 SWITCH-06-ON          VALUE 1.
10 SWITCH-05                PICTURE 9.
    88 SWITCH-05-OFF         VALUE 0.
    88 SWITCH-05-ON          VALUE 1.
10 SWITCH-04                PICTURE 9.
    88 SWITCH-04-OFF         VALUE 0.
    88 SWITCH-04-ON          VALUE 1.
10 SWITCH-03                PICTURE 9.
    88 SWITCH-03-OFF         VALUE 0.
    88 SWITCH-03-ON          VALUE 1.
10 SWITCH-02                PICTURE 9.
    88 SWITCH-02-OFF         VALUE 0.
    88 SWITCH-02-ON          VALUE 1.
10 SWITCH-01                PICTURE 9.
    88 SWITCH-01-OFF         VALUE 0.
```

```
           88 SWITCH-01-ON             VALUE 1.
       10 SWITCH-00                 PICTURE 9.
           88 SWITCH-00-OFF             VALUE 0.
           88 SWITCH-00-ON              VALUE 1.

*******************************************************
* TO COMPRESS BYTE-SWITCHES INTO BIT-SWITCHES FOR    *
*       TIPFLAG.                                      *
*                                                     *
* CALL 'TIPBITS' USING BIT-SWITCHES,                  *
*                      BYTE-SWITCHES.                 *
*                                                     *
*******************************************************
* TO EXPAND BIT-SWITCHES TO BYTE-SWITCHES FOR        *
*       PROGRAM USE.                                  *
*                                                     *
* CALL 'TIPBYTES' USING BIT-SWITCHES,                 *
*                       BYTE-SWITCHES.                *
*                                                     *
*******************************************************
```

TIPBYTES - Convert Bits to Bytes

This subroutine is supplied as a utility for COBOL language programmers that need to manipulate bits. TIPBYTES converts a full-word (defined in COBOL as 9(9) BINARY) into a string of 32 bytes with each byte containing a 0 or 1 (X'F0' or X'F1') depending on the value in the corresponding bit in the full-word.

The bits in a full-word are numbered from 31 to 0 from **LEFT to RIGHT**.

Syntax:

```
CALL "TIPBYTES" USING   bit-switches
                        byte-switches
```

Where:

bit-switches
The full-word field (defined as PIC 9(9) BINARY) that contains the bits t hat are to be converted into a byte representation.

byte- switches
The resulting bytes that are set to a graphic zero or one (X'F0' or X'F1') depending on the setting of the corresponding bits in the field BIT-SWITCHES.

Example:

```
MOVE 118                TO BIT-SWITCHES.
CALL "TIPBYTES" USING   BIT-SWITCHES
                        BYTE-SWITCHES
```

The field "BYTE-SWITCHES" would then contain the following:

PIC X(32) '00000000000000000000000001110110'

A supplied copy book named TC-BITS defines the two parameters in the above syntax description. See the description of the this copy book on page 32.

TIPDATE - Return Date

This routine returns the date in a readable format. An optional parameter may be supplied to convert a date other than today's date.

Syntax:

```
CALL "TIPDATE" USING    date-area
                      [ yymmdd ]
```

Where:

date-area
A 30 character field that receives the date in descriptive language. Example (English) result: "MONDAY APRIL 11 1988 "

yymmdd
Optional parameter allowing the calling program to supply a specific date to be translated into readable format. This field is assumed to be defined as PIC 9(6) with the date in YYMMDD format (example: 891225).

Example:

```
05  TODAYS-DATE              PIC X(30).

    CALL "TIPDATE" USING    TODAYS-DATE
```

TIPDUMP - Force Program Dump

Call this subroutine to force a program dump at a specific point in the processing. This method is simpler than the technique sometimes used by COBOL programmers to force a deliberate program abort - adding garbage to a packed field.

Syntax:

```
CALL "TIPDUMP"
```

There are no parameters.

All LINKAGE-SECTION areas, PIB, CDA, MCS and WORK are printed in Hexadecimal and the program terminates.

The dump is contained in the user's home directory in the file **log.xxxxxxxx** where xxxxxxxx is the transaction name.

TIPDXC - Delayed Transfer Control

Call this subroutine to accomplish a delayed transfer of control to another program. The calling program must specify (in the field PIB-TRID) the transaction name of the program to receive control. The calling program then terminates. The called program receives control *after the next input message is available from the terminal*.

The calling program's CDA data is copied to the CDA of the next program for a length which is the least of:

1.1. the size of the calling program's CDA area
1.2. the size of the called program's CDA area
1.3. the value specified by the calling program in the field PIB-CDA-LENGTH.

Syntax:

```
CALL "TIPDXC"
```

Where:

PIB-CDA-LENGTH
This field may be set to a value representing the maximum number of bytes in the CDA that are to be passed to the CDA of the program to which control is being transferred.

PIB-TRID
Must be set to the transaction name of the program to which control is to be transferred.

Error Conditions:

PIB-STATUS	Meaning
PIB-NOT-FOUND	The program identified by the value in the field PIB-TRID is not defined in the TIP definition, the load module could not be found, or there was insufficient memory to load the program. If you receive bad status and want a more detailed description, use PIB-DETAIL-STATUS. See PIB-DETAIL-STATUS in PIB Process Information Block for more information.
PIB-	The user running the calling program does not have sufficient security to run the requested program or the

PIB-STATUS	Meaning
SECURITY	requested program is locked at this time of day.

Example:

```
MOVE "????????"          TO PIB-TRID
CALL "TIPDXC"
GO TO ERROR-CALLING-TIPDXC
```

Note: The program receiving control will not be scheduled until an input message is available. The calling program must, therefore, avoid the pitfall of issuing the call to TIPDXC without having first issued an output message to permit a subsequent input message to be accepted.

TIPJUMP - Direct Transfer Control

TIPJUMP This call directly transfers control to another program on the same program stack level. The calling program must move the name of the transaction to receive control to the PIB-TRID field and then call TIPJUMP. Only TIP/30 native mode programs may be called using TIPJUMP.

Note: This call is unlike all other subroutine calls that PCS provides to transfer control from program to program because all of the program's work areas (PIB, CDA, MCS, WORK) are directly handed to the program that receives control!

In this special situation, the catalogue entries which pertain to area sizes for the called program are not relevant and are ignored.

The TIPJUMP call can be viewed as a way for a transaction program to continue execution using a different load module.

Syntax:

```
MOVE '????????'      TO PIB-TRID
CALL 'TIPJUMP'
```

Error Conditions:

PIB-STATUS	Meaning
PIB-NOT-FOUND	The program is not catalogued, or the load module could not be loaded, or the field PIB-TID was erroneously modified by the program prior to calling TIPJUMP. If you receive bad status and want a more detailed description, use PIB-DETAIL-STATUS. See PIB-DETAIL-STATUS in "PIB — Process Information Block" .

PIB-STATUS	Meaning
PIB-SECURITY	The user running the initiating program does not have a high enough security to run the requested program or the transaction is locked at this time of day.

TIPFLAG - Flag Services

TIP flag services provides user programs with the ability to manipulate up to 32 binary switches. These switches (flags) are stored as bits of a full-word within TIP and may be accessed by any TIP transaction program or by console operator commands (see the description of operator commands FLAG, ON, and OFF).

The program may set or clear a flag (set to 1 or clear to 0) or may interrogate the current setting of a flag or flags. The flags may be used individually or in combination.

An important feature of this subroutine is the ability for the program to *wait* for one or more of the flags to be in a specific state (either off or on) and then immediately flip the state of the flag or flags. This technique allows a flag or flags to be used as a semaphore to queue access to an event.

The TIPFLAG subroutine requires the programmer to provide a MASK field to identify the subset of the 32 bit flags that are to be manipulated (either set, cleared, or interrogated). This MASK field may have *one or more* bits set on. In most applications, the program is interested in a single one of the flags and, in such cases, only a single bit in the MASK is on.

The bits in a full-word are numbered from 31 to 0 from **LEFT to RIGHT**.

Syntax:

```
CALL "TIPFLAG" USING    function
                        mask
                      [ result ]
```

Where:

function

A character code (0 through 9) representing the function to be performed by TIPFLAG:
In the following descriptions, "set" means the value 1; "clear" means the value 0.

0 Wait for *any* of the flag bits identified in the mask to be set.
1 Wait for *all* of the flag bits identified in the mask to be set.
2 Wait for *any* of the flag bits identified in the mask to be set, then clear the flag bits identified by the mask.
3 Wait for *all* of the flag bits identified in the mask to be set, then clear the flag bits identified by the mask.
4 Wait for *any* of the flag bits identified in the mask to be clear.
5 Wait for *all* of the flag bits identified in the mask to be clear.
6 Wait for *any* of the flag bits identified in the mask to be clear, then set the flag bits identified by the mask.
7 Wait for *all* of the flag bits identified in the mask to be clear, then set the flag bits indicated by the mask.
8 Set the flag bits indicated by the mask.
9 Clear the flag bits indicated by the mask.

mask

A binary full-word that identifies the flags to be acted on by this call to TIPFLAG. Each bit represents a flag. The bits of the full-word are numbered from 31 to 0 from left to right.

result

The field that receives a copy of the flag word after the indicated function is performed.

An easy way to determine whether a flag (or flags) is on or off is to specify function code 8 or 9 with a mask that is all zero (meaning set or clear *no* flags). The result field after the call to TIPFLAG provides a "view" of the current setting of all the flags.

Example:

Assume that a flag bit (say flag 13) is nominated to control access to an auxiliary printer (or some other "resource"). The basic scheme is:

- if flag 13 is set on, the resource is in use and prospective users of that resource must wait for it (this is the same as saying wait for the flag to go to zero!)
- when a program is finished using the resource, the flag must be set to zero (cleared) so that other programs that are queued waiting for the flag can be serviced - one at a time.

The following code illustrates the correct method for a program to "queue" for the resource (by queuing for flag 13 in this case).

```
WORKING-STORAGE SECTION.
```

```
        ...
    COPY TC-FLAG.
        ...
    01 WORKAREA.
        ...
    COPY TC-BITS.
        ...
PROCEDURE DIVISION ...
        ...
8000-QUEUE-FOR-DEVICE.
    MOVE 8192                TO BIT-SWITCHES
*
* 8192 (decimal) = 2 ** 13
* 10 0000 0000 0000 (binary)
*
    CALL "TIPFLAG" USING     WAIT-ALL-CLEAR-SET
                             BIT-SWITCHES
*
* Control will not return til flag 13 is clear
*
* ...do our thing
*
* when we are finished, clear flag 13 so next
* queued program can get control
*
    MOVE 8192                TO BIT-SWITCHES.
    CALL "TIPFLAG"USING      SET-OFF
                             BIT-SWITCHES
```

The program first identifies which of the 32 flags are of interest (MOVE 8192 TO BIT-SWITCHES). The program then calls TIPFLAG with a function code "WAIT-ALL-CLEAR-SET". This has the effect of pausing the program until the specified flag is CLEAR and *immediately setting the flag before returning control to the program*.

The program performs its function and, when finished, clears the flag to allow other potential users to "enter the gate". It is important that all programs which are queuing for flags use this technique to ensure that only one program at a time is able to acquire control of the flag or flags.

Note: In the above example, the choice of flag 13 made it quite feasible to move a number to the full-word and thus obtain the proper bit pattern in the mask. In practice, COBOL makes it very awkward to move 10 digits to a binary full-word elementary item. This is exactly the situation that is addressed by the subroutines TIPBITS and TIPBYTES described earlier in this documentation.

Instead of directly moving a value (say 8192 - representing flag 13) to the mask field, the following technique can always be used:

```
    MOVE ALL 0               TO BYTE-SWITCHES
```

```
MOVE 1                      TO SWITCH-13
CALL "TIPBITS" USING        BIT-SWITCHES,
                            BYTE-SWITCHES
```

TC-FLAG Copy Book:

The COBOL copy book **TC-FLAG** provides a complete set of TIPFLAG function codes. COBOL programs can make use of the subroutines TIPBITS and TIPBYTES to convert bits to bytes or vice versa.

Since this copy book uses COBOL VALUE clauses, it must be placed in the program's WORKING-STORAGE SECTION.

```
*****************************************************
* USED AS FUNCTION CODES TO DIRECT TIP FLAG SERVICES *
*****************************************************
    05  WAIT-ANY-SET           PICTURE X VALUE "0".
    05  WAIT-ALL-SET           PICTURE X VALUE "1".
    05  WAIT-ANY-SET-CLEAR     PICTURE X VALUE "2".
    05  WAIT-ALL-SET-CLEAR     PICTURE X VALUE "3".
    05  WAIT-ANY-CLEAR         PICTURE X VALUE "4".
    05  WAIT-ALL-CLEAR         PICTURE X VALUE "5".
    05  WAIT-ANY-CLEAR-SET     PICTURE X VALUE "6".
    05  WAIT-ALL-CLEAR-SET     PICTURE X VALUE "7".
    05  SET-ON                 PICTURE X VALUE "8".
    05  SET-OFF                PICTURE X VALUE "9".
```

TC-BITS Copy Book

The COBOL copy book **TC-BITS** defines work areas that may be used by the COBOL program that is manipulating TIPFLAGs. This copy book is also used in conjunction with the subroutines TIPBITS and TIPBYTES.

This copy book is normally placed in the program's WORKAREA.

TIPFORK - Start Program at a Terminal

Start a program running on another terminal (TIP session) in the network as an independent, asynchronous process. The program that is started at another terminal runs independently of the initiating program.

Each TIP session has an associated PIB-TID and PIB-TERM-NAME. The values can be set by creating terminal definitions with **smterm** or by using the environment variable TIPTERM. See **smterm** in the *TIP Utilities* manual for details.

The TIP shell must be running at the target terminal. This means that the ability to start a program on the target TIP session depends on the security attributes of the target session and *not* the security attributes of the session issuing the TIPFORK. This is a change from TIP/30.

If another program is currently running at the specified TIP session then the request will be queued (in FIFO order). When the session returns to

the TIP prompt (no programs are active) then the queued requests will be run in sequence (one at a time).

This queuing capability (not provided with TIP/30) eliminates one of the possible error conditions that an application has to deal with.

Before issuing this call, the calling program must:

- move the transaction-id of the program to be started to PIB-TRID
- move the 8-character terminal name to PIB-TERM-NAME or move the 4-character terminal identifier to PIB-TID.

The calling program's CDA data is copied to the CDA of the next program for a length that is the least of:

- the size of the calling program's CDA area
- the size of the called program's CDA area
- the value specified by the calling program in the field PIB-CDA-LENGTH.

Syntax:

```
[ MOVE ?                TO PIB-CDA-LENGTH ]
  MOVE '????????'       TO PIB-TRID
  MOVE '????????'       TO PIB-TERM-NAME
  CALL 'TIPFORK'
```

Where:

PIB-CDA-LENGTH
This field may be set to a value representing the maximum number of bytes in the CDA that are to be passed to the CDA of the program that is being started.

PIB-TRID
Must be set to the transaction name of the program that is to be started

PIB-TID
PIB-TID is a 4-byte field to be compatible with TIP/30.

I f PIB-TERM-NAME contains spaces, LOW-VALUES, or the caller's terminal name, and PIB-TID contains spaces, low-values, or the caller's terminal id, the forked program will run in the background. See TIPFORK - Start Background Program.

PIB-TERM-NAME
PIB-TERM-NAME defaults to the terminal name that you are currently signed on to, and you do not need to alter it during a TIPFORK operation if the PIB-TID field is set correctly to the terminal to which you would like the transaction routed.

The reserved terminal names *BYP and *MST may be moved to the field PIB-TERM-NAME to start a new process running on the bypass terminal or master terminal respectively. The bypass and master terminal can be examined or modified with the smterm utility.

Example: By Name

```
*
* START PRINT PROGRAM ON BYPASS TERMINAL
*
    MOVE 'PRINTPGM'     TO PIB-TRID
    MOVE '*BYP '        TO PIB-TERM-NAME
    CALL 'TIPFORK'
    IF NOT PIB-GOOD PERFORM REPORT-ERROR
```

Example: By TID

```
*
* START PRINT PROGRAM ON BYPASS TERMINAL
*
    MOVE 'PRINTPGM'     TO PIB-TRID
    MOVE '*BYP'         TO PIB-TID
    CALL 'TIPFORK'
    IF NOT PIB-GOOD PERFORM REPORT-ERROR
```

Error Conditions:

PIB-STATUS	Meaning
PIB-NOT-FOUND	There is no TIP session with the PIB-TID (terminal id) or PIB-TERM-NAME.
PIB-SECURITY	The user running the initiating program does not have sufficient security clearance to run the requested program or the requested program is not available because it is locked at this time of day.

Additional considerations:

1.4. If the transaction (specified in PIB-TRID) does not have a security entry in any of the active groups at the targeted TIP session, TIPFORK is successful (it returns PIB-GOOD), but an error message is displayed on the target TIP session:

"Invalid transaction code! xxxxxxxx"

xxxxxxxx is the value of PIB-TRID when TIPFORK was called.

1.5. If the TIP session at the requested terminal does not have security (permission) to run the requested program then TIPFORK is successful (returns PIB-GOOD), but an error message is

displayed on the targeted TIP session:

```
Security prevents use of xxxxxxxx
```

xxxxxxxx is the value of PIB-TRID when TIPFORK was called.

1.6. On return from the call, the fields PIB-TRID and PIB-TID will be restored to the values appropriate for the program that issued the call to TIPFORK.

TIPFORK - Start Background Program

This call starts a specified program running in "background". A background program is a transaction program that is not associated with any terminal - essentially a free-standing program. The background program runs independently of the initiating program.

The calling program's CDA data is copied to the CDA of the next program for a length that is the least of:

- the size of the calling program's CDA area
- the size of the called program's CDA area
- the value specified by the calling program in the field PIB-CDA-LENGTH.

As a background process, the program has access to all TIP functions except those functions that directly solicit input from a terminal.

Background programs are not prohibited from using calls that solicit terminal input; they are, however, not allowed to actually use the terminal for input. A background program *cannot* use input redirection.

A background process is useful for time consuming file processing operations, for which the user does not require a response.

Syntax:

```
[ MOVE ?                    TO PIB-CDA-LENGTH ]
 MOVE "????????"            TO PIB-TRID
 CALL "TIPFORK"
```

Where:

PIB-CDA-LENGTH
This field may be set to a value representing the maximum number of bytes in the CDA that are to be passed to the CDA of the program that is being started in background.

PIB-TRID
The field PIB-TRID must be set to the transaction name of the program that is to be started in background.

Example:

```
*
```

```
* START "USERS" TRANSACTION IN BACKGROUND
*
    MOVE "USERS"            TO PIB-TRID
    CALL "TIPFORK"
    IF NOT PIB-GOOD
        PERFORM REPORT-ERROR
    END-IF
```

Error Conditions:

PIB-STATUS	Meaning
PIB-NOT-FOUND	The program identified in the PIB-TRID is not defined in the TIP definition, or the executable could not be loaded. There are no resources available. If you receive bad status and want a more detailed description, use PIB-DETAIL-STATUS. See **PIB-DETAIL-STATUS** in *PIB — Process Information Block* for more information.
PIB-SECURITY	The user running the initiating program does not have sufficient security clearance to run the requested program or the requested program is not available because it is locked at this time of day.

Additional Considerations:

1.7. The program issuing the call to TIPFORK will not receive control until the child process has started running unless an error is reported.

1.8. The user id of the person running the program is carried forward into the order of search path of the process being started subject to the following condition (TIP does this internally):

BACK$nnn, caller's user id, caller's groups, TIPY

or

If the user *was* defined with:

8.1.1. ¯SEARCH=GROUPS, the search path of the new process becomes: caller's groups, TIPY

8.1.2. ¯SEARCH=NO, the search path of the new process becomes: TIPY

TIPFORKW - Start Program in New Window

TIPFORKW has not been implemented in TIP Studio. For the functionality found with TIPFORKW prior to issuing the call to TIPFORK change the PIB-TID field to the name of the terminal you want the program to run on. This is exactly the way TIP/30 worked.

This call starts a specified program running in a newly created window under TIP/fe. You must be running TIP/fe to use this.

The calling program's CDA data is copied to the called program's CDA for a length that is the smallest of:

- The size of the calling program's CDA area
- The size of the called program's CDA area
- The value specified by the calling program in the field PIB-CDA-LENGTH.

As a new transaction running with TIP/fe, the program has access to all TIP/ix functions..

Syntax:

```
[ MOVE ?                TO PIB-CDA-LENGTH ]
  MOVE "????????"       TO PIB-TRID
  CALL "TIPFORK"
```

Where:

PIB-CDA-LENGTH
This field may be set to a value representing the maximum number of bytes in the CDA that are to be passed to the CDA of the program that is being started.

PIB-TRID
Set this field to the transaction name of the program to be started.

Example:

```
*
* START "CREDRPT" TRANSACTION IN A NEW WINDOW
*
    MOVE "CREDRPT"          TO PIB-TRID
    CALL "TIPFORKW"
    IF NOT PIB-GOOD
        PERFORM REPORT-ERROR
    END-IF
```

Error Conditions:

PIB-STATUS	Meaning
PIB-NOT-FOUND	The program identified in the PIB-TRID is

PIB-STATUS	Meaning
	not defined in the TIP/ix definition, or the executable could not be loaded. There are no resources available. If you receive bad status and want a more detailed description, use PIB-DETAIL-STATUS. See **PIB-DETAIL-STATUS** in *PIB — Process Information Block* on page 20.
PIB-SECURITY	The user running the initiating program does not have sufficient security clearance to run the requested program or the requested program is not available because it is locked at this time of day.

Additional Considerations:

- The program issuing the call to TIPFORKW will not receive control until the child process has started running unless an error is reported.
- The user id of the person running the program is carried forward into the order of search path of the process being started subject to the following condition (TIP/ix does this internally):

  caller's user id, caller's groups, TIPY

  or

  If the user *was* defined with:

  - ¯SEARCH=GROUPS, the search path of the new process becomes: caller's groups, TIPY
  - ¯SEARCH=NO, the search path of the new process becomes: TIPY

TIPGRPS - Retrieve Elective Groups

Use this call to retrieve the elective groups to which the user has access.

Syntax:

```
CALL "TIPGRPS" USING GRPS
```

Where:

GRPS

A data structure that is described by the following copy book (TC-GRPS):

TC-GRPS Copy Book

```
*-----------------   ---------------------------------------*
*                                                           *
* TC-GRPS: FORMAT OF TABLE RETURNED FROM 'TIPGRPS'          *
*                                                           *
* INPUT: MOVE NUMBER-OF-ENTRIES-WANTED TO GRPS-MAX          *
* CALL 'TIPGRPS' USING GRPS.                                *
*                                                           *
* OUTPUT: GRPS-ACTUAL WILL BE THE NUMBER OF ENTRIES RETURNED *
* GRPS-NAME (X) HOLDS THE GROUP NAMES AS THEY               *
* APPEAR IN THE ORDER OF SEARCH                             *
*-----------------------------------------------------------*
 05  GRPS.
     10  GRPS-MAX                    PICTURE 9999 BINARY.
     10  GRPS-ACTUAL                 PICTURE 9999 BINARY.
     10  GRPS-NAMES.
         15  GRPS-NAME               PICTURE X(8)
                                     OCCURS 16 TIMES.
```

Where:

GRPS-MAX
A binary half-word that is set by the calling program to a value between 1 and 16 (inclusive).

The value placed in this field specifies the maximum number of group names that are to be returned. Under most circumstances, the program requests 16 (the maximum).

GRPS-ACTUAL
A binary half-word that is set after the call to the number of group names actually returned by the subroutine.

This value will not exceed the value provided in GRPS-MAX.

GRPS-NAME
An array of group names. Only GRPS-ACTUAL of these will have resultant values. This array corresponds (in one-to-one order) with the elective groups in the user's current order of search..

TIPGRPST - Change Elective Groups

This call alters the elective groups to which the current user has access and, therefore, alters the user's order of search. The alteration is *temporary*; the changes remain in effect for the current session or until the groups are altered again.

The calling program supplies a list of group names that are to be used as the user's elective groups. After a successful call to this subroutine, the user's order of search may be changed.

Syntax:

```
CALL "TIPGRPST" USING   GRPS
```

Where:

GRPS
A data structure that is described by the TC-GRPS copy book. See TIPGRPS on page 47 for a listing and explanation of the filed in the TC-GRPS copybook.:

Error Conditions:

PIB-STATUS	Meaning
PIB-NOT-FOUND	One or more of the suggested group names is not in the user's groupset(s). If you receive bad status and want a more detailed description, use PIB-DETAIL-STATUS. See PIB-DETAIL-STATUS in PIB Process Information Block for more information.

New way of handling groups on TIP/as:

To summarize the differences in assigning groups between TIP/30 or TIP/ix and TIP/as:

- Independent elective groups from TIP/30 and TIP no longer exist. Groups are now only assigned to users via the master group set and logon set.
- Logon set must be a subset of master group set. This is a new requirement on TIP Studio that should improve the implementation and management of group sets. Master group set contains all groups that a user can ever access. Logon set consists of those groups that are active when the user starts a session. TIPGRPST allows a program to modify the active groups to any group in the master group set.

Additional Considerations:

- If the first group name contains an asterisk (*), the TIPGRPST subroutine resets the user's elective groups to the elective groups defined for the user in the TIP definition.
- If a supplied group name is spaces, the corresponding group name in the order of search will be set to spaces (implying "no group here").

Warning: The subroutine will make either all of the requested alterations or none of them. If *any* of the requested groups names is not within the user's groupset, the TIPGRPST subroutine will make no changes!

TIPMSG - Retrieving Error Messages

This function allows your program to retrieve error messages from the TIP error message file. See the **mfm** utility in *TIP Utilities, ARP-617-00* for a description of how to create, change, or delete the error file messages.

The discussion of the **mfm** utility also includes an explanation of edit codes used in supplying optional variable data that is merged with the message text (MSGD-TEXT).

Syntax:

```
CALL "TIPMSG" USING      FCS-GET
                         MSG-PACKET
                         MSG-DATA
                       [ msg-variable ]
```

Where:

FCS-GET
Function code from the TC-FCS copy book.

MSG-PACKET
The TIPMSG interface packet from the TC-MSG copy book.

MSG-DATA
The data record returned by TIPMSG (as outlined in the TC-MSG copy book.)

msg-variable
If you intend to use variable input in your messages you must define a MSG-VARIABLE field in your program. See the discussion of the mfm utility for an explanation of edit codes used in supplying optional variable data that is merged with the message text (MSGD-TEXT).

TC-MSG Copy Book

```
05  MSG-PACKET.
    10  MSG-PACKET-IN.
        15 MSGP-LANGUAGE     PICTURE X.
        15 MSGP-PRODUCT      PICTURE X(8).
        15 MSGP-NUMBER       PICTURE X(6).
    10  MSG-PACKET-OUT.
        15 MSGP-CLASS        PICTURE X.
        15 MSGP-FLAGS        PICTURE X(4).
05  MSG-DATA.
    10  MSGD-LENGTH          PICTURE 9(4)
                              BINARY SYNC.
    10  FILLER               PICTURE X(2).
    10  MSGD-CONTROL         PICTURE X.
    10  MSGD-TEXT            PICTURE X(240).
```

The layout of TC-MSG closely follows the record layout of the error messages in the TIP error message file. The following describes fields in TC-MSG that you may use in your program to supply information to TIPMSG:

MSGP-LANGUAGE
Specify a national language.
Default: **A**merican English.

MSGP-PRODUCT
The product name (for example: TIPIX)

MSGP-NUMBER
Message number (for example: ALL000)

The following describes fields TIPMSG returns to your program:

MSGP-CLASS
The message class as follows:

space	Informational
C	Catastrophic
E	Error
I	Informational
W	Warning
*	Message not found or I/O error on TIP error message file.
>	Product name not defined.
!	Call function illegal (not FCS-GET)
<	Message requires variable data but none supplied.

MSGP-FLAGS
Application dependent flags

MSGD-LENGTH
The length of the message plus five bytes for header information (length and print code.)

MSGD-CONTROL
The print control code

MSGD-TEXT
The message text.

Below is an example of an MFM utility screen that displays some of the data and message text fields that you complete when you add error messages to the TIP error message file:

```
T I P / i x - Message File Maintenance TF$MFM1A
 Function: CH
==========================================
Language: A
 Product: TIPIX___
```

```
Msg-id: 000014 American English
Class: _ (Information, Warning, Error, Catastrophic)
Flags: ____ (Application dependent flags)
Format: ________ (Maintenance screen format name)
Print-code: _____ (HOME,PSPnn,PSKnn,SPnn,SKnn -
default=Print SPace01)
Compress?: N (Y=Multiple spaces are removed from
returned message)
<----------------M-e-s-s-a-g-e---T-e-x-t----------------->
<---:----1----:----2----:----3----:----4----:----5----:----6
Record successfully
updated._____ (60)

_____
_____ (120)

_____
_____ (180)
<----------------C-u-r-r-e-n-t---T-e-x-t----------------->
Record successfully updated.
Date of creation 08/45/84 and last change 93/08/24 < _ >
F1=Refresh F2=Next Record F6=Delete Msg Wait=Cancel
```

Error Conditions:

PIB-STATUS	Meaning
PIB-NOT-FOUND	The record does not exist.

Example of TIPMSG CALL use:

Below is an example of the WORKING-STORAGE-SECTION definitions and procedures of a COBOL program that retrieves TIP error file messages:

```
WORKING-STORAGE-SECTION.

01  ERROR-MESSAGES.          COPY TC-MSG OF TIP.

    05  MSG-VARIABLE         PICTURE X(80).


RETRIEVE-MESSAGE SECTION.

    MOVE "A"                  TO MSGP-LANGUAGE.
    MOVE "TIP30"              TO MSGP-PRODUCT.
    MOVE "ML1001"             TO MSGP-NUMBER.
    MOVE "user id"            TO MSG-VARIABLE.

    CALL "TIPMSG" USING       FCS-GET
                              MSG-PACKET
                              MSG-DATA
```

```
                                        MSG-VARIABLE

           EXIT SECTION.
```

If you intend to use variable input in your messages you must define a MSG-VARIABLE field in your program. See the discussion of the **mfm** utility for an explanation of edit codes used in supplying optional variable data that is merged with the message text (MSGD-TEXT).

TIPPEER - Peer-to-Peer Processing

TIPPEER implements synchronous, two-way communication between two cooperating TIP transaction programs. The transactions may be executing on the same or different TIP systems.

To initiate a peer-to-peer conversation, a program must supply control information by altering the contents of its PIB (Process Information Block). The initiator program sets the PIB-TRID and PIB-LOCAP fields to specify the transaction program it wants.

You invoke TIPPEER functions with COBOL CALL statements. The format is similar to that used by TIPFCS. The first parameter is always a function code. The second parameter is a logical name packet for the peer-to-peer conversation. The third parameter is a record area.

There are only 4 possible functions used with TIPPEER. They are: OPEN, CLOSE, GET and PUT. You can use the standard TIPFCS function codes found in the TC-FCS COPY book (FCS-OPEN, FCS-CLOSE, FCS-GET and FCS-PUT).

The record is in standard variable-length record format. The first binary half-word holds the record length (including the 4-byte header), followed by a two-byte filler, and then the record text.

Normally, the record text should always be valid display data (that is, all ASCII or EBCDIC characters — no binary, signed numeric, or packed decimal fields). The reason for this is that TIPPEER may be maintaining a conversation between two transaction programs that are executing on different computers with dissimilar architectures. As TIPPEER passes records from one system to the other, it translates the contents of each record to the appropriate character set for the computer receiving the data.

However, your application can specify an option to the open function to leave the data “as is." This would be useful, if the data contained some binary or packed information. Since TIPPEER does not know the layout of the data, it must either assume that the entire record is character data (and translate it), or assume that it contains some non-character data (and leave the entire record alone).

Example

```
05  PEER-PKT.
    10 PEER-NAME PICTURE X(8).
    10 PEER-STS PICTURE X.
05  PEER-RECORD.
    10 RECORD-LENGTH PICTURE 9(4) BINARY SYNC.
    10 FILLER PICTURE XX.
    10 PEER-DATA PICTURE X(length of record).

    CALL "TIPPEER" USING    FCS-function,
                            PEER-PKT,
                           [PEER-RECORD]
```

TIPPEER Logical Name Packet

Every call to TIPPEER must specify the logical name packet. The logical packet name contains an application-assigned name for the peer-to-peer conversation.

The application that is requesting the conversation is the *client* application. The client application talks to the *server* application.

An application may initiate (be the client in) one or more conversations. However, server applications may only take the server role in one peer-to-peer session. A server application may in turn initiate other peer-to-peer conversations. In this case, the server application would become the client application for the new conversation(s) it initiates.

The actual name placed in the packet is up to the application. It must not conflict with any other file name that appears in the application's active file table (AFT). When the server application starts, the TIP system creates an AFT entry with the name of **$PRIMARY**. Therefore, the server application must use the logical name **$PRIMARY** when referring to the conversation with its client application.

Record Passing

Within a TIPPEER conversation, both parties must cooperate to produce an orderly conversation. We all know how chaotic it is when both parties of a telephone conversation constantly speak without waiting for the other side to finish. The same is true for a TIPPEER conversation. One side has to be in receive mode when the other side is in transmit mode. After your application says something (by issuing a PUT), it should listen (by issuing a GET).

When the server application is first initiated, it is in send mode. This means that the first operation it should perform is a PUT function to send a record back to the client. This record is typically an acknowledgment record that confirms to the client application that the server has started successfully and is ready to proceed.

The following table illustrates the logic flow in a typical TIPPEER conversation:

Client	Server
Client application fills in the PIB and issues a TIPPEER FCS-OPEN function to establish a peer-to-peer conversation.	
TIP verifies the PIB fields. If the PIB-LOCAP field contains a different value than the current LOCAP, TIP established an link to the identified LOCAP.	
TIP schedules the server program (either locally or remotely based on the definition of the server program or PIB-LOCAP value).	
The client application now receives control back from the OPEN request. The application can check the PIB-STATUS to determine if any error occurred. If the open was successful, the client application would now issue a TIPPEER FCS-GET and wait for a record to arrive from the server.	
	The server program would begin execution and perform its start-up operations. If the server does not want to talk, it must terminate by issuing a TIPRTN function call. Otherwise, the server program (which now has the send token) issues a TIPPEER FCS-PUT function to send a record to its client acknowledging that it has started successfully.
	The server program would then issue a TIPPEER FCS-GET function to wait for a record from its client application.
The client program will receive	

Client	Server
the record from the server, process it and will send a record back to the server.	
The conversation will continue until one side decides to terminate the conversation. *Typically, the client will decide that it no longer requires the server and issues an FCS-CLOSE function on the TIPPEER conversation.* *(If the server wanted to terminate the conversation, it would issue a TIPRTN function call.)*	
The client program issues a TIPPEER FCS-CLOSE to the connection.	
	The server program will receive PIB-EOF status on the GET. At this point, TIP breaks the connection with the client. The server would likely perform its clean up functions (close files, etc.) and issue a TIPRTN function call to terminate.

PIB Fields Used

The TIP Process Information Block (PIB) contains fields that are needed for peer-to-peer conversations. These fields are:

Field	Description
PIB-LOCAP	Optional. This is the name of the TIP system (LOCAP) where the server application will run. A host computer may run several TIP systems.
PIB-TRID	This is the name of the server transaction program to be scheduled. If you do not specify the LOCAP, the PIB-TRID (transaction ID) will determine which LOCAP will initiate the server.
PIB-UID	This is the id of the user who is running the application that is requesting the conversation
PIB-TID	This is the terminal name for the application that is requesting the conversation.

Request Conversation, OPEN

A client application that wants to initiate a conversation must fill in PIB-TRID with the name of the transaction program it wants. It may also fill in the PIB-LOCAP field to make the transaction run on a specific LOCAP. If the client application changes the PIB-LOCAP, TIP initiates the transaction on the specified LOCAP. If the PIB-LOCAP does not change, and the transaction has no alternative LOCAP, then it is initiated on the local TIP system. If a TIPPEER server wants to reject a session as soon as it starts, it should call TIPRTN.

The third parameter on the OPEN request is optional. If specified, it identifies a standard TIP file descriptor packet (TC-FDES). The field FDES-FCS-PERM within this packet is used to indicate whether translation is required. If not specified, the default is to translate the data to the character set of the receiving system. If translation is not wanted, set FDES-FCS-PERM to “N”.

Example

```
03  FILE-DESCRIPTOR.          COPY TC-FDES.
    05  PEER-PKT.
        10  PEER-NAME         PICTURE X(8).
        10  PEER-STS          PICTURE X.

    MOVE "SERVER1" TO         PIB-TRID
    MOVE "CLIENT1" TO         PEER-NAME
    MOVE FCS-PERM-TRANSLATE TO FDES-FCS-PERM
    CALL "TIPPEER" USING      FCS-OPEN
                              PEER-PKT
                              FILE-DESCRIPTOR
    IF NOT PIB-GOOD
  .... conversation did not get started ...
    END-IF
```

Error Conditions

PIB-STATUS	Meaning
PIB-NOT-FOUND	The transaction program or the LOCAP is not available.
PIB-SECURITY	The request did not pass the system security checks.
PIB-EOF	TIP found the transaction, but it (the server application) rejected the conversation.
PIB-WRONG-MODE	A logic error occurred. A client program specified "$PRIMARY" as the file name, or the open statement contained incorrect parameters

PIB-STATUS	Meaning
PIB-DUP-AFT	A conversation or file of this name is already in the AFT.
PIB-NO-MEMORY	The system did not have enough free memory.
PIB-MISSING-PARAMS	Incorrect number of parameters.

Close the Conversation, CLOSE

Only the client application may issue a CLOSE. This will close the conversation. The server program will get a PIB-EOF status on the next TIPPEER call, and the TIP system closes the conversation. Issuing the FCS-CLOSE function will create a commit point for the transaction. This means that updates done by both the client and the server will be committed at this point.

Example:

```
MOVE "CLIENT1"           TO PEER-NAME
CALL "TIPPEER" USING     FCS-CLOSE
                         PEER-PKT
```

Additional Considerations:

- Once the sever application receives control back from the TIPPEER call (likely from an FCS-GET function) with PIB-EOF status, it is no longer participating in the same transaction as its client. From this point forward, the server is running independently from the client and any further updates it performs are not coordinated with its client application.
- The server application may terminate the conversation by doing a CALL "TIPRTN" at any time. However, if the server encounters a condition that would require it to terminate the TIPPEER conversion, it should send a “goodbye” record to the client program indicating that the client should close the connection. The server and the client programs must agree on the format of the goodbye record. The client interprets it as a request by the server to terminate the connection. After sending the goodbye record to the client, the server would issue an FCS-GET request and wait for the client to terminate the connection. The server will recognize that the connection has been broken, because it receives a PIB-EOF status upon completion of the FCS-GET request.

Send a Record, PUT

To send a record to the other side of a conversation, use the FCS-PUT function.

You must specify the record length. For example, if you have 10 bytes of data, you must specify a length of 14. This is because you have to account for the 2-byte length field and the 2-byte filler.

If you want TIP to perform translation for you, the record must not contain any binary or packed decimal data. In other words, all data must be ASCII or EBCDIC characters. TIP will translate the data to the other computer's character set.

If you specified that you do not want translation (when you opened the peer-to-peer conversation), the entire record is left 'as is'.

Example:

```
05  PEER-RECORD.
    10  RECORD-LENGTH        PICTURE 9(4)
                              BINARY SYNC.
    10  FILLER               PICTURE XX.
    10  PEER-DATA            PICTURE X(rec-len).

    MOVE length TO           RECORD-LENGTH
    MOVE data TO             PEER-DATA
    MOVE "CLIENT1" TO        PEER-NAME
    CALL "TIPPEER" USING     FCS-PUT
                             PEER-PKT
                             PEER-RECORD

    IF PIB-GOOD
     ... record sent ok ...
    ELSE
     ... record was not delivered ...
    END-IF
```

Note: After successfully issuing an FCS-PUT to the TIPPEER connection, the application must issue an FCS-GET to wait for a reply, or FCS-CLOSE to end the conversation. In other words, you cannot issue a FCS-PUT right after an FCS-PUT.

Receive Record, GET

When an application wants to receive a record it issues a GET request. The issuing program must set the record length field to the maximum length (including the 4 byte header) that it can accept. After the GET function is complete, assuming no error occurred, the length field will contain the actual length of the record received.

Example:

```
MOVE maxlength TO          RECORD-LENGTH
MOVE "CLIENT1" TO          PEER-NAME
CALL "TIPPEER" USING       FCS-PUT
                           PEER-PKT
```

```
                                PEER-RECORD
CALL "TIPPEER" USING            FCS-GET
                                PEER-PKT
                                PEER-RECORD
EVALUATE TRUE
    WHEN PIB-GOOD
     ... record received from other peer ok ...
    WHEN PIB-NOT-FOUND
    ... The application has attempted to do two GETs,
        a GET must be followed by a PUT or CLOSE ...
    WHEN PIB-EOF
     ... other peer program closed conversation ...
    WHEN PIB-MSG-AVAIL
     ... record available from local terminal ...
    WHEN OTHER
     ... record was not received, some other error ...
END-EVALUATE
```

Primary Peer Conversation for the TIPPEER Server

When an application program is scheduled to service a conversation initiated by a client program, the initial TIPPEER conversation with the client is already established. The server program does not have to issue an FCS-OPEN as TIP has already done this as part of the program initiation function. When TIP creates the TIPPEER connection for the server program to use, it creates it with the name of $PRIMARY. The server must use this name when communicating to its client partner. Since the server did not open the $PRIMARY connection, it should not attempt to close it. TIP will automatically terminate the conversation when the server program issues a TIPRTN.

Once initiated, the server application may establish other TIPPEER conversations if needed. In this case, the server would take on the role of a client in any new conversations that it may initiate.

Example:

```
MOVE maxlength TO               RECORD-LENGTH
MOVE "$PRIMARY" TO              PEER-NAME
CALL "TIPPEER" USING            FCS-GET
                                PEER-PKT
                                PEER-RECORD
EVALUATE TRUE
    WHEN PIB-GOOD
     ... record received from other peer ok ...
     WHEN PIB-NOT-FOUND
     ... The other application passed
     ... "Permission to GET"
     ... This application should only do PUT or CLOSE
     WHEN PIB-EOF
     ... other peer program closed conversation ...
     WHEN PIB-MSG-AVAIL
     ... record available from local terminal ...
     WHEN OTHER
     ... record was not received, some error ...
END-EVALUATE
```

Transaction Processing Using TIPPEER

When a client program OPENs a TIPPEER session, and TIP schedules the server program, the two programs are part of the same transaction. This means that if either program issues a transaction end (TREN) either implicitly (for example, with TIPMSGI) or explicitly (with FCS-TREN), then TIP will secure the updates of both programs. Similarly, if either program issues a rollback request, then both programs will have their respective updates rolled back. This situation continues until one of the programs breaks the TIPPEER connection.

These distributed transaction process capabilities of TIPPEER are in effect without regard for where the client and server actually execute. That is, even if the client program is running on one computer and the server running on another, both programs are part of a single transaction.

TIPQUEUE - Record Queuing

In the previous section, you saw that Peer-to-Peer conversations are bi-directional, connection-oriented dialogues that occur in real-time between two cooperating applications. Just like phone calls.

In this section you will learn about TIPQUEUE. Record queuing is a unidirectional, store-and-forward facility that is not real-time but does provide for guaranteed record delivery, just like leaving a message on a telephone answering computer.

The TIPQUEUE protocol is for applications that require client programs to send records to server programs within a distributed or local TIP environment. TIPQUEUE provides a connectionless protocol that guarantees delivery of records from client programs to server programs.

TIPQUEUE allows transaction programs to queue records to named queues. You use a TIP utility program to define the queues. (In other words, the queues are defined outside your application.) The queue's name is known to the network. Each named queue has a number of properties that identify attributes of the queue, for example:

- the LOCAP where the queue is stored
- the name of the program that services the queue
- the interval at which the queue server program is scheduled.

A server program is a normal TIP transaction. TIP schedules the server when the server's queue has records in it (and it is the right time to schedule the server).

All records written to a queue are guaranteed to be delivered to their respective server. This implies that if the server system is not available then the local system will retain the records until the server system becomes available. The server receives records from a queue in the same order the client wrote them (FIFO).

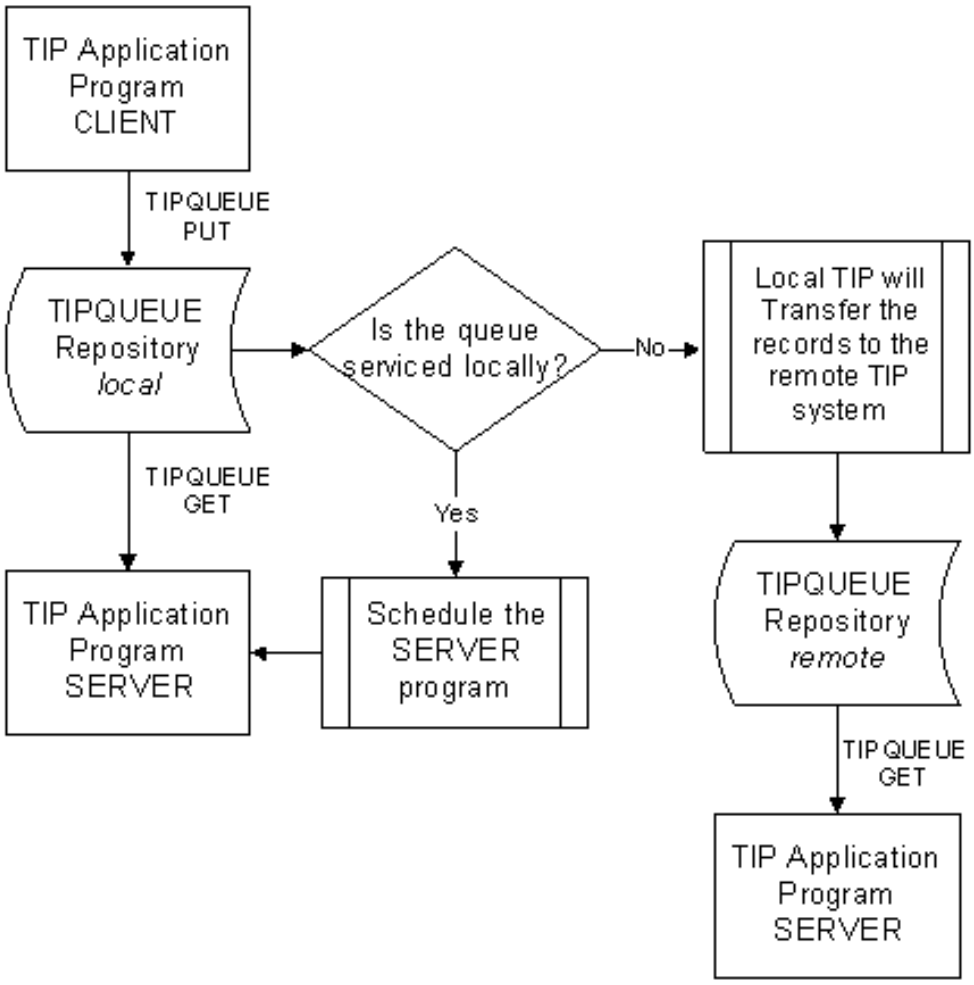
The writing of a record to a queue is part of a transaction and the reading of a record from a queue is also part of a transaction. If the transaction is aborted, or the system stops before a record has been committed to a queue, then the TIP recovery subsystem (that is, rollback) restores the queue to a point of consistency.

Client applications can send records to server applications via a named queue. Each named queue defined in a TIP system is bound to one server application. That is, there is a one-to-one mapping between queue names and server names. The servers are simply transaction programs that are defined in the TIP environment.

Local and Remote Queues

Every TIP system has a local TIPQUEUE repository where records are held until they can be processed. When a queue is defined to be serviced locally, the identified server program is scheduled at the appropriate time to process the queued records. If, however, the queue is defined to be serviced remotely, the queued records must be transferred to the remote TIP system. Once the records have been transferred and stored in the remote TIPQUEUE repository, then the appropriate server will be scheduled, in that TIP system, to process the records.

The following diagram illustrates this record flow:



The lifetime of records queued using TIPQUEUE is as follows:

- The records that a client application writes to a named queue (using FCS-PUT), are committed to the local queue when the application establishes a commit point. Clients can send records to a queue that is defined on the same LOCAP as the clients or on a remote LOCAP (this is transparent to the application program).
- When the client program has reached a commit point (TREN) and it has written records to the queue, it will schedule a server program.
- If the queue is defined to be serviced locally, the server is scheduled. The queue definition determines whether the server is scheduled immediately or on a timed basis.
- If, however, the queue is defined to be serviced on another TIP system, then a special server program is scheduled. This special server is part of the TIP system and is designed to move the queued records from the local TIP system to the remote one. It

does this by establishing a TIPPEER conversation with a partner program on the remote TIP system and transferring the records to it. The partner program writes the records to the remote queue (its local queue). When the partner program comes to a commit point, that TIP system will go through the same process as identified above to schedule a local server program.

- When the server program is scheduled, it OPENs the queue (as an input file) and processes the queued records. When the server application establishes a commit point, the records that it has read from its queue are deleted from the queue.

The administrator must set the delivery interval to be large enough to allow all the queued records to be delivered within the allocated time. Even if the send status of the queue is CLOSED, all the records that have already been queued for a particular queue will be delivered to their destination, subject to the constraints of the delivery time.

Note: Once a server starts, any change in its service status (time interval, time lock, and so on) will not affect its operation. It will only be affected the next time it is scheduled.

TIPQUEUE Service Time Schedule

You use the queue definition utility to specify the service time schedule.

The timelock values you specify when you define the queue control when data is serviced, and when data is transferred between TIP systems. There are two cases when defining a queue: a locally serviced queue, and a remotely serviced queue. The following paragraphs deal with each case.

For a locally served queue, the timelock values determine when (in 24-hour format) the queue is locked. TIP does not schedule the server when the queue is locked. For example, if you want the queue serviced between 5pm and 11pm, specify the timelock as 23:00 to 17:00.

For a remotely serviced queue, the timelock value specifies the portion of the day (in 24-hour format) during which the queue is locked. If a remotely serviced queue is locked, the data is not transferred to the remote TIP system. For example, if you want to transfer a queue over a phone line, and you want to do it from 11pm to 6am when the rates are lower, specify the timelock values as 06:00 to 23:00. At 11pm TIP unlocks the queue, and starts to transfer the records. As the records are transferred to the remote TIP system and written the queue on that system, the server will be scheduled according to the queue definition on that TIP system

You can use the SMQUE utility to mark a queue as CLOSED or HELD. If a queue is CLOSED, the server will still service records at the appropriate times. However, applications cannot write new records to a CLOSED queue. The state of the queue is only checked during the FCS-OPEN function call. If the queue is closed, the TIP program will receive an error when it attempts to open the queue.

If a queue is marked HELD, then no server will be started to process or transfer the records. Client programs can continue to write new records to the queue (assuming it is not closed). The queue will not be serviced until the HELD status is removed.

If the destination LOCAP is down when the delivery time takes effect, the records are saved until the destination LOCAP comes back up again. Then the records are transferred.

TIPQUEUE Interface (API)

You call the API with COBOL 'CALL' statements. The first parameter will always be the function code, the second parameter will be the logical queue name packet, and the third parameter will be a record buffer (only used on GET and PUT functions).

Example:

```
05  QUEUE-RECORD.
    10  RECORD-LENGTH          PICTURE 9(4)
                               BINARY SYNC.
    10  FILLER                 PICTURE XX.
    10  RECORD-DATA            PICTURE X(rec-len).
05  QUEUE-PKT.
    10  QUEUE-NAME             PICTURE X(8).
    10  QUEUE-STS              PICTURE X.
    CALL "TIPQUEUE" USING      FCS-func,
                               QUEUE-PKT,
                               QUEUE-RECORD
```

The queue name that an application uses must already be defined in the TIP catalogue. You can also use the TIP catalogue to direct a logical queue name to the real queue name.

A queue server program that is initiated due to the arrival of a record on the queue will be passed the name of the queue to open in the CDA as parameter 1. The server program must use this name to identify the queue it is to service.

Open the Queue - FCS-OPEN

Before an application can use a queue, it must issue a successful OPEN function for the queue it wants to use. Client applications use the simple (two-parameter) format of the open function.

However, server applications, must specify a third parameter on the FCS-OPEN function. This parameter is a standard TIP file descriptor packet (defined by the TC-FDES copy module) and must have the read-only indicator set (FDES-FCS-PERM set to FCS-PERM-READONLY - this value is declared in the TC-FCS copy module).

Example - Opening a queue as a client:

```
MOVE "QUEUE1" TO               QUEUE-NAME
```

```
CALL "TIPQUEUE" USING        FCS-OPEN
                             QUEUE-PKT
IF NOT PIB-GOOD
 ... queue is not valid ...
END-IF
```

Example - Opening the queue as a server:

```
     03  FILE-DESCRIPTOR.     COPY TC-FDES.

MOVE CDA-PARAM (1) TO         QUEUE-NAME
MOVE FCS-PERM-READONLY TO     FDES-FCS-PERM
CALL "TIPQUEUE" USING         FCS-OPEN
                              QUEUE-PKT
                              FILE-DESCRIPTOR

IF NOT PIB-GOOD
 ... queue is not valid ...
END-IF
```

Error Conditions:

PIB-STATUS	Meaning
PIB-CLOSED	The queue is closed. Records cannot be queued
PIB-NOT-FOUND	The specified queue does not exist in the network
PIB-MISSING-PARAM	Wrong number of arguments passed.

Additional Considerations

- When the TIP system schedules the queue server, it passes the name of the queue in the CDA as parameter 1. The server should use this name when attempting to open the queue. You can manually schedule the queue server from a command line, by entering the transaction code for the server with the queue name as parameter 1.
- transact quename
- This is often convenient for testing or for situations where you want to run the server immediately, but the normal definition of the queue will not automatically schedule the server for some time.

Closing a Queue - FCS-CLOSE

When an application wants to stop processing records or is terminating, it should issue the FCS-CLOSE function to close the queue.

Example:

```
CALL "TIPQUEUE" USING   FCS-CLOSE
```

```
                                    QUEUE-PKT
IF NOT PIB-GOOD
 ... queue is not valid ...
END-IF
```

Error Conditions:

PIB-STATUS	Meaning
PIB-FUNCTION	The program has no associated named queue.
PIB-MISSING-PARAM	Wrong number of arguments passed.
PIB-HELD	A lock cannot be obtained on the control record for the queue file. Currently TIPQUEUE waits up to 30 seconds for a lock, after that it returns PIB-HELD.

Write a Record to a Queue - FCS-PUT

When a client application wants to send a record to a named queue, it will use the FCS-PUT function. Sending a record to a queue is an asynchronous activity, that is, the call returns as soon as the TIPQUEUE system has accepted the record. The record will only be queued for delivery after the client program has committed the record (via FCS-TREN). Thus any records which are put into the TIPQUEUE system since the last commit point can be rolled back.

You must specify the record length (4 or greater). The record itself should not contain any binary or packed decimal data. All data must be valid ASCII or EBCDIC characters.

The queue name may be a logical name that is further defined in the TIP catalog.

Example:

```
05  QUEUE-RECORD.
    10  RECORD-LENGTH        PICTURE 9(4)
                              BINARY SYNC.
    10  FILLER                PICTURE XX.
    10  RECORD-DATA           PICTURE X(rec-len).
05  QUEUE-PKT.
    10  QUEUE-NAME PIC X(8).
    10  QUEUE-STS PIC X.
 ...
    MOVE LENGTH TO            RECORD-LENGTH
    MOVE DATA TO              RECORD-DATA
    MOVE "WRKQUE" TO          QUEUE-NAME

    CALL "TIPQUEUE" USING     FCS-PUT
                              QUEUE-PKT
```

```
                                        QUEUE-RECORD
IF PIB-GOOD
 ... record queued OK ...
ELSE
 ... record was not queued ...
END-IF
```

Error Conditions

PIB-STATUS	Meaning
PIB-MISSING-PARAM	Wrong number of arguments passed
PIB-EOF	You cannot send records of 0-length
PIB-OVERFLOW	You cannot send records of more than 32,767 bytes in length.
PIB-FUNCTION	A system error occurred.
PIB-HELD	A lock cannot be obtained on the control record for the queue file. Currently TIPQUEUE waits up to 30 seconds for a lock, after that it returns PIB-HELD.

Get a Record from the Queue - FCS-GET

The server application may receive records from its corresponding named queue using FCS-GET. If no records are available on the queue, the program will receive PIB-EOF status.

When receiving records, the server application must specify the maximum record size and have a work area that is large enough to receive the record. If the record received is larger than the maximum size specified by the server, then TIP does not retrieve the record from the queue, but sets the PIB-STATUS to PIB-OVERFLOW.

The reception of a record is part of a transaction. TIP removes the record from the queue when the server establishes a commit point. To push any records received since the last commit point back into the queue, do a rollback.

Example:

```
MOVE maxlength TO                   RECORD-LENGTH
CALL "TIPQUEUE" USING               FCS-GET
                                    QUEUE-PKT
                                    QUEUE-RECORD

EVALUATE TRUE
    WHEN PIB-GOOD
     ... record received from client OK ...
    WHEN PIB-EOF
     ... no records available ...
    WHEN other
```

```
        ... record was not received, - check pib status
END-EVALUATE
```

Upon return**,** *record-length* will contain the actual size of the record received.

Error Conditions

PIB-STATUS	Meaning
PIB-OVERFLOW	Record received is larger that the user's record area.
PIB-MISSING-PARAM	Wrong number of arguments passed.
PIB-EOF	You cannot receive records of 0-length.
PIB-FUNCTION	A system error occurred.
PIB-HELD	A lock cannot be obtained on the control record for the queue file. Currently TIPQUEUE waits up to 30 seconds for a lock, after that it returns PIB-HELD.

Developing Client-Server Applications

Once a client starts writing records to a queue, or a server starts reading records from a queue, TIP maintains record locks on the queue file until the client reaches a commit point. (See Transaction end.)

When designing an application that uses TIPQUEUE, you should try to minimize the interval over which records are locked. This is especially true if several users run a program that writes to a given queue or if several different programs write data to the same queue.

Server programs usually want to run until they have processed all the records in the queue that they are servicing. Records that have been read from the queue are not removed from the queue until transaction end time. Therefore, as a server program is reading the queue, it should create commit points (by issuing FCS-TRENs) at the appropriate time. Otherwise, all the records read from the queue, and any other records updated in files, remain locked until the transaction comes to a commit point.

When an application issues a call to TIPQUEUE with a function code of FCS-PUT, FCS-GET, or FCS-CLOSE, a lock is requested on the control record for the queue file. This lock is not relinquished until a commit point has been reached in the transaction. This is necessary to allow queue updates to be coordinated with updates on data files and to allow for the possibility of rollback.

If a lock cannot be obtained on the control record for the queue file, these functions (FCS-PUT, FCS-GET, and FCS-CLOSE) fail. Currently

TIPQUEUE waits up to 30 seconds for a lock on the control record and after that time it returns PIB-HELD.

To minimize contention for the control record:

- Queues can be assigned to separate queue files. This is recommended for heavily used queues. By default queues are assigned to the file TIP$QUE.
- Keep transaction intervals to a minimum. Issuing an FCS-PUT to a queue and then holding record locks while waiting for screen input (by moving "H" to PIB-LOCK-INDICATOR) is not recommended. Other transactions are unable to write to the queue file while the record lock on the queue control record is outstanding.

The best application strategy is to perform all necessary screen interaction then issue file and queue updates.

A server program can also act as a *client* by not only receiving records, but also sending records to named queues. That is, a server can receive records, perform some preliminary processing on them, and then send them to another queue for more processing by another server.

TIPRTN - End Online Program

This call terminates an online TIP program.

If the terminating program was running in background, TIP simply de-allocates all of the areas of memory that were assigned to the program and marks the background process table available. Background programs, by definition, have no program to return to.

If the terminating program was running in foreground (at a terminal) control returns to the program that called the terminating program.

If the terminating program was executed from the TIP command line, control returns to the TIP Command Line Processor.

Syntax:

```
[ MOVE ? TO PIB-CDA-LENGTH ]
  CALL "TIPRTN"
```

Where:

PIB-CDA-LENGTH

The program may move a value to this field to control the number of bytes of data in the CDA that can potentially be copied to the CDA of the program that is next to receive control.
The number of bytes that are copied to the next program's CDA is computed as the least of the values in the field PIB-CDA-LENGTH in the PIB of each of the two programs

involved.
For example, a program that has used the CDA more or less as a work area and does not wish to return any data to the calling program can move zero to PIB-CDA-LENGTH. In that case, the calling program's CDA will remain intact.

Error Conditions:

There is no return of control after a call to TIPRTN.

Additional Considerations:

- The contents of the CDA are copied back to the calling program (unless the terminating program is running in background).
- The terminating program may place a value in the field PIB-RPG-UPSI to return information to the calling program. This facility is primarily intended to be used in situations where some sort of exceptional status is to be returned to the calling program (and requires the two programs to agree on some sort of convention governing the contents of that field).

TIPSNAP - Snap Dump Memory

This subroutine allows a program to produce "snap" dumps of various sections of memory. The specified locations of memory are displayed in a report that is output to a file named "**log.xxxxxxxx**" where "xxxxxxxx" is replaced by the name of the transaction that invoked TIPSNAP.

Syntax:

```
CALL "TIPSNAP" USING    bgn-1 end-1
                      [ bgn-2 end-2 ]
                      [ bgn-3 end-3 ]
                      [ bgn-4 end-4 ]
```

Up to four pairs of parameters may be passed; each pair represents the starting and ending location of an area of memory that is to be dumped.

Example:

```
CALL "TIPSNAP" USING    WORK-AREA END-WORK
                        MCS END-MCS
```

Additional Considerations:

- This call is useful when debugging programs but should be removed when placing a program in production.
- If the call is made using:

  ```
  CALL "TIPSNAP" USING MCS WORK-AREA
  ```

  The call will still occur but you may not get the contents of the snap. This is because TIP startup code uses UNIX MALLOC and

each area is allocated separately. It could be that the MCS and WORK-AREA may not be contiguous. If this happens, try using:

```
CALL "TIPSNAP" USING MCS END-MCS
```

where END-MCS is a field in the MCS area

- Micro Focus COBOL compiler directive "REF" should allows a programmer to correlate an address found in the TIPSNAP dump back to an address within the application program. This can speed up debugging time by allowing the programmer to find exact locations in the dump much faster than trying to progress it manually.

  Note: Inglenet does not release the "make.mf" file with this option turned on since it does make the listing much larger than usual.

TIPSUB - Perform Program

This call invokes another transaction program as if it was a subroutine of the calling program. The calling program is suspended while the called program executes. The called program may call another program, and so on, to a maximum of 16 nested calls. When a called program terminates, control returns to the calling program.

The classic example of the use of a facility such as TIPSUB is a program that offers a menu or choice of several other programs. Typically, a screen format is displayed that offers the terminal operator a number of choices of application systems.

Once the user has indicated his choice and the selection has been validated, the program calls TIPSUB to invoke the main transaction of the application subsystem.

When the application subsystem terminates, control returns to the original program, which repeats the cycle.

Syntax:

```
[ MOVE ? TO             PIB-CDA-LENGTH ]
  MOVE "????????" TO    PIB-TRID
  CALL "TIPSUB"
```

Where:

PIB-CDA-LENGTH
This field may be set to a value representing the maximum number of bytes in the CDA that are to be passed to the CDA of the program that is to be invoked.

PIB-TRID

Must be set to the transaction name of the program that is to be invoked.

If TIP Distributed Transaction Processing is configured, the program may move a LOCAP name to the field PIB-TID to indicate to the TIPSUB subroutine that the program that is to be performed is to execute on the LOCAP name specified.

The contents of the CDA of the calling program are copied to the CDA of the called program, to serve as the called program's initial CDA contents. On return from the TIPSUB call, the CDA contents of the called program are copied back to the CDA of the calling program.

The calling program's CDA data is copied to the CDA of the next program for a length which is the least of:

- the size of the calling program's CDA area
- the size of the called program's CDA area
- the value specified by the calling program in the field PIB-CDA-LENGTH.

Example 1:

```
MOVE SPACES TO          CDA
MOVE "PAYUP" TO         PIB-TRID
CALL "TIPSUB"
IF NOT PIB-GOOD
    PERFORM ERROR-ON-SUB
END-IF
```

Example 2:

```
* Perform "ACCTSUMM" txn on other LOCAP *
MOVE "PROD" TO          PIB-TID
MOVE "ACCTSUMM"         TO PIB-TRID
MOVE SPACES TO          CDA
CALL "TIPSUB"
IF NOT PIB-GOOD
    PERFORM ERROR-ON-SUB
END-IF
```

Error Conditions:

PIB-STATUS	Meaning
PIB-NOT-FOUND	The program is not defined, the load module could not be loaded or the size of the load module and required areas (CDA, WORK-AREA, MCS, etc.) is too large for available memory. If you receive bad status and want a more detailed description, use PIB-

PIB-STATUS	Meaning
	DETAIL-STATUS. See PIB-DETAIL-STATUS in PIB (Process Information Block). If the field PIB-TID was set to a different LOCAP name before the call to TIPSUB, this error condition is reported if the specified program is not found at the other LOCAP or (in the extreme case) the LOCAP name itself is not valid or a connection cannot be made with the LOCAP.
PIB-SECURITY	The user running the initiating program does not have high enough security to run the requested program or the requested program is locked due to the time of day.
PIB-PROG-ABEND	The called program aborted (program checked) during execution. In this case, PMDA is called on behalf of the called program and when PMDA has finished processing, control returns to the calling program with this error status. If the calling program receives PIB-PROG-ABEND status, the contents of the CDA are undefined (since PMDA uses the CDA as a work area).

Calling TIP Utilities

This section describes the procedures that must be followed when a user-written transaction program calls a utility transaction supplied with the TIP system.

The transaction programs that are supplied with the TIP system are written on the assumption that the programs are executed directly from the TIP command line (there are some minor exceptions to this general statement).

To successfully call these utilities, it is necessary for the calling program to carefully arrange the contents of the CDA to contain any needed parameters in exactly the same format as the data would appear if the transaction was called from the TIP command line.

In the following examples, it is assumed that the calling program defines the first 152 bytes of the CDA area using the supplied Copy book TC-CDA. Although the examples illustrate the use of TIPSUB to call the TIP

utilities, other methods of transferring control (TIPXCTL, TIPFORK, etc.) may be used if appropriate.

Example:

To run the standard “WHOSON” TIP utility from the command line, the user would enter the following:

```
TIP?►WHOSON
```

The user-written program calls the transaction in this manner:

```
MOVE SPACES TO           CDA
MOVE "WHOSON" TO         PIB-TRID
CALL "TIPSUB"
    IF NOT PIB-GOOD ...
END-IF
```

TIPSUBP - Call a Subprogram

TIPSUBP provides a way to emulate TIP/30’s CALL “TIPSUBP”, and IMS/90’s CALL “SUBPROG” features.

Syntax:

```
MOVE "????????" TO       PIB-TRID
CALL "TIPSUBP" USING     parameters
```

Where:

PIB- TRID
Must be set to the name of the subroutine to be invoked.

TIP/30 supports CALL “TIPSUBP” and IMS/90 supports CALL “SUBPROG”. Both of these calls allow transaction programs to call separately compiled subroutines. TIP/30 loads these subroutines into memory at system startup. This avoids having to re-compile applications when the subroutines change and saves memory by loading the subroutine only once.

A TIP/30 or IMS/90 application moves the subroutine name to PIB-TRID, then issues CALL “TIPSUBP” USING <parameters>.

On UNIX the appropriate method is to use “shared libraries”. On UNIX all executables are automatically shared and re-compilation time is very fast. Because of the protected memory spaces on UNIX it is not possible to jump from one application address space into another so the implementation of TIPSUBP and SUBPROG under TIP must be quite different.

Compile COBOL Subroutines

Compile your COBOL subroutines so that they can be used in a shared library. For details, see your COBOL vendor's documentation.

Add Subroutines to Library

All subroutines that may be invoked via TIPSUBP, must be compiled into object module format, and added into a normal UNIX archive library (using the UNIX ar command).

The library should at least hold all the subroutines that would be used by a particular group of transaction programs.

For details on how to create a shared library, see the documentation for your UNIX system. Unfortunately, these details tend differ for each version of UNIX.

Create TIPSUBP or SUBPROG

Once the library is created, use genmain to scan the library and construct a specific version of TIPSUBP (or SUBPROG) for that library. TIP modules and IMS modules must be kept in separate libraries:

For TIP:

```
genmain -S library.a
```

For IMS:

```
genmain -iS imslib.a
```

Where -S is the option to create TIPSUBP (or SUBPROG) and put it into the specified archive library.

Linking and Executing

The updated library may be used to link (ld) applications, which will invoke the subroutines via TIPSUBP.

Alternatively, the library could be converted into a shared library and then used when compiling, linking and executing the transaction programs.

Error Conditions:

Currently, none are returned.

TIPTIMER - Timer Services

This function allows the user program to pause for a specific length of time. An on line program may choose to delay its execution for a variety of reasons:

- To wait for an input message from the terminal
- To allow other users of the system access to the processor (to avoid monopolizing the processor)

- To wait for a specific number of seconds for some application related reason.

Syntax:

```
CALL "TIPTIMER" USING   wait-time
                       [time-status]
                       [preview]
```

Where:

wait-time

A binary full-word (PIC S9(9) BINARY) that specifies the number of seconds that the issuing program wishes to wait. This parameter is required.

TIP does not modify this field. If appropriate, it may be coded as a constant in the WORKING-STORAGE section of a COBOL program.

The program is reactivated when the specified number of seconds has elapsed, or an input message is available.

A value of zero in this field implies that the program does not wish to delay but is willing to relinquish control of the processor if some other process in the TIP system is ready to run.

Processes that would otherwise monopolize the system should periodically delay with a WAIT-TIME of zero. Candidates are processes that perform:

- □ CPU intensive activities
- □ Prolonged periods of sequential file reading.

Note: TIP cannot provide TIMER services with accuracy better than one second. The program is delayed at least the number of seconds that is specified.

If *wait-time* is set to a negative value, the value of the system parameter TIMEOFF in the tipix.conf file will be used as the time to wait. Since TIMEOFF is specified in minutes and TIPTIMER expects a value in seconds TIP calculates the default wait-time as (TIMEOFF * 60).

This is useful when a site would like to implement a standard wait time in their programs. If this technique is used then the wait time is easily altered by adjusting the TIMEOFF system parameter. For programs that must operate on both TIP and TIP/30 the value supplied (to request the default waittime) must be **-1**.

The following technique may be used by programs which wish to "wake up" at a specific time of day:

Obtain the current time of day from the operating system (the COBOL verb "ACCEPT" is handy for this).

Compute the number of seconds between the current time of day and the desired wake up time (taking into account possible day changes).

Issue a TIPTIMER call to wait for the computed number of seconds.

Be careful *not* to compare exactly for a specific time of day! It is better to check for a "greater than or equal to" condition to avoid missing the exact time.

Another method for getting scheduled at certain time of day is to use the TIPQUEUE facility. Define a "queue" which is to schedule the transaction even if there is no data in the queue at a certain time of day.

time-status

This parameter is optional and may be omitted if the next parameter is also omitted.

A one-byte status code that is set by the TIPTIMER subroutine to indicate the reason the program was reactivated. This result status is also returned in the field PIB-STATUS

PIB-MSG-AVAIL

An input message is available (the requested time has not elapsed).

- When this status is returned to the program, the program has an input message available. The normal course of action is to use one of the TIP subroutines (example: TIPMSGI, PARAM, etc.) to read the input message.
- An input message may have been the result of the terminal user pressing the XMIT key, a function key or the MSG-WAIT key.

PIB-TIMED-OUT

The specified number of seconds has elapsed and no input message is available
The two status codes are mutually exclusive. Only one of the two possible events can occur.

preview

This parameter is an optional 12-byte field into which TIP places the first 12 bytes (converted to status was returned). The contents of this field are not defined if TIPTIMER returns a status of "PIB-TIMED-OUT".

Example:

```
05  TIMER-WAIT                PICTURE S9(9)
                               BINARY.
05  TIMER-STATUS              PICTURE X.
 ...
    MOVE +60 TO                TIMER-WAIT
    CALL "TIPTIMER" USING      TIMER-WAIT
                               TIMER-STATUS
```

In this example, TIP suspends execution of the program for approximately 60 seconds or until an input message from the terminal is available.

If a message arrives, TIMER-STATUS contains "M" (PIB-MSG-AVAIL).

Calling TIPTIMER does **not** cause the TIP system to examine (or alter) the setting of the PIB-LOCK-INDICATOR.

Warning: Calling TIPTIMER with a wait time of 60 seconds or less does not cause the TIP system to release any file system record locks acquired by the process. This means that a process may delay for up to 60 seconds *while locking records*.

If a program that has locked one or more records calls TIPTIMER with a delay time exceeding 60 seconds, TIP aborts the program with the reason code: "Resources locked, waiting TIPTIMER".

TIPUSR - Where is User

This subroutine is called to return the name of the terminal where a specified TIP user is located. The subroutine searches for the specified TIP user on the system and returns the terminal name of the first location where that user is logged on.

Syntax:

```
CALL 'TIPUSR' USING     USER-PKT
```

Where:

USER-PKT
A group item in the program's work area where the user name is specified and the terminal name is returned.

The layout of the area is illustrated in the example that follows.

Example:

```
05  USER-PKT.
    10  USER-NAME        PICTURE X(8).
    10  USER-TERM        PICTURE X(4).
 ...
    MOVE SPACES          TO USER-PKT.
    MOVE 'ALLINSON'      TO USER-NAME.
    CALL 'TIPUSR' USING USER-PKT.
    IF USER-TERM = SPACES
        GO TO USER-NOT-ON.
```

Additional Considerations:

- If the specified user is not found on the system, the terminal name in the packet is set to spaces.

TIPUSRID - User Information

TIP programs use this call to retrieve information about a specified TIP user id. Information on the elective groups that the user belongs to and the comment from the user's TIP definition record are returned.

Syntax:

```
CALL "TIPUSRID" USING userid-DATA userid
```

Where:

user id-DATA
A group item in the program's work area where the result information is returned. The layout of the area is illustrated in the example that follows. The information returned includes the first two elective groups and comment information that is in the user's TIP definition.

user id
An eight-character field containing the user id to be used in the search for information.

Example:

```
05  userid-DATA.
    10  userid              PICTURE X(8).
    10  USER-GRP1           PICTURE X(8).
    10  USER-GRP2           PICTURE X(8).
    10  USER-CMT            PICTURE X(30).
...
    MOVE "ALLINSON"         TO userid
    CALL "TIPUSRID" USING   userid-DATA
                            userid
    IF NOT PIB-GOOD
        GO TO USER-DOESNT-EXIST
    END-IF
```

Error Conditions:

PIB-STATUS	Meaning
PIB-NOT-FOUND	The specified user id is not defined. The result area is cleared to spaces when this error condition occurs.

Additional Considerations:

- As shown in the example, the second parameter may safely be included in the area reserved for the result.

TIPUSRST – Set new User Information

TIP programs use this call to change the active TIP user id.

Syntax:

```
CALL "TIPUSRST" USING   new-user
```

Where:

New-user

A group item in the program's work area with the new user-id and password (if required). The format of the area is illustrated in the example that follows.

Example:

```
05  new-user.
    10  user-id              PICTURE X(8).
    10  Pass-Word            PICTURE X(8).
...
    MOVE "ALLINSON"          TO user-id
    MOVE  "Secret"           TO Pass-Word
    CALL "TIPUSRST" USING    new-user
    IF NOT PIB-GOOD
        GO TO USER-DOESNT-EXIST
    END-IF
```

Error Conditions:

PIB-STATUS	Meaning
PIB-NOT-FOUND	The specified user id is not defined. The result area is cleared to spaces when this error condition occurs.
PIB-SECURITY	The password did not match or the new userid has higher TIP security that the current userid.

TIPWINAP - Run a DOS or Windows Program

TIPWINAP has not been implemented in TIP Studio. For the functionality found with TIPWINAP please use the CreateProcess WIN32 call to start a "windows application".

This call can only be used with TIP/fe and with MCS running in SMART mode. After the call, TIP/fe will start up the desired program. If TIP/fe for Windows is being used, the program will run asynchronously to TIP/fe.

Syntax:

```
CALL "TIPWINAP" USING cmdline
```

Where:

Cmdline
Specifies the name of the DOS/Windows program to run. This buffer must be 80 bytes long.

When running the DOS version of TIP/fe, a program will be started up only if enough memory is available. Windows TIP/fe will not run DOS *.com* programs.

TIPWINAP will use the DOS PATH environment variable to find the executable to run if no path is specified.

Example:

```
MOVE 'ff c:\config.sys' TO  COMMANDLINE
CALL 'TIPWINAP' USING       COMMANDLINE
```

If 'ff' is in the path, it will be found and executed.

TIPXCTL - Transfer Control

The TIPXCTL subroutine transfers control to another program on the same program stack level - once the transfer of control is complete, the calling program terminates (control will not automatically return).

The contents of the CDA of the calling program are copied to the CDA of the called program, to serve as the called program's initial CDA contents.

The calling program's CDA data is copied to the CDA of the next program for a length, which is the least of:

- The size of the calling program's CDA area
- The size of the called program's CDA area
- The value specified by the calling program in the field PIB-CDA-LENGTH.

The calling program must move the name of the transaction to receive control to the PIB-TRID field and then call TIPXCTL.

Syntax:

```
[ MOVE ? TO                 PIB-CDA-LENGTH ]
  MOVE "????????" TO        PIB-TRID
  CALL "TIPXCTL"
```

Where:

PIB-CDA-LENGTH
This field may be set to a value representing the maximum number of bytes in the CDA that are to be passed to the CDA of the program that is being invoked

PIB-TRID
Must be set to the transaction name of the program to which control is to be transferred

Example:

```
MOVE ...                    TO CDA
MOVE "NXTSTP"               TO PIB-TRID
CALL "TIPXCTL"
PERFORM ERR-ON-XCTL
```

Error Conditions:

PIB-STATUS	Meaning
PIB-NOT-FOUND	The program is not defined, or the load module could not be loaded, or the field PIB-TID was erroneously modified by the program prior to calling TIPXCTL. If you receive bad status and want a more

PIB-STATUS	Meaning
	detailed description, use PIB-DETAIL-STATUS. See PIB-DETAIL-STATUS in PIB Process Information Block on page 20.
PIB-SECURITY	The user running the initiating program does not have a high enough security to run the requested program or the transaction is locked at this time of day.

Example:

An example of the use of TIPXCTL is to provide a means for a transaction program to offer the user the ability to both exit the transaction *and* logoff the TIP system.

To accomplish this, the program includes code such as this:

```
MOVE SPACES                    TO CDA
MOVE "LOGOFF"                  TO PIB-TRID
CALL "TIPXCTL"
CALL "TIPERASE"
MOVE "UNABLE TO LOGOFF"        TO ERROR-TEXT
CALL "ROLL" USING              ERROR-TEXT
CALL "TIPRTN"
```

Hint:

- The code illustrated above does not need to check whether or not the transfer of control to the LOGOFF transaction was completed (if the TIPXCTL failed for any reason, the program is given control back after the TIPXCTL).

The LOGOFF program will refuse to perform its function unless LOGOFF is called at stack level 1. LOGOFF is not permitted if the program stack is not empty.

Message Control System (MCS)

This chapter describes the facilities provided by TIP to enable an online program to perform input and output operations at a terminal.

Provided Interfaces

Three levels of interface are provided:

Interface	Description
Message Control System (MCS)	MCS is a high level interface; that is, it allows application programmers to develop screen formats (templates) and use them in online programs. Using MCS, the programmer can achieve a high degree of hardware independence.
Line-Oriented Input/Output	The Line-oriented I/O interface consists of a number of subroutines, which facilitate the interactive use of the terminal in a line-by-line fashion. A program using these subroutines issues one line prompts and retrieves single line replies.
Direct Communications Input/Output (DCIO)	The DCIO interface allows the program to exercise direct control over the activity of the terminal. This is a low level interface that requires the application programmer to supply the control codes that are to be sent to the terminal. The DCIO interface is primarily intended for unusual applications that require direct control of the terminal. It is intended for use only when the facilities of the higher level interfaces (MCS or Line-Oriented I/O) cannot achieve the desired results.
Terminal Paging	TIP provides paging, an efficient way to save screens (pages) into a file, and access them. Each page contains all the information necessary to repaint a full screen including the data.

MCS Screen Formats

The TIP Message Control System provides the capability to create, test and use screen formats (templates) in online programs. These screen formats are unique because they are not defined in the programs that use them. The user program sends and receives only data field information to and from the terminal.

The MCS System handles all communications codes and heading information. There are four major components of MCS; three are online utility transaction programs:

Utility	Description
TFD	Utility transaction to define and update screen formats.
MSGSHOW	Utility transaction to test screen formats.
MSGAR	Utility transaction that provides librarian services for screen formats.

The fourth component of MCS is the Message Formatter:

MSGFMT
: The message formatter an internal part of TIP that provides an interface between the formats and the data supplied by the program. MSGFMT is the TIP format handler.

  For output operations, it merges user data supplied in the MCS interface packet, with the information in the screen format and sends it to a terminal.

  For input operations, MSGFMT extracts the data from the incoming communications message and stores it in the MCS interface packet.

MCS Interface Packet

The layout of the data area of the MCS interface packet is similar to that of a fixed-length data record. There is no provision for tab stops or cursor coordinates; such items are defined in the screen format by TFD and handled completely by MSGFMT at user program execution time.

Optimization of Output Messages

The Message Formatter optimizes all output messages. For example, in the interest of efficiency, a series of blanks may be replaced with a cursor positioning code sequence.

Common Carrier Lines

MCS optimization can make a significant improvement in communication throughput; especially over low speed Common Carrier lines.

Other Advantages

Any online program may call any defined format using its assigned eight-character name. Furthermore, the programmer may change heading information in screen formats without changing the programs that use them. User programs only process the data since the Message Formatter in TIP handles all communications control characters and heading information.

These features greatly reduce the programming effort and development time required to put online programs into production.

How Screen Formats Work

The following diagrams illustrate the relationship between the program and the TIP screen format. Further information about how a screen format is defined may be found in the description of the utility transaction TFD (TIP Format Definition).

Assume that a screen format has been defined as follows:

```
             ACME WIDGET COMPANY

   Name: UUUUUUUUUUUUUUUUUUUUUUUUU

Address: UUUUUUUUUUUUUUUUUUUUUUUUU
         UUUUUUUUUUUUUUUUUUUUUUUUU
         UUUUUUUUUUUUUUUUUUUUUUUUU

Balance: -ZZZ,ZZZ,ZZ9.99


EEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEE
```

The first line is heading information; the strings of "U" characters define uppercase data fields; the "Balance" field is a numeric field with a floating and leading negative sign and floating zero suppression. The string of "E" characters defines an "error" field that may be used by a program to output informational or error messages.

When the facilities of MCS are used by an online program, the program only defines the data fields that correspond to the data fields defined in the screen format. For example, referring to the screen format show above, the program might define 5 fields this way (error fields are not considered data fields in this context and are handled separately):

```
05  S-NAME                  PICTURE X(25).
05  S-ADDR-1                PICTURE X(25).
```

```
05  S-ADDR-2                    PICTURE X(25).
05  S-ADDR-3                    PICTURE X(25).
05  S-BALANCE                   PICTURE S9(9)V99.
```

The program deals with the data fields - the heading information and the automatic output editing capabilities of the screen handler are transparent to the user program.

To output data for example, a program moves the desired data to the appropriate fields and calls the TIP MCS subroutine "TIPMSGO" to output the screen format and the data supplied by the program.

Conversely, a call to the TIP MCS subroutine "TIPMSGI" causes data from the screen to be placed in the program's data fields — the program does not need to be concerned with the mechanics of the terminal operation or the communication sub-system.

MCS Subroutines

An online TIP program uses TIP screen formats by issuing subroutine calls to the TIP Message Control System to transfer data to and from the terminal. The subroutines are summarized as follows:

Subroutine	Description
TIPASK	May display a one line question and returns a one-line answer from the user to the application program. Pops up a small window with the question and answers fields and then removes the window when the answer is given.
TIPASKYN	Similar to TIPASK except that only a single character reply ("Y" or "N") is accepted.
TIPERASE	Erase screen. Also removes all windows pushed onto the stack.
TIPLIST	Use to invoke help text processing from within an application program.
TIPMENU	Define an 80-byte menu bar.
TIPMSGE	Send "error" message to terminal.
TIPMSGEO	Define a deferred error message.
TIPMSGI	Input data from terminal.
TIPMSGO	Output data to terminal.
TIPMSGOV	Display an MCS screen format and overlay the

Subroutine	Description
	current screen.
TIPMSGPR	Outputs an MCS screen format logical contents (headings and data) as print lines to TIPPRINT (TIP printing interface).
TIPMSGRS	Pops the current screen off and restores the contents previously (immediately) displayed.
TIPMSGRV	Force read terminal screen.
TIPTITLE	Define an 80-byte screen title bar.

All of these subroutines are described in subsequent sections. The following section describes the interface packet that these subroutines use to control the action of the subroutine.

Program Control after CALL

The online program issues CALLs to these subroutines and receives control directly following the CALL to the subroutine.

This means that online programs can transfer data to and from the terminal in much the same manner as a batch program transfers data to and from a disk file (for example).

The MCS interface provides hardware independence by requiring the program to handle only the data fields.

Using Screen Formats

The following code fragment illustrates the *general* structure of a TIP program that uses screen formats. Do not interpret the following code literally - use it to conceptualize the general structure.

Example:

```
SEND-OUTPUT.
    CALL "TIPMSGO" USING ...
GET-INPUT.
    CALL "TIPMSGI" USING ...

    IF USER-REQUESTED-EXIT
        GO TO END-PROGRAM
    ELSE
         --evaluate input data--
    END-IF
```

```
IF ANY-ERRORS-DETECTED
    MOVE ERROR TEXT TO ERROR-MESSAGE-TEXT
    CALL "TIPMSGE" USING ...
    GO TO GET-INPUT
     --update information on file etc.--
END-IF
GO TO SEND-OUTPUT
```

This program fragment is intentionally not structured the way code usually is; it merely illustrates that the "flow" of an online program can be quite straightforward and need not involve programming concepts that differ radically from batch programming.

Sample Program tstwin

The TIP release includes the COBOL source for a TIP demonstration program named **tstwin.cbl.** The **tstwin** program illustrates how to use the new windowing features of TIP, that are supported by the MCS facility.

MCS Interface Packet

TC-MCS copybook

The COBOL copybook TC-MCS in the TIP library defines the MCS interface packet. The MCS interface packet controls the interface between an online program and the TIP Message Control System. The Message Control System assumes that this interface packet immediately precedes the fields that contain the data for the screen format that is in use.

```
*
* TIP - MESSAGE CONTROL SYSTEM PACKET
*
    02  MCS-NAME                PICTURE X(8).
    02  MCS-TERM                PICTURE X(4).
    02  MCS-FUNCTION            PICTURE X.
        88  MCS-SEND-FULL           VALUE " ".
        88  MCS-RECEIVE-ALL         VALUE "A".
        88  MCS-DATA-ONLY           VALUE "D".
        88  MCS-UNSOLICITED         VALUE "M".
        88  MCS-SCREEN-PRINT        VALUE "P".
        88  MCS-REFRESH             VALUE "R".
        88  MCS-SHORT-XMIT          VALUE "S".
    02  MCS-HOLD                PICTURE X.
        88  MCS-KEYBOARD-LOCK       VALUE "L".
    02  MCS-SIZE                PICTURE S9(4) COMP-4.
    02  MCS-STATUS              PICTURE X.
        88  MCS-GOOD                VALUE " ".
        88  MCS-XMIT                VALUE " ".
        88  MCS-MSG-WAIT            VALUE "0".
        88  MCS-FKEY1               VALUE "1".
        88  MCS-FKEY2               VALUE "2".
```

```
           88  MCS-FKEY3              VALUE "3".
           88  MCS-FKEY4              VALUE "4".
           88  MCS-FKEY5              VALUE "5".
           88  MCS-FKEY6              VALUE "6".
           88  MCS-FKEY7              VALUE "7".
           88  MCS-FKEY8              VALUE "8".
           88  MCS-FKEY9              VALUE "9".
           88  MCS-FKEY10             VALUE "A".
           88  MCS-FKEY11             VALUE "B".
           88  MCS-FKEY12             VALUE "C".
           88  MCS-FKEY13             VALUE "D".
           88  MCS-FKEY14             VALUE "E".
           88  MCS-FKEY15             VALUE "F".
           88  MCS-FKEY16             VALUE "G".
           88  MCS-FKEY17             VALUE "H".
           88  MCS-FKEY18             VALUE "I".
           88  MCS-FKEY19             VALUE "J".
           88  MCS-FKEY20             VALUE "K".
           88  MCS-FKEY21             VALUE "L".
           88  MCS-FKEY22             VALUE "M".
           88  MCS-FPOC               VALUE "N".
           88  MCS-TIMED-OUT          VALUE "T".
           88  MCS-F-REBUILD          VALUE "1" "5" "N".
           88  MCS-F-NEXT             VALUE "2" "6".
           88  MCS-F-UPDATE           VALUE "4" "8".
           88  MCS-F-FIELD            VALUE "<".
           88  MCS-F-MENU             VALUE ">".
       02  MCS-FILLER             PICTURE X.
           88  MCS-UNDERLINE          VALUE "_".
           88  MCS-ASTERISK           VALUE "*".
           88  MCS-SPACE              VALUE " ".
       02  MCS-COUNT              PICTURE S9(4) COMP-4.
/
       02  MCS-DATA.
*
* USER SUPPLIED RECORD LAYOUT FOR MCS SCREEN FOLLOWS
```

If an online program uses more than one screen format, the program redefines the MCS-DATA area to account for the differing layouts of the screen formats.

An online program interfaces with MCS through subroutine calls that transfer data to and from the terminal. These subroutines use the information placed in the interface packet.

The following is a description of the fields that make up the MCS packet

MCS-NAME

A field that must contain the desired screen format name.

Screen formats are assigned a name when they are defined using the TFD program. The format name may be up to eight characters in length and must start with a character that is not a digit.

If the field MCS-NAME contains underscore character(s), MCS replaces underscores with the user's LANGUAGE= code (as specified in the USER definition record), and attempts to find that screen format. If the user does not have a language code assigned, underscores are replaced with the letter "A".

MCS-TERM

This field is used to specify the intended destination of an output message (if it is different than the terminal that is issuing the call to the MCS subroutine).

If this field does not contain a valid terminal name (namely: spaces or low values), the screen format I/O is directed to the terminal where the program is running.

If the specified terminal is not currently connected to TIP the terminal name is ignored

MCS-FUNCTION

This field specifies additional optional processing. Each MCS subroutine description includes a discussion of the relevant values of this field.

MCS-HOLD

This field may be set to the value "L" before calling TIPMSGO to lock the terminal keyboard following delivery of the output message.

The contents of this field are not preserved - the program must insert the desired value before issuing a call to MCS

MCS-SIZE

MCS sets this field to the maximum number of bytes that may have been received as a result of an input message. The online program can use this value to determine whether the data received on an input message represents a "full screen". This is discussed in the description of the TIPMSGI subroutine call.

The online program should not modify this field.

This field is set to the appropriate value after a call to an MCS subroutine (for example, TIPMSGI).

MCS-STATUS

MCS sets this field after a call to request terminal input. The value indicates what type of terminal activity was detected: for example, MSG WAIT or a function key or XMIT. Various 88 level items are provided to simplify program coding.

After an output message (TIPMSGO or TIPMSGE), if an input message is already available this field is set to the value "M".

The special status code MCS-F-FIELD indicates that control was returned to the application because of field level input. A field, for which the application had requested field level input, was changed and then exited.

The special status code MCS-F-MENU indicates that control was returned to the application because the user selected an item from the on-screen menu bar (see the documentation for the subroutine TIPMENU).

MCS-FILLER

This field is set to the desired "fill" character to use on output. Choices are: space, underscore or asterisk character.

During TIPMSGO, the fill character is used to replace:

leading spaces in unprotected numeric fields (caused by zero suppression)

trailing spaces in unprotected alphanumeric fields

Fill characters are not used in protected data fields. Fill characters received from the terminal during TIPMSGI are replaced by spaces or zeroes depending on the field type.

This field is not modified by MCS

MCS-COUNT

The TIPMSGO subroutine expects this field to contain a count of the number of data bytes in the MCS-DATA area that are output to the screen format.

If this value is less than the maximum number of data field bytes in the format, the MCS formatter uses the MCS-FILLER character in data fields, which follow the fields implied by the count.

If greater, any excess (trailing) bytes are ignored.

When terminal input is received, the value in this field indicates the number of data characters received:

The input count is always less than or equal to the value that MCS reports in the MCS-SIZE field (the maximum) and always includes the full size of the last field where the cursor was resting.

For example, if the terminal operator enters a partial value in a long field and presses XMIT somewhere within that field, the value in MCS-COUNT will be adjusted upward to

include the full length of that field. The field itself in the MCS-DATA area will be padded on the right with the appropriate character depending on the type of field (numeric or alphanumeric).

MCS-DATA

This group item defines the start of the data fields that are defined in the screen format.

The elementary fields in this group item must be defined by the programmer in the same order as they appear in the screen format (top to bottom and left to right). The type and size (in bytes) of the elementary fields must also match the definition of the field that was specified when the screen format was defined.

Define the fields in this group item as display type fields packed, binary or floating point fields are not permitted.

Use the COBOL command provided by the MSGAR online utility program to create a library element containing the field layout corresponding to a screen format. This library element can then be tailored and placed following the MCS-DATA group item

MCS Subroutine CALLS

TIPASK - Display One Line and Return Answer

TIPASK displays a one-line question and then returns the one line answer from the user to the application program. TIPASK is like PROMPT but it pops up a small window with the question and answer fields and then removes the window when the answer is given. The original screen contents are preserved and restored when the TIPASK window is cleared.

To use this MCS routine from TIP/as your program must be defined as a TIP/ix program rather than a TIP/30 program, and you need to link your program with TIPIXAPI32U.LIB rather than TIP30API32U.LIB.

Syntax:

```
CALL "TIPASK" USING     reply-text
                        question-text
```

Where:

reply-text

The result field TIPASK places the user's reply (max of 80 characters) in this field.

The contents of the reply field are displayed on the screen as the initial data in the answer field (if you do not want anything displayed as a default response, move SPACES to the reply field before issuing the CALL).

question-text

A field (maximum 80 bytes) containing the text of the question to ask.

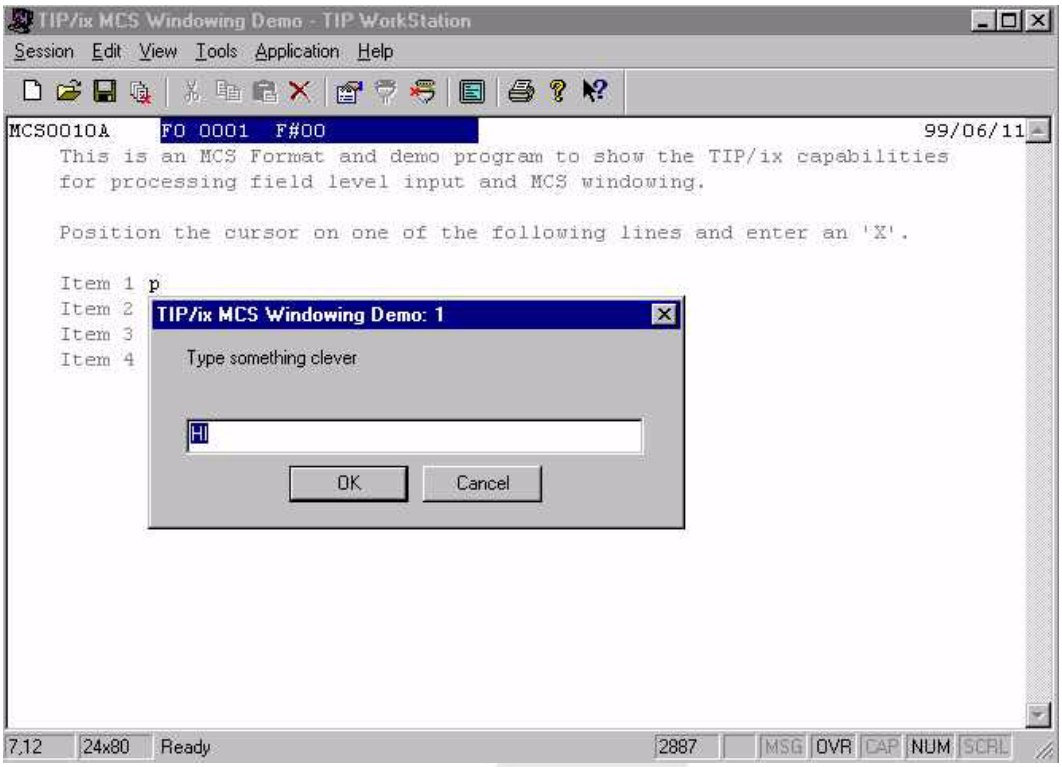
Example:

```
WORKING-STORAGE SECTION.
....
05  QUESTION                PICTURE X(80)
    VALUE "Please supply your company name".
.....
LINKAGE SECTION.
01  WORK-AREA.
    05  REPLY-TEXT          PICTURE X(80).
.....
PROCEDURE DIVISION USING    PIB
                            CDA
                            MCS
                            WORK-AREA.
....
    CALL "TIPASK" USING     REPLY-TEXT
                            QUESTION
```

Additional Considerations:

If possible, the "ask" window is positioned near the field where the cursor is located.

Example of TIPASK Window Prompt:



TIPASKYN - Display One Line and Return Answer

TIPASKYN is similar to TIPASK except that only a single character reply is accepted.

To use this MCS routine from TIP/as your program must be defined as a TIP program rather than a TIP/30 program, and you need to link your program with TIPIXAPI32U.LIB rather than TIP30API32U.LIB.

Syntax:

```
CALL "TIPASKYN" USING   reply-text
                        question-text
```

Where:

reply-text
The result field TIPASKYN places the user's reply (max 1 character) in this field.

question-text
A field (maximum 80 bytes) containing the text of the question to ask. The text of the question should provide a clue as to the single character replies that are acceptable

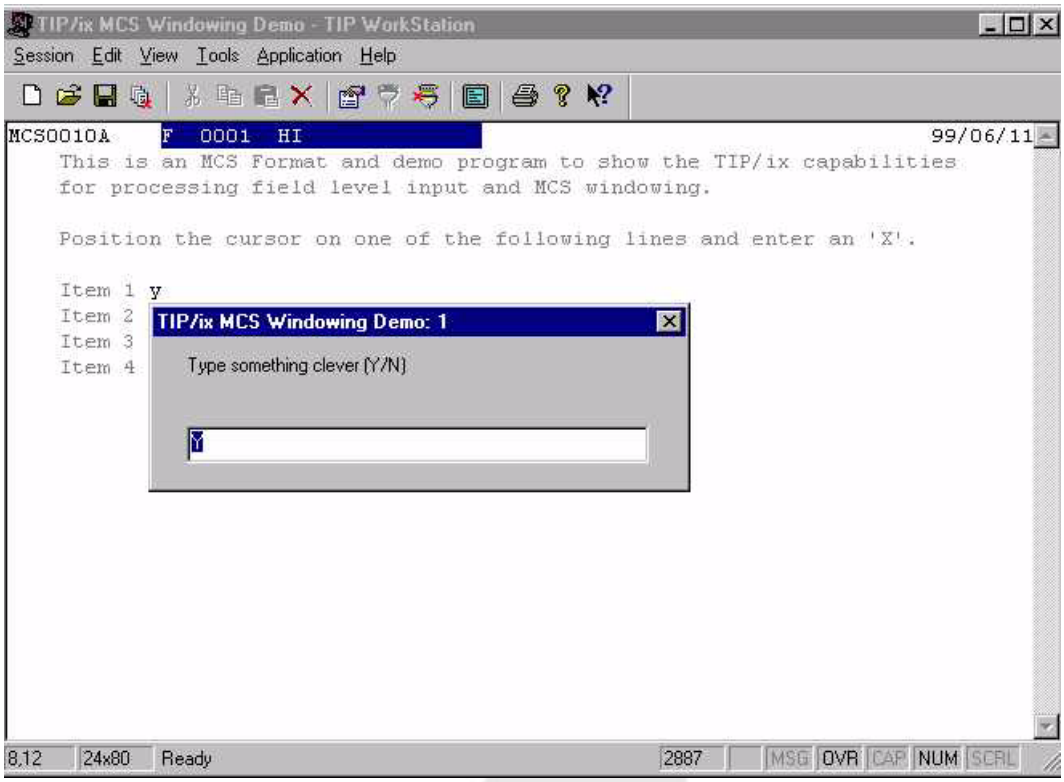
Additional Considerations:

- For TIPASKYN the contents of the reply field are displayed on the screen as the initial data in the answer field. If you do not want anything displayed move SPACES to the reply field before the CALL.
- If possible, the "ask" window is positioned near the field where the cursor is located.

Example:

```
WORKING-STORAGE SECTION.
 ....
05  OK-DEL              PICTURE X(80)
 VALUE  "Ok to delete this record? (Y/N)"
 .....
LINKAGE SECTION.
01  WORK-AREA.
    05  RESPONSE        PICTURE X.
 .....
PROCEDURE DIVISION USING    PIB
                            CDA
                            MCS
                            WORK-AREA.
 ....
    MOVE "N" TO RESPONSE
    CALL "TIPASKYN" USING   RESPONSE
                            OK-DEL
```

Example of TIPASKYN Window Prompt:



TIPERASE - Erase Screen

The TIPERASE subroutine erases the terminal screen. The program may want to make this function part of the processing that occurs when the program terminates. If any overlay screens were on the screen (placed there by calls to TIPMSGOV), they will be removed too.

Syntax:

```
CALL "TIPERASE"
```

Additional Considerations:

The entire screen is erased. Protected and unprotected data and heading information is removed.

Example:

```
...
CALL "TIPMSGI" USING    MCS
IF MCS-FKEY4
    CALL "TIPERASE"
    CALL "TIPRTN"
END-IF
```

The above example illustrates a technique to detect function key **F4** and erase the screen before exiting the program.

TIPLIST - Pick From a List

The TIPLIST subroutine can be used to display help text that is externally defined in the associated screen format or to display application-supplied data in a list format. Displayed data can be selected by the terminal user and **XMIT** pressed to notify the application which line item is selected.

The list may include headings, comments for each item, scroll bars (for lists larger than life), and hot keys for rapid selection.

There are three versions of syntax for this subroutine.

Note that to create a list larger than 99 rows or 5000 bytes in total you must use Syntax 3, which has no limit.

To use this MCS routine from TIP/as your program must be defined as a TIP program rather than a TIP/30 program, and you need to link your program with TIPIXAPI32U.LIB rather than TIP30API32U.LIB.

Syntax 1 (2 parameters):

```
CALL "TIPLIST" USING    help-name
                        sel-text
```

Where:

help-name
The first parameter holds the name (PIC X(8)) of externally defined help text that you wish to be displayed. The help text must have been defined at the time the screen format was created with TFD.

sel-text
On return from the call, the field defined by the second parameter contains the user-selected text.

The field PIB-MCS-KEY indicates which function key or whether XMIT was pressed. The field PIB-MCS-FIELD holds the line number of the selected text. Only selectable text lines are numbered. This field must be defined as X(80).

Example:

```
WORKING-STORAGE SECTION.
 ...
    05  HELP-NAME            PICTURE X(8)
        VALUE "APPLHELP".
 ...
LINKAGE SECTION.
```

```
01  WORK-AREA.
    05  SEL-TEXT              PICTURE X(80).
 ...
PROCEDURE DIVISION USING       PIB
                               CDA
                               MCS
                               WORK-AREA.
 ...
    CALL "TIPLIST" USING       HELP-NAME
                               SEL-TEXT
```

Syntax 2 (3 parameters):

```
CALL "TIPLIST" USING    help-size
                        sel-text
                        list-data
```

Where:

help-size

The first parameter (PIC x(8)) contains two 2-digit fields that declare (respectively) the number of fields and size, in bytes, of each field passed as the third parameter

sel-text

On return from the call, the field defined by the second parameter contains the user-selected text. This field must be defined as X(80).

list-data

The third parameter defines the list data (the number of lines and the size of each line as specified in the HELP-SIZE parameter above.)

The first capitalized letter in the text is considered a "hot key" character at run-time so that the user can quickly move to that line by pressing the "hot key" instead of scrolling through the list.

The field PIB-MCS-KEY indicates which function key or whether XMIT was pressed. The field PIB-MCS-FIELD holds the line number of the selected text. Only selectable text lines are numbered.

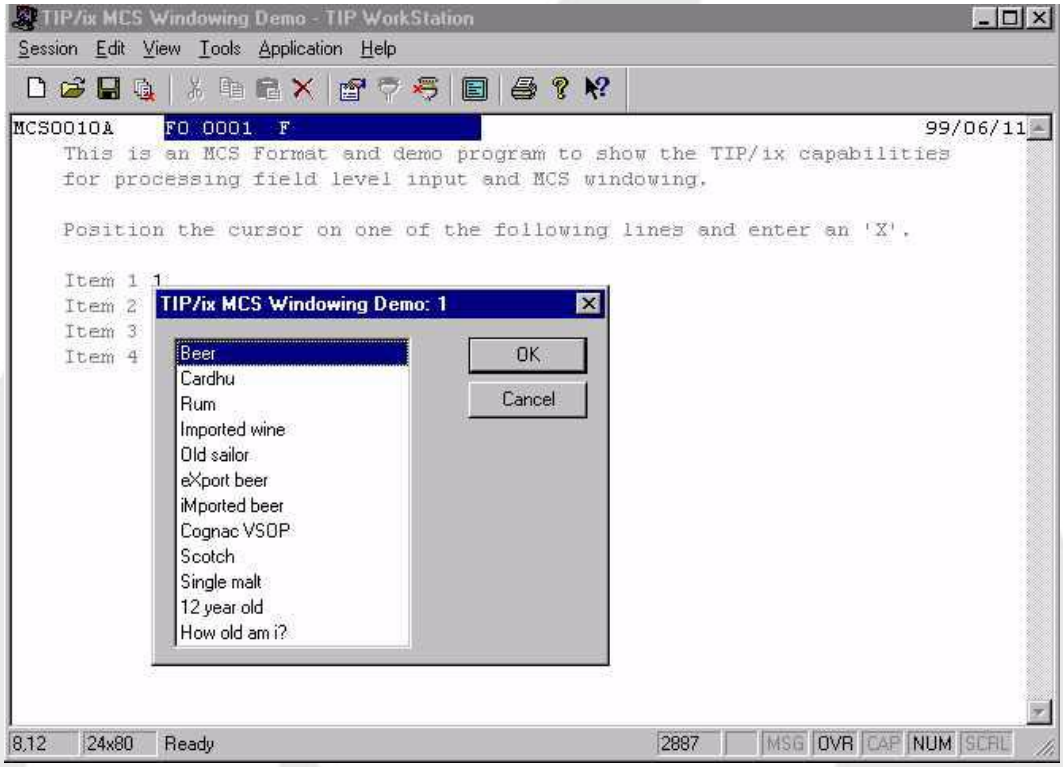
Example:

```
    05  HELP-SIZE              PICTURE X(8)
        VALUE "0635".
.....
    05  LIST-DATA.
```

```
           10  DAT-LINE          PICTURE X(35)
                                 OCCURS 6 TIMES.
     .....
        05  SEL-TEXT             PICTURE X(80).
     ....
   PROCEDURE DIVISION USING      PIB
                                 CDA
                                 MCS
                                 WORK-AREA.
     ....
        CALL "TIPLIST" USING     HELP-SIZE
                                 SEL-TEXT
                                 LIST-DATA
```

Using TIPLIST with application-supplied data allows you to collect information and present it to the end user as a list. The end user may then select some item from the list and respond by pressing **XMIT** or a function key. The application could then update, delete, add or display more detailed information on the item selected.

Example of TIPLIST in action:



In the second format of the call to TIPLIST (with three parameters), the first line of data *may* contain keywords that are used to specify the position of the list and special processing. The order of the keywords is not significant:

Syntax 3 (4 parameters):

```
CALL "TIPLIST" USING    rows
                        Cols
                        sel-text
                        list-data
```

Where:

rows
Lets you select the number of rows that appear in a list.

cols
Lets you select the number of columns that appear in a list.

sel-text
On return from the call, the field defined by the second parameter contains the user-selected text. This field must be defined as X(80).

list-data
The fourth parameter defines the list data (the number of lines (rows) and the size of each line (cols) as specified in the parameters above.)

The first capitalized letter in the text is considered a "hot key" character at run-time so that the user can quickly move to that line by pressing the "hot key" instead of scrolling through the list.

The field PIB-MCS-KEY indicates which function key or whether XMIT was pressed. The field PIB-MCS-FIELD holds the line number of the selected text. Only selectable text lines are numbered.

Example:

```
 ...
WORKING-STORAGE SECTION.
 ...
01  HELP-SIZE                PICTURE X(8)
                              VALUE '0506 '.
 ...
01  LIST-DATA.
    10  FILLER               PICTURE X(50)
    VALUE 'HEADCHAR=$,CMTCHAR=*,STYLE=LIST,POS=10,38'.
    10  FILLER               PICTURE X(50)
        VALUE '$This is a Header'.
    10  FILLER               PICTURE X(50)
        VALUE 'LINE 2'.
    10  FILLER               PICTURE X(50)
        VALUE '*This is a comment.'.
    10  FILLER               PICTURE X(50)
        VALUE 'LINE 4'.
    10  FILLER               PICTURE X(50)
        VALUE '*You may put comments here.'.
```

```
           10  FILLER              PICTURE X(50)
               VALUE 'LINE 6'.
           10  FILLER              PICTURE X(50)
               VALUE '*Please press Return.'
        ...
        01 WORK-AREA.
           05  ROWS                PICTURE 9(8) BINARY.
           05  COLS                PICTURE 9(8) BINARY.
           05  SEL-TEXT            PICTURE X(80).
        ...
       PROCEDURE DIVISION USING     PIB
                                    CDA
                                    MCS
                                    WORK-AREA.
       0000-INITIALIZATION.
           MOVE 8                   TO ROWS.
           MOVE 50                  TO COLS.
           CALL 'TIPLIST' USING     ROWS
                                    COLS
                                    SEL-TEXT
                                    LIST-DATA.
           CALL 'ROLL' USING        SEL-TEXT.
           CALL "TIPRTN".
```

Using TIPLIST with application-supplied data allows you to collect information and present it to the end user as a list. The end user may then select some item from the list and respond by pressing **XMIT** or a function key. The application could then update, delete, add or display more detailed information on the item selected. Using ROW and COLS you may control the size of your list or menu.

This is an example of a TIPLIST generated list:

*** missing picture ****

Options associated with TIPLIST:

```
HEADCHAR=x,CMTCHAR=x,LINES=nn,STYLE=xxxx,
 POS=r,c, SELECT={YES|NO|AUTO}
```

Syntax:

```
05  FILLER          PICTURE X(50)
 VALUE "CMTCHAR=*,LINES=03,STYLE=MENU,POS=14,15".
```

Where:

HEADCHAR=x

This keyword specifies the single character that is to be considered a marker for initial lines of data that are to be treated as headings for the list. The initial lines of data that begin with this character are used to construct a heading or title box for the list.
Default heading character is "!"

CMTCHAR=x

This keyword specifies the single character that is to be considered a marker for a comment line. A comment line may follow a data line in the list data. Comments are displayed in a box at the bottom of the list when the cursor rests on a particular list item.

Default heading character is "#"

LINES=nn

This keyword specifies the number of lines of data to be presented in the list (if there are more items, a scrolling bar is also displayed.) The value specified must be between 2 and 20 inclusive; a value less than 2 is set to 2, values greater than 20 are set to 20.

If this keyword is omitted or is not conformable with the POS= keyword, MCS selects a number of lines that is dependent on the position of the list on the screen.

STYLE=xxxx

This keyword specifies the style desired for the list: STYLE=LIST or STYLE=MENU

If this keyword is omitted, the LIST style is used.

POS=r,c

This keyword specifies the row number (r) and the column number (c) where the upper left corner of the list is to be placed. The list is placed as close as possible to the specified location.

If this keyword is omitted, the list is placed as close as possible to the cursor location without obscuring the field where the cursor is resting.

SELECT=

YES The user must explicitly press ENTER or XMIT to select an item. This is the default.

NO Do not allow user to select an item. The user can only use ESCAPE or MSG WAIT to exit the TIPLIST.

AUTO

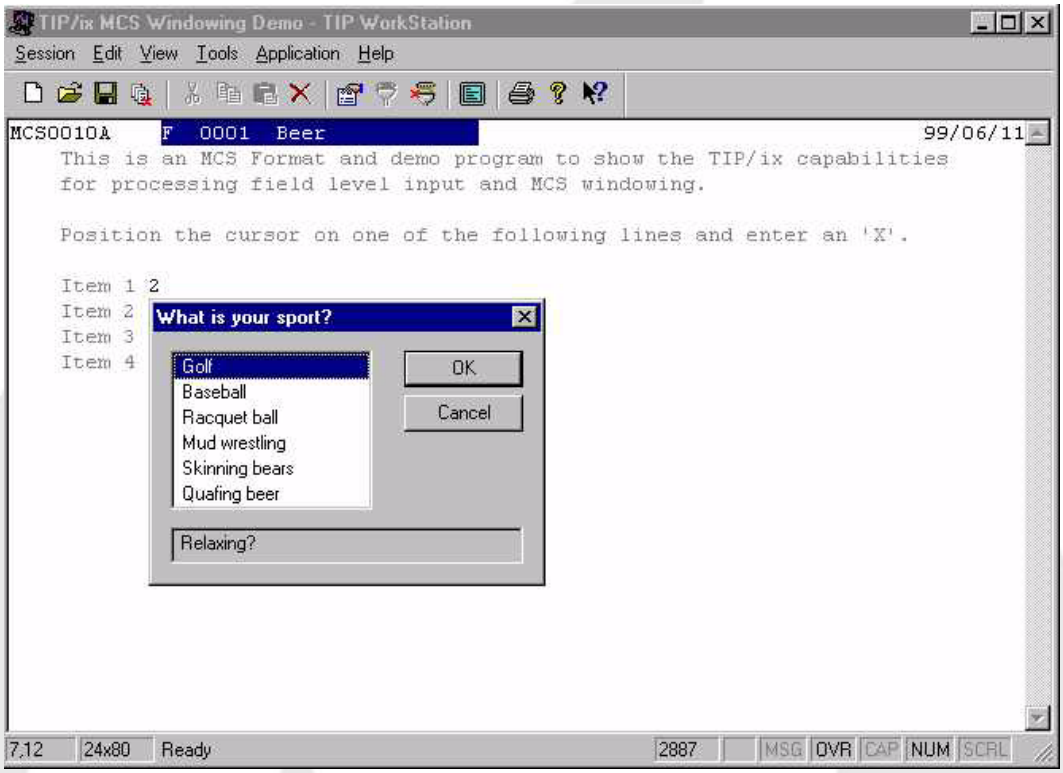
Automatically select an item (as if ENTER or XMIT were pressed) when the first or capitalized letter is typed (if it is unique).

Example of a List with heading and comments

```
01  IN-HELP2                PICTURE X(8) VALUE "1330".
01  IN-TEXT2.
    05  FILLER              PICTURE X(30)
        VALUE "!What is your sport?".
    05  FILLER              PICTURE X(30)
        VALUE " Golf ".
    05  FILLER              PICTURE X(30)
        VALUE "# Relaxing? ".
```

```
05  FILLER              PICTURE X(30
    VALUE " Baseball ".
05  FILLER              PICTURE X(30
    VALUE "# Go Blue jays ".
05  FILLER              PICTURE X(30
    VALUE " Racquet ball ".
05  FILLER              PICTURE X(30
    VALUE "# Smack it hard".
05  FILLER              PICTURE X(30
    VALUE " Mud wrestling ".
05  FILLER              PICTURE X(30
    VALUE "# Male or female".
05  FILLER              PICTURE X(30
    VALUE " Skinning bears".
05  FILLER              PICTURE X(30
    VALUE "# Sandy's favorite".
05  FILLER              PICTURE X(30
    VALUE " Quaffing beer ".
05  FILLER              PICTURE X(30
    VALUE "# Barry's favorite".
```

This coding appears in the TIP sample program **tstwin**. The list coded above is displayed as follows:



The cursor is resting on the selection "Golf" and the corresponding comment line "Relaxing?" appears in the comment box at the bottom of the list. As the user moves up or down through the list, the comment changes to reflect which item is currently in focus.

TIPMENU - Display Menu Bar

The call TIPMENU displays a "LOTUS 1-2-3 style" 80-character menu bar containing specific keywords that the terminal user can *later* select to perform specific actions. The menu bar is displayed on the top line of the screen unless a prior call to TIPTITLE has used line 1 (in that case, the MENU line appears on line 2.)

To use this MCS routine from TIP/as your program must be defined as a TIP program rather than a TIP/30 program, and you need to link your program with TIPIXAPI32U.LIB rather than TIP30API32U.LIB.

Syntax:

```
CALL "TIPMENU" USING menu-text
```

Where:

menu-text

The first parameter is an alphanumeric field containing the menu choices that are to be offered to the terminal user on the menu bar. This field defines an area of exactly 80 bytes; each 10-byte subfield can be used as a menu choice. The following example shows how menu text is constructed:

```
01 MENU1.
 05 FILLER PIC X(10) VALUE "Display ".
 05 FILLER PIC X(10) VALUE "Update ".
 05 FILLER PIC X(10) VALUE "Cancel ".
 05 FILLER PIC X(10) VALUE "End ".
 05 FILLER PIC X(10) VALUE "Quit ".
 05 FILLER PIC X(10) VALUE "Home ".
 05 FILLER PIC X(20) VALUE " Pick one and Enter".
 01 FILLER REDEFINES MENU1.
 05 MENU-ITEM OCCURS 8 PIC X(10).
```

Note: The first subfield (or group of 10 bytes) that contains a leading space character is considered the end of the choices. In the above example, there are 6 choices; the text "Pick one and Enter" is merely placed as a comment at the end of the menu bar.

Additional Considerations:

- This call only displays the menu bar on the screen. To select an item from the menu bar, the user must press the keyboard key that is assigned to the functionality "go to menu bar". See the definition of keyboard mapping in “TIP Installation and Operation” under the heading "Terminal Interface" for additional information. The default key to enter the menu bar is CTRL-\. Once the user enters the menu

bar, a menu item can be selected and XMIT can be pressed. When the menu item is selected and XMIT is pressed, the status code MCS-F-MENU is set and the program returns from TIPMSGI and can take whatever action is appropriate.

- The selected item is returned in PIB-MCS-FIELD as an item number.

TIPMSGE - Send Error Text To Screen

After a call to TIPMSGI, the program normally validates the data received from the terminal.

Programs can use the TIPMSGE subroutine call to:

- output an error (or informational) message
- indicate data fields that contain questionable values
- inform the terminal user that the input was not acceptable.

The TIPMSGE subroutine can accomplish two different objectives:

- Deliver error message text to the screen format.
- Identify data fields that are not acceptable to the program.

To deliver error message text, the program passes a parameter that defines a string of error text. The TIPMSGE subroutine retrieves from this location a number of bytes of character data the length of which corresponds to the sum of all "EEEEE" fields in the screen format definition.

Note: Although commonly referred to as an "error" message, the text could be a purely informational message, such as: "Searching File - Please Wait"

To highlight data fields that are in error, the program may move HIGH-VALUES (hexadecimal FF) to a field or fields in the MCS-DATA area before calling the TIPMSGE subroutine. The TIPMSGE subroutine uses the value in MCS-COUNT to determine how far to search the MCS-DATA area for any fields containing HIGH-VALUES. Normally this count has been set by the prior call to TIPMSGI.

TIPMSGE causes such flagged fields to "blink". If data fields in the screen format are "blinked", TIPMSGE leaves the cursor in the first character of the first field that is in "error". If no fields are blinked, the cursor remains in the cursor resting location defined for the screen format.

The TIPMSGE subroutine examines the field "MCS-FUNCTION". If this field contains the character "R", the TIPMSGE subroutine first "refreshes" all the data fields in the screen format. The refresh operation is accomplished by resending all of the FCC attributes to the fields (on terminals that use FCC). This effectively "unblinks" any fields that are already blinking **before** causing new fields to blink.

Set MCS-FUNCTION to "R" only when there are consecutive calls to TIPMSGE, so that the terminal operator won't have to guess which fields

are currently blinking (as opposed to those blinking due to prior calls to TIPMSGE).

Set MCS-FUNCTION to "M" to cause the terminal to beep.

Syntax:

```
CALL "TIPMSGE" USING    MCS
                        Text
                      [ fcc-mods ]
                      [ cursor-mods ]
```

Where:

MCS The MCS interface packet (previously described).

text The name of an elementary field or group item that contains the "error" text to be used to fill the type "EEEE" fields in the screen format.
The TIPMSGE subroutine copies characters from this field until it fills all error fields ("EEEE") in the screen format. For example, if the screen format contained two error fields: one of 20 characters, another of 70, TIPMSGE expects 90 characters (20+70) in this field.

fcc-mods
Optional table of two byte entries (two bytes per field) used in modification of FCC (Field Control Character) attributes of each data field.

See FCC Modifications on page 122 for details.

cursor-mods
Optional table of one-byte entries (one byte per field) uses in specifying the field where the cursor is to rest after the call to TIPMSGE.

See Cursor Positioning on page 126 for details.

Example:

```
05 ERROR-TEXT                PICTURE X(30).
 ...
 ...
PERFORM GET-INPUT-MSG.
 ...
IF SCREEN-ACCT-NUMBER < "A0000"
    MOVE HIGH-VALUES         TO S-ACCT-NUMB
    MOVE "INVALID ACCOUNT NUMBER"
                             TO ERROR-TEXT
    CALL "TIPMSGE" USING     MCS
                             ERROR-TEXT
```

```
END-IF
```

Additional Considerations:

- TIP sets MCS-COUNT to zero after a call to the TIPMSGE subroutine. It is not possible to avoid specifying FCC-MODS if the CURSOR-MODS parameter is specified.

TIPMSGEO - Define Deferred Error Text

Use the TIPMSGEO subroutine to "define" error message text to MCS. This error text is not acted upon immediately but is "remembered" by MCS and is appended to the end of the next output to the terminal by TIPMSGO.

TIPMSGEO does not actually send any data to the terminal; it is a mechanism that allows the program to issue a TIPMSGE in anticipation of a subsequent TIPMSGO. This technique saves the double transmission that often occurs when a program issues a TIPMSGO immediately followed by a TIPMSGE.

Syntax:

```
CALL "TIPMSGEO" USING   text
```

Where:

text The elementary or group item field name that contains the "error" text that is "remembered" during the next call to the TIPMSGO subroutine.

Make the TEXT area as large as the sum of the sizes of all error fields ("EEEE") in the screen format

Additional Considerations:

- MCS saves the data in the TEXT area and uses this text only on the next call to TIPMSGO. Whatever text is in the TEXT area when the TIPMSGO occurs is the data that is sent to the "E" fields.
- A common programming "trick" is to move error text to a work field whenever an error is detected in the input from the terminal. The paragraph that outputs data to the screen calls TIPMSGO and then conditionally calls TIPMSGE if the work field does not contain spaces. This results in two consecutive outputs to the terminal.
- Using TIPMSGEO instead effectively merges the two outputs into a single transmission.

TIPMSGI - Read Data from Screen Format

Online programs issue a call to the TIPMSGI subroutine to request terminal input. The use of TIPMSGI presumes that a TIP screen format

has already been used to send output to the terminal. This call is used at points in the online program where input is required from the terminal, for example, after a CALL to TIPMSGO or TIPMSGE.

Syntax:

```
CALL "TIPMSGI" USING MCS [ fld-ctrl ]
```

Where:

MCS

The MCS interface packet. Before issuing a call to TIPMSGI, your application must ensure that the MCS interface packet contains appropriate values in a number of the fields.

fld-ctrl

Optional second parameter. A cursor control array.

Each entry in the array is a single byte corresponding to a field of the MCS format and permits field level control.

Place an "X" in this field to have control return to the program when the field has changed and the cursor is leaving the field;

Place an "L" in this field to have control return to the program when the cursor is leaving the field (whether or not the field changed);

Place an "E" in this field to have control return to the program when the field is entered.

The MCS-STATUS status will be MCS-F-FIELD and the MCS-COUNT will be set to include the field just exited. The PIB-MCS-FIELD value will also be the field number just exited.

Before calling TIPMSGI, your application must ensure that the MCS interface packet contains appropriate values in these fields:

MCS-NAME

The program normally specifies the same screen format name in the field "MCS-NAME" for related output and input functions

MCS-FUNCTION

MCS-FUNCTION may be set to a space or the value "A". A space indicates no special input processing is required. Setting MCS-FUNCTION to "A" requests TIPMSGI to guarantee the input message retrieves ALL the unprotected data from the screen. When MCS-FUNCTION contains "A" and XMIT is pressed from a location that is not within or beyond the last unprotected data field, MCS

automatically places the cursor in the bottom right corner of the screen and issues an auto-transmit sequence to reread the entire screen.

This feature can almost double the transmission traffic from the terminal (first there is the partial transmit, then the full transmit) and therefore can be quite costly.

To minimize transmission traffic, a preferable technique is to compare MCS-COUNT (the count of actual data characters received) to MCS-SIZE (the maximum possible received on that transmission); if MCS-COUNT is less than MCS-SIZE, the program informs the user (via a call to TIPMSGE) that XMIT was pressed at the wrong screen location; then calls TIPMSGI again to allow the terminal user to press XMIT from the proper location.

Before a call to TIPMSGI, the program may also modify various fields defined in the PIB:

PIB-WAIT-TIME

The program may move a value to PIB-WAIT-TIME to specify the amount of time that TIPMSGI is to wait for input from the terminal. If PIB-WAIT-TIME is not altered (and presumably contains zero), the TIPMSGI subroutine does not impose a time limit on the arrival of the desired input message.

If an input message does not arrive within the number of seconds defined by the contents of PIB-WAIT-TIME, the call to TIPMSGI completes, and the resulting value of PIB-STATUS is "PIB-TIMED-OUT". Programs which place a limit on the arrival time of input messages, must be prepared to handle this situation.

If PIB-WAIT-TIME is set to a negative value then the value of system parameter TIMEOFF in the tipix.conf file will be used as the time to wait. Since TIMEOFF is specified in minutes and TIPMSGI expects a value in seconds TIP calculates the default PIB-WAIT-TIME as (TIMEOFF * 60).

This is useful when a site would like to implement a standard wait time in their programs. If this technique is used then the wait time is easily altered by adjusting the TIMEOFF system parameter. For programs that must operate on both TIP and TIP/30 the value supplied (to request the default wait time) must be -1.

For more details, see the description of the PIB-WAIT-TIME field in the PCS section of this manual.

PIB-LOCK-INDICATOR

The program may choose to move "H" to the field PIB-

LOCK-INDICATOR to coerce the TIP File Control System to hold any current record locks that have been acquired by the program.

If the PIB-LOCK-INDICATOR is not set to "H", the file system releases all record locks acquired by the program that is calling TIPMSGI. This action is taken by the file system to prevent programs from locking records and waiting for an inordinate length of time for terminal input.

If the program chooses to hold record locks across a TIPMSGI call, the program should also move an appropriate value to PIB-WAIT-TIME to place an upper limit on the length of time that the record locks will be maintained.

Example:

```
05  FLD-CTRL.
    10  FIELDS                 PICTURE X
                                OCCURS 20.

    MOVE SPACES                 TO FLD-CTRL
    MOVE "X"                    TO FIELDS (2)
                                   FIELDS (4)
    CALL "TIPMSGI" USING        MCS
                                FLD-CTRL
    EVALUATE TRUE
        WHEN MCS-F-FIELD
         ... field 2 or 4 was just changed ...
        WHEN MCS-XMIT
         ... process complete screen ...
    END-EVALUATE
```

When the program issues a call to TIPMSGI, MCS waits for the next input message from the terminal. Unless the program has specified a maximum time to wait in the PIB-WAIT-TIME field in the PIB, the program does not return from the call to TIPMSGI until input is received from the terminal. The input may be via the **XMIT** key, the **MSG WAIT** key or a function key.

Upon returning from the call to TIPMSGI, the user program must interrogate the field MCS-STATUS to establish the type of input received.

If MCS-STATUS indicates MCS-XMIT (or MCS-GOOD), the unprotected data from the screen was extracted by MCS and placed in the appropriate fields within MCS-DATA.

Warning: No data is transferred from the device if a function key is pressed.

Error Conditions:

PIB-STATUS	Meaning
PIB-TIMED-OUT	There was no response within the time allowed.
PIB-MSG-AVAIL	The response is available. This is not an error.

- A program may not request two consecutive inputs from a terminal without some intervening output message. If a user program requests terminal input and does not satisfy this constraint, TIP causes the program to abort with the following reason code:

  INPUT REQUEST WHEN OUTPUT IS DUE

- If the program placed a maximum wait time value in the field PIB-WAIT-TIME, the PIB-STATUS is set to either PIB-TIMED-OUT or PIB-MSG-AVAIL after the call to TIPMSGI, depending on which of those two mutually exclusive events occurred.

TIPMSGO - Output Data to Screen Format

MCS provides the TIPMSGO subroutine to display a TIP screen format (with or without) accompanying data.

Syntax:

```
CALL "TIPMSGO" USING          MCS
                            [ FCC-MODS ]
                            [ CURSOR-MODS ]
```

Where:

MCS
The MCS interface packet.

FCC-MODS
Optional table of two-byte entries (two bytes per field) that are used to modify the FCC (field control character) attributes of selected data fields.

See FCC Modifications on page 122 for details.

CURSOR-MODS
Optional table of one byte entries (one byte per field) that specifies the field where the cursor is to rest after the call to TIPMSGO.

See Cursor Positioning on page 126 for details.

Since this subroutine call is normally the first interaction between the program and TIP MCS, the program *must* first correctly initialize various fields in the MCS packet:

MCS-NAME
: The program must supply the name of the screen format to display. MCS searches for the named format in various groups according to the setting of the keyword MCSEARCH= in the terminal user's definition record.

MCS-TERM
: This field may be set to the name of the desired output terminal. The default is the terminal that is running the program.

  This field need only be modified if the program wants to output the screen on a terminal other than the terminal running the program.

  Only screen OUTPUT may be redirected in this manner - terminal input must always occur at the terminal running the program.

  MSC-FUNCTION of M must be used with MCS-TERM if you intend to send a screen as an unsolicited message to a specified alternate terminal. For screen to be displayed MSG-WAIT must be pressed on the receiving terminal.

  If your intentions are to display the screen automatically on the specified alternate terminal then you should use the TIPFORK function. A TIP session must be started on both terminals and the alternate terminal must not be running any other transactions.

MCS-FUNCTION
: Before issuing a call to TIPMSGO, the program may specify one of a number of function codes in this field:

  space Transmit the entire screen format (both headings and data).

  D Transmit data only (not the heading information). When "D" is specified in MCS-FUNCTION, data fields that contain low values are not sent to the terminal - the program may use this technique to avoid resending unchanged data to the terminal, thereby reducing output transmission.

  M Send the output screen format as an unsolicited message (sends data and heading information).

  P Output screen format with a "print" code at the end of the output message - to transfer screen to auxiliary printer.

S Stop sending heading text when the available MCS-DATA is exhausted (as specified by the value in MCS-COUNT).

T Unsolicited and Print. The message is sent to the specified terminal as an unsolicited message. At the end of the message text the control code to cause a "print" operation is included. When the receiving user presses the MSG WAIT key, the message is displayed and printed on his AUX1 printer.

MCS-HOLD

Set this field to the value "L" to cause MCS to LOCK the terminal keyboard after the TIPMSGO is completed.

If a program wishes to send a series of outputs to the terminal, this setting may be used to lock the keyboard on all but the final output call.

A call to TIPMSGI, or a call to TIPMSGO with MCS-HOLD not set to "L" unlocks the keyboard. The contents of this field are not preserved - the program must insert the desired value before issuing a call to TIPMSGO.

MCS-FILLER

The program must specify which fill character to use: space, underscore or asterisk. If this field contains an invalid choice of character, an underscore is assumed

MCS-COUNT

The program must specify the number of bytes of data in the MCS-DATA area that are to be merged with the screen format. This value can range from zero - when the program has no data to output - to a maximum of the sum of all data fields in the screen format.

If the screen format was defined with "default data" , the default data will be displayed if either of the following is true:

11.1.1..1. the field is located beyond the end of the data supplied in MCS-DATA - according to the value of MCS-COUNT.

11.1.1..2. the field contains low-values.

If the program intends to output all of the data for a particular screen format, a popular technique is to place a large value in this field (for example, 9999). If new fields are later added to the screen format, the programmer does not need to remember to find and modify all references to the previous high count.

MCS-DATA

If the program has data that is to be output to the screen

format, the data is placed in the appropriate elementary fields in this group item before the CALL is issued.

Additional Considerations:

- When "D" is specified in MCS-FUNCTION (transmit data only), MCS assumes that the heading data is already displayed on the terminal and sends only the data, as specified by the value in the field MCS-COUNT.
- MCS only sends a data field if the corresponding area in MCS-DATA contains a value that is not LOW-VALUES (X'00'). The program can output selected fields, using MCS-FUNCTION="D"; setting those fields that are not to be sent to LOW-VALUES.

Error Conditions:

- If the screen format that is named in the field MCS-NAME cannot be located, (a spelling error?), the program receives PIB-NOT-FOUND error status and the terminal screen is erased. The following message is displayed on the terminal:

```
<<<< UNDEFINED SCREEN FORMAT REQUESTED >>>>
$TRANID$ requested UNDEF
```

Where:

$TRANID$
: is the transaction code of the program that issued the TIPMSGO CALL

UNDEF
: is the data that was found in the field MCS-NAME.

TIPMSGOV - Overlay Current Screen

The TIPMSGOV call displays an MCS screen format and overlays the current screen. TIPMSGOV takes exactly the same parameters as TIPMSGO. See *TIPMSGO — Output Data to Screen Format* on page 150. The new screen is positioned based on the values in PIB-ALT-MCS-ROW and PIB-ALT-MCS-COL and is boxed in. You may issue this call up to 15 times to produce a tiling effect on the terminal. Each TIPMSGOV request saves the previous contents of the screen.

When a call is issued to TIPMSGOV, the value in the field PIB-MCS-OVERLAY (programs can interrogate this field to determine how many of the maximum 15 overlays are displayed).

TIPMSGOV will return with a PIB-STATUS of PIB-OVERFLOW if the MCS internal stack overflows, that is, if too many screens have been overlaid.

To use this MCS routine from TIP/as your program must be defined as a TIP program rather than a TIP/30 program, and you need to link your program with TIPIXAPI32U.LIB rather than TIP30API32U.LIB.

Syntax:

```
CALL "TIPMSGOV" USING   MCS
                      [ FCC-MODS ]
                      [ CURSOR-MODS ]
```

Where:

MCS
The MCS interface packet previously described.

FCC-MODS
Optional table of two-byte entries (two bytes per field) that are used to modify the FCC (field control character) attributes of selected data fields.

See FCC Modifications on page 122 for details.

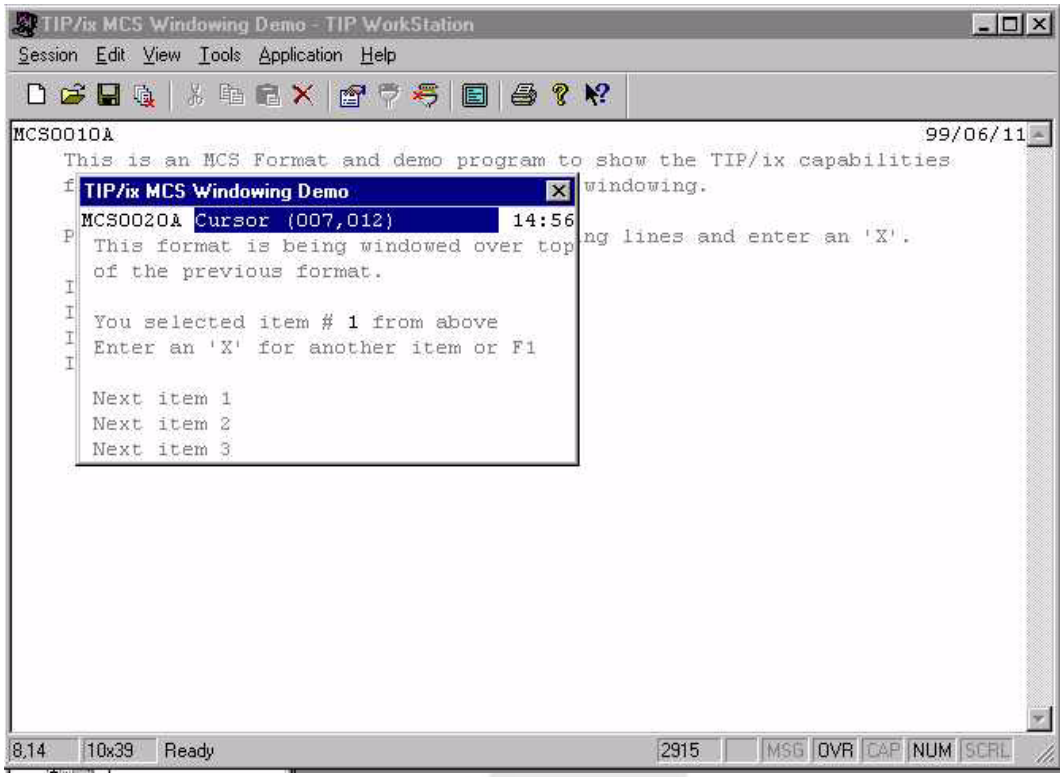
CURSOR-MODS
Optional table of one-byte entries (one byte per field) that specifies the field where the cursor is to rest after the call to TIPMSGO.

See Cursor Positioning on page 126 for details.

Example:

```
MOVE 5                  TO PIB-ALT-MCS-ROW
MOVE 10                 TO PIB-ALT-MCS-COL
CALL "TIPMSGOV" USING   MCS
CALL "TIPMSGI" USING    MCS
                        FLD-CTRL
EVALUATE TRUE
    WHEN MCS-FKEY6
        CALL "TIPMSGRS"
        ....
    WHEN MCS-FKEY8
        CALL "TIPERASE"
        ...
    WHEN ...
        ...
    END-EVALUATE
IF PIB-MCS-OVERLAY > 0
    CALL "TIPMSGRS"
END-IF
```

Example of TIPMSGOV displaying an overlay screen:



TIPMSGPR - Print Current Screen

TIPMSGPR has not been implemented in TIP Studio. For the functionality found with TIPMSGPR move a 'P' to the MCS-FUNCTION field prior to issuing a TIPMSGO call. This is exactly the way TIP/30 worked.The call TIPMSGPR prints an MCS screen format (headings and data) by creating a print line for each line of the screen format and passing that print line to the TIPPRINT subroutine.

The interface to TIPPRINT must already be OPEN; this call outputs as many lines as are represented by the screen format.

Syntax:

```
CALL "TIPMSGPR" USING   print-packet
                        MCS
                        tipprint-buffer.
```

print-packet

The first parameter is the printer definition packet that was used as the first parameter on the CALL TIPPRINT with an FCS-OPEN function. This packet is described in the documentation for TIPPRINT; essentially it contains the name of the printer that is to be used.

MCS

The second parameter is the MCS area for the current screen format.

tipprint-buffer

The third parameter is the printer buffer that was used as the fourth parameter on the CALL TIPPRINT with an FCS-OPEN function. This packet is described in the documentation for TIPPRINT; essentially it contains the name of the buffer that TIPPRINT uses.

Additional Considerations:

- This routine generates as many print lines as needed to represent the current screen format. This routine only issues calls to TIPPRINT with the FCS-PUT function code; it does not open or close the TIPPRINT interface. Other print lines (regardless of origin) can be output by the program before and after using this call.

TIPMSGRS - Pop the Current Screen

The call TIPMSGRS is the logical inverse of the call to TIPMSGOV. TIPMSGOV pushes an overlay screen on a stack; TIPMSGRS pops the overlay stack and restores the previous screen contents. Each time TIPMSGRS is called (and there is something to restore!), the value in PIB-MCS-OVERLAY is decremented by 1.

Syntax:

```
CALL "TIPMSGRS"
```

There are no parameters for this call.

Example:

```
MOVE 5                    TO PIB-ALT-MCS-ROW
MOVE 10                   TO PIB-ALT-MCS-COL
CALL "TIPMSGOV" USING     MCS
CALL "TIPMSGI" USING      MCS
                          FLD-CTRL
EVALUATE TRUE
    WHEN MCS-FKEY6
        CALL "TIPMSGRS"
          ....
    WHEN MCS-FKEY8
        CALL "TIPERASE"
          ...
    WHEN ...
          ...
END-EVALUATE
IF PIB-MCS-OVERLAY > 0
```

```
        CALL "TIPMSGRS"
    END-IF
```

TIPMSGRV - Force Full Screen Transmit

On Uniscope terminals, the data between HOME or the last start of entry character (>) and the cursor is transmitted to the host whenever the terminal operator presses **XMIT** (the character that is under the cursor is normally included too!).

The terminal operator may (by mistake) press **XMIT** part way through a screen thereby transmitting only a partial screen instead of the whole screen. This causes only some of the intended data to be transmitted to the host.

A TIP program may use the TIPMSGRV function to ensure that the entire screen is read when input is requested from the terminal. After a call to TIPMSGI, MCS sets the field MCS-COUNT to the number of characters of data received. The program can compare this value with the value in MCS-SIZE, which is the maximum number of bytes that *could have* been received on that transmission.

If MCS-COUNT is less than MCS-SIZE, the cursor was not in or beyond the last data field when **XMIT** was pressed.

The program can ignore this operator error by calling TIPMSGRV. The TIPMSGRV subroutine positions the cursor at the bottom right corner of the terminal (or at the end of a specific row) and causes an auto-transmit to occur (effectively transmitting the screen contents).

After the call to TIPMSGRV, all unprotected data from the screen is placed in the data area of the MCS packet - the program must not call TIPMSGI — the TIPMSGRV subroutine repeats the call to TIPMSGI after forcing the cursor to the appropriate location and causing an auto transmit.

Syntax:

```
CALL "TIPMSGRV" USING   MCS
                      [ row ]
```

Where:

MCS

The MCS interface packet previously described.

row

Optional binary halfword field (PIC 9(2) BINARY) that specifies the screen row number where the cursor is placed before the auto-transmit.

For example, specify a row number of 12 to cause TIPMSGRV to position the cursor in the last column of row 12 before issuing the auto-transmit code.

If this parameter is omitted or the value is out of range, the cursor is placed at the end of the last row of the terminal.

TIPTITLE - Display Title

The call TIPTITLE displays a title on the first display line of your screen. The title text is automatically centered.

To use this MCS routine from TIP/as your program must be defined as a TIP program rather than a TIP/30 program, and you need to link your program with TIPIXAPI32U.LIB rather than TIP30API32U.LIB.

Syntax:

```
CALL "TIPTITLE"
```

Example:

```
01  ATITLE                    PICTURE X(80)
    VALUE "TIP MCS Windowing Demo".
 ...
    CALL "TIPTITLE" USING   ATITLE
```

Additional Considerations:

- TIPTITLE will always display a title on line one of the display. If there is anything already on line one, TIPTITLE will overlay it. A subsequent call to TIPMSGO is adjusted down 1 row to accommodate the title line. A call to TIPERASE cancels the effect of the title

FCC Modifications

The attributes of data fields in a screen format are specified in the screen format definition. There are situations, however, when the program needs to modify the attributes of a field in a screen format while the screen format is in use.

Using an override mechanism of MCS the program can dynamically alter the attributes of a field — on calls to TIPMSGE and TIPMSGO.

This facility is available only on terminals that support the Field Control Character (FCC) method of establishing field attributes.

FCC modifications are specified as a table of two-byte entries that MCS uses to modify the attributes of the field(s) on the terminal. For additional information see the Unisys publication *UTS-400 Programmer Reference (UP-8359) - FCC Sequence from Host Processor*.

Each table entry consists of two characters that represent the "m" and "n" characters used in the construction of the FCC sequence for the field corresponding to the table entry (two bytes per field).

The field characteristics depend on the setting of the characters:

Character	Description
Space	Set either character to this value to avoid modifying the FCC attributes of the corresponding field.
X'00'	Low values (binary zeroes) may be used in the same way as a space (see description of "space" above).
*	Set either character to an asterisk to make the cursor rest in the corresponding data field when the data is sent to the terminal.
.U	Set the two bytes to this value to unprotect the field while leaving the other characteristics unchanged.
.P	Set the two bytes to this value to protect the field, while leaving the other characteristics unchanged.
.B	Set the two bytes to this value to blink the field, while leaving the other characteristics unchanged.

TC-FCC copybook

Include the supplied COBOL copybook (TIP/TC-FCC) in the program (in the WORKING-STORAGE SECTION) to simplify selection of the desired "m" and "n" characters.

```
*
* TIP/30 - FCC MODIFICATION EQUATES
*
* FOLLOWING VALUES ARE USED FOR THE FCC "M" FIELD
*
05  FCC-M-TAB-NRM-CHG          PICTURE X VALUE "0".
05  FCC-M-TAB-OFF-CHG          PICTURE X VALUE "1".
05  FCC-M-TAB-LOW-CHG          PICTURE X VALUE "2".
05  FCC-M-TAB-BLK-CHG          PICTURE X VALUE "3".
05  FCC-M-TAB-NRM              PICTURE X VALUE "4".
05  FCC-M-TAB-OFF              PICTURE X VALUE "5".
05  FCC-M-TAB-LOW              PICTURE X VALUE "6".
05  FCC-M-TAB-BLK              PICTURE X VALUE "7".
05  FCC-M-NRM-CHG              PICTURE X VALUE "8".
05  FCC-M-OFF-CHG              PICTURE X VALUE "9".
05  FCC-M-LOW-CHG              PICTURE X VALUE ":".
05  FCC-M-BLK-CHG              PICTURE X VALUE ";".
05  FCC-M-NRM                  PICTURE X VALUE "<".
05  FCC-M-OFF                  PICTURE X VALUE "=".
05  FCC-M-LOW                  PICTURE X VALUE ">".
05  FCC-M-BLK                  PICTURE X VALUE "?".
*
```

```
*** FOLLOWING VALUES ARE USED FOR THE FCC "N" FIELD
*
05  FCC-N-ANY                    PICTURE X VALUE "0".
05  FCC-N-ALPHA                  PICTURE X VALUE "1".
05  FCC-N-NUMERIC                PICTURE X VALUE "2".
05  FCC-N-PROTECT                PICTURE X VALUE "3".
05  FCC-N-ANY-RIGHT              PICTURE X VALUE "4".
05  FCC-N-ALPHA-RIGHT            PICTURE X VALUE "5".
05  FCC-N-NUMERIC-RIGHT          PICTURE X VALUE "6".
*
* A VALUE OF SPACE IN EITHER THE M OR N FIELD IMPLIES
* NO MODIFICATION DESIRED FOR THOSE ATTRIBUTES
*
* THESE VALUES ARE USED TO CHANGE PROTECTION
*
05  FCC-PROTECT                  PICTURE XX VALUE ".P".
05  FCC-UNPROTECT                PICTURE XX VALUE ".U".
*
* THESE VALUES ARE USED TO CHANGE INTENSITY
*
05  FCC-SHADED                   PICTURE XX VALUE ".S".
05  FCC-OFF                      PICTURE XX VALUE ".O".
05  FCC-NORMAL                   PICTURE XX VALUE ".N".
05  FCC-LOW                      PICTURE XX VALUE ".L".
05  FCC-BLINK                    PICTURE XX VALUE ".B".
05  FCC-REVERSE                  PICTURE XX VALUE ".R".
05  FCC-FLASHING                 PICTURE XX VALUE ".F".
05  FCC-GROTESQUE                PICTURE XX VALUE ".G".
05  FCC-HIDEOUS                  PICTURE XX VALUE ".H".
05  FCC-COLOR-10                 PICTURE XX VALUE ".0".
05  FCC-COLOR-11                 PICTURE XX VALUE ".1".
05  FCC-COLOR-12                 PICTURE XX VALUE ".2".
05  FCC-COLOR-13                 PICTURE XX VALUE ".3".
05  FCC-COLOR-14                 PICTURE XX VALUE ".4".
05  FCC-COLOR-15                 PICTURE XX VALUE ".5".
05  FCC-COLOR-16                 PICTURE XX VALUE ".6".
*
* THESE VALUES ARE USED TO CHANGE INTENSITY AND ADD TABS
*
05  FCC-SHADED-TAB               PICTURE XX VALUE "#S".
05  FCC-OFF-TAB                  PICTURE XX VALUE "#O".
05  FCC-NORMAL-TAB               PICTURE XX VALUE "#N".
05  FCC-LOW-TAB                  PICTURE XX VALUE "#L".
05  FCC-BLINK-TAB                PICTURE XX VALUE "#B".
05  FCC-REVERSE-TAB              PICTURE XX VALUE "#R".
05  FCC-FLASHING-TAB             PICTURE XX VALUE "#F".
05  FCC-GROTESQUE-TAB            PICTURE XX VALUE "#G".
05  FCC-HIDEOUS-TAB              PICTURE XX VALUE "#H".
05  FCC-COLOR-10-TAB             PICTURE XX VALUE "#0".
05  FCC-COLOR-11-TAB             PICTURE XX VALUE "#1".
05  FCC-COLOR-12-TAB             PICTURE XX VALUE "#2".
05  FCC-COLOR-13-TAB             PICTURE XX VALUE "#3".
05  FCC-COLOR-14-TAB             PICTURE XX VALUE "#4".
05  FCC-COLOR-15-TAB             PICTURE XX VALUE "#5".
05  FCC-COLOR-16-TAB             PICTURE XX VALUE "#6".
*
* THESE VALUES ARE USED TO CHANGE INTENSITY AND PROTECT
```

```
*
05  FCC-SHADED-PROT             PICTURE XX VALUE "!S".
05  FCC-OFF-PROT                PICTURE XX VALUE "!O".
05  FCC-NORMAL-PROT             PICTURE XX VALUE "!N".
05  FCC-LOW-PROT                PICTURE XX VALUE "!L".
05  FCC-BLINK-PROT              PICTURE XX VALUE "!B".
05  FCC-REVERSE-PROT            PICTURE XX VALUE "!R".
05  FCC-FLASHING-PROT           PICTURE XX VALUE "!F".
05  FCC-GROTESQUE-PROT          PICTURE XX VALUE "!G".
05  FCC-HIDEOUS-PROT            PICTURE XX VALUE "!H".
05  FCC-COLOR-10-PROT           PICTURE XX VALUE "!0".
05  FCC-COLOR-11-PROT           PICTURE XX VALUE "!1".
05  FCC-COLOR-12-PROT           PICTURE XX VALUE "!2".
05  FCC-COLOR-13-PROT           PICTURE XX VALUE "!3".
05  FCC-COLOR-14-PROT           PICTURE XX VALUE "!4".
05  FCC-COLOR-15-PROT           PICTURE XX VALUE "!5".
05  FCC-COLOR-16-PROT           PICTURE XX VALUE "!6".
```

Example:

Assume that the screen format has three fields: *name*, *address*, and *credit limit*.

```
05  SCREEN-NAME                 PICTURE X(40).
05  SCREEN-ADDR                 PICTURE X(40).
05  SCREEN-CRLIMIT              PICTURE S9(5)V99.
```

Also assume that an FCC-MODS table is set up in the program's WORK area to build the modifications. Although the table can be specified as an array (that is indexed or subscripted), the following method is preferable because fields can be added or removed from the screen format without major maintenance work (since the FCC modification entries are referenced by name rather than absolute position in the table).

```
05  FCC-MODS.
    10  FCC-MOD-NAME            PICTURE X(2).
    10  FCC-MOD-ADDR            PICTURE X(2).
    10  FCC-MOD-CRLIMIT         PICTURE X(2).
```

To protect the credit limit in the program (presuming that the field is defined by the screen format to be unprotected) the following statements are required:

```
    MOVE SPACES                  TO FCC-MODS.
    MOVE ".P"                    TO FCC-MOD-CRLIMIT
         ...
    CALL "TIPMSGO" USING         MCS
                                 FCC-MODS
```

In this example, the COBOL coding is relatively simple because the literal is exactly two bytes long and conveniently matches the receiving field. Many times, however, it is necessary to construct a two byte "m" and "n" sequence from the entries provided in the copybook TIP/TC-FCC.

COBOL provides a STRING verb to facilitate this sort of operation:

```
STRING FCC-M-TAB-BLK FCC-N-NUMERIC
DELIMITED BY SIZE
INTO FCC-MOD-CRLIMIT
```

The statement shown above concatenates the two named fields from the copybook (in that order) to create a two-byte value that is then placed in the field FCC-MOD-CRLIMIT. The specification FCC-M-TAB-BLK indicates that a tab is to be set (-TAB) and that the field is to blink (-BLK). The specification FCC-N-NUMERIC indicates that the field is to have the numeric attribute forced on.

Using the STRING verb eliminates the need to define each FCC MOD entry as a group item with two subordinate single byte elementary items.

Additional Considerations:

- It is crucial that there are exactly two bytes per field in the FCC modification table - use the COBOL command of the MSGAR utility transaction to verify the number of data fields in the screen format.

Cursor Positioning

The program may wish to use the FCC-MODS parameter to alter the attributes of a field (see previous section) **and** to force the cursor into a field that has an FCC mod specified. Since the table entry cannot simultaneously hold the FCC modification and the asterisk character, the program must use the CURSOR-MODS parameter (when calling TIPMSGE and TIPMSGO) in such a situation.

The CURSOR-MODS parameter specifies a table of one-byte entries (one byte per field in the screen format).

The program may place an asterisk (*) in the appropriate byte to force the cursor to rest in the corresponding field in the screen format. This facility is normally required only when the program needs to use FCC-MODS to alter a field's attributes and also needs to force the cursor into the same field.

Additional Considerations:

- It is crucial to have exactly one byte per field in the CURSOR modification table. Use the COBOL command of the MSGAR utility transaction to verify the number of data fields in the screen format.

Context Sensitive Help

You may enter help text into the TIP "TIPMCS" file using either **tfd** or **msgar** utility programs. The **msgar** commands **HI**mport **HX**port and

HUpdate are available for manipulating help information (see the documentation for **msgar** in the *TIP Utilities* manual.)

Help text is comprised of a series of lines of ASCII text and is given a name. TFD associates the help text name with any given data field in an MCS format or the format itself. An end user may, at any time, press **Ctrl-f h** to cause TIP to display the help text associated with the field where the cursor is resting. Help text usage is application independent - the application is not affected by its use.

Once TIP has displayed the help text window, the end user may use the cursor control keys to scan through the help text. TIP will highlight the currently selected line of help text. If the user presses **ENTER** the line of highlighted data is entered into the field on the MCS format. If the end user presses **ESC** nothing is entered. In both cases TIP removes the help text window from the terminal.

Help Text Definition

You may create help text in an ASCII text file using any text editor. A single text file may contain one or more help text definitions. The file is imported and each named help becomes a separate record in "TIPMCS". Use the command **MSGAR HI MYHELP**.

The **MSGAR** utility reads the text file looking for a line that begins with **HELP=**. Following the equal sign is the name of the help text, some comments enclosed in double quotes and then a period. Subsequent keywords help to define the display characteristics of the help text.

The first few lines beginning with an exclamation mark are treated as heading lines. Heading lines are displayed but the end user cannot select them (they are protected). The first line or lines may contain some keywords that define additional information.

End users may select any line following the heading lines (if any). Lines following a selectable line, and beginning with an # define additional information that appears in a footing area when that line is selected.

The entire help text definition is terminated with a line, which begins with a period**.**

Applicable keywords are:

LINES=
Number of selectable lines to display at once. If not specified, this is calculated by TIP.

HEADCHAR=
Defines a character other than "!" that identifies the heading lines.

CMTCHAR=
Defines a character other than "#" that identifies the footing information lines.

POS=(row,col)
Defines the exact position where the help text is to be displayed. If omitted, this is calculated to be close to the field cursor location.

STYLE=MENU
Declares that the information is to be displayed in a "menu bar" presentation style instead of a "pick from a list" style.

Example of Help Text:

```
HELP=DEMO1 "This is a sample of Help".
LINES=4,CMTCHAR=*
!This is a demo of the
!TIP help facility
!Select one of the following
Apple
*Delicious and good for you
Banana
*From down south
Orange
*From Florida
Potato
*From P.E.I.
Carrot
*Good for your eyes
Prunes
*Good for your digestion
```

In this example only four lines of selectable text will be displayed at once, but the end user may scroll through off screen data. The help processor will also quick scan to a line when you enter a letter on the keyboard. If you entered the letter "p" then the help text processor will scan to the next line that contains an uppercase "P".

The displayed help text window for the above example would look like the following:

```
*----------------------------*
|This is a demo of the       |
|TIP help facility           |
|Select one of the following |
*----------------------------*
|Apple  |                    |
|Banana |                    |
|Orange |                    |
|Potato |                   v|
```

```
*-----------------------------*
|Delicious and good for you   |
*-----------------------------*
```

The small narrow box on the right with the **v** is indicating that there are more lines off-screen below. You can move to the off screen data by using the down arrow key. You could quickly go to the Carrot line by pressing the letter **C**.

The "menu bar" presentation style would look like the following:

```
Apple Banana Orange Potato Carrot Prunes
 Delicious and good for you
```

TSTWIN - Sample TIP Program

The following sample program is provided with TIP in binary and source code format. This simple program illustrates how to use the new windowing features of TIP, which are supported by the MCS facility. The new TIP calls in this program are: TIPMENU, TIPLIST, TIPASK, TIPFORKW, TIPWINAP, TIPASKYN, TIPMSGOV, and TIPMSGRV.

The TIP CALLs that are used in this program are explained elsewhere in this manual.

```
IDENTIFICATION DIVISION.
 PROGRAM-ID. TSTWIN.
 *-------------------------------------------------------------+
 *
 * T I P / i x S A M P L E P R O G R A M *
 *
 *-------------------------------------------------------------+
 *
 * This is a sample program to illustrate how *
 * to use the MCS windowing features of TIP. *
 *
 *-------------------------------------------------------------+
 ENVIRONMENT DIVISION.
 CONFIGURATION SECTION.
 DATA DIVISION.
WORKING-STORAGE SECTION.
01 IN-HELP1 PICTURE X(8) VALUE "1235".
 01 IN-TEXT1.
 05 FILLER PIC X(35) VALUE "Beer ".
 05 FILLER PIC X(35) VALUE "Cardhu ".
 05 FILLER PIC X(35) VALUE "Rum ".
 05 FILLER PIC X(35) VALUE "Imported wine".
 05 FILLER PIC X(35) VALUE "Old sailor ".
 05 FILLER PIC X(35) VALUE "eXport beer ".
 05 FILLER PIC X(35) VALUE "iMported beer".
 05 FILLER PIC X(35) VALUE "Cognac VSOP ".
 05 FILLER PIC X(35) VALUE "Scotch ".
 05 FILLER PIC X(35) VALUE "Single malt ".
```

```
 05 FILLER PIC X(35) VALUE "12 year old ".
 05 FILLER PIC X(35) VALUE "How old am i?".
01 IN-HELP2 PICTURE X(8) VALUE "1330".
 01 IN-TEXT2.
 05 FILLER PIC X(30) VALUE "!What is your sport?".
 05 FILLER PIC X(30) VALUE " Golf ".
 05 FILLER PIC X(30) VALUE "# Relaxing? ".
 05 FILLER PIC X(30) VALUE " Baseball ".
 05 FILLER PIC X(30) VALUE "# Go Blue jays ".
 05 FILLER PIC X(30) VALUE " Racquet ball ".
 05 FILLER PIC X(30) VALUE "# Smack it hard".
 05 FILLER PIC X(30) VALUE " Mud wrestling ".
 05 FILLER PIC X(30) VALUE "# Male or female".
 05 FILLER PIC X(30) VALUE " Skinning bears".
 05 FILLER PIC X(30) VALUE "# Sandy's favorite".
 05 FILLER PIC X(30) VALUE " Quaffing beer ".
 05 FILLER PIC X(30) VALUE "# Barry's favorite".
01 ATITLE PIC X(80) VALUE "TIP MCS Windowing Demo".
01 MENU1.
 05 FILLER PIC X(10) VALUE "Display ".
 05 FILLER PIC X(10) VALUE "Update ".
 05 FILLER PIC X(10) VALUE "Cancel ".
 05 FILLER PIC X(10) VALUE "End ".
 05 FILLER PIC X(10) VALUE "Quit ".
 05 FILLER PIC X(10) VALUE "Home ".
 05 FILLER PIC X(20) VALUE " Pick one and Enter".
 01 FILLER REDEFINES MENU1.
 05 MENU-ITEM OCCURS 8 PIC X(10).
 77 ROW10 PICTURE 999 BINARY VALUE 10.
/
LINKAGE SECTION.
 01 PIB. COPY TC-PIB.
 01 MCS. COPY TC-MCS.
 *
 * LAYOUT OF THE MENU SELECTION DISPLAY
05 MENU-SELECT OCCURS 4 TIMES PICTURE X.
 *
 * LAYOUT OF OVERLAY DISPLAY
 02 OVERLAY-SCREEN REDEFINES MCS-DATA.
 05 MENU-POS 1PICTURE 9.
 05 OVER-SELECT OCCURS 3 TIMES PICTURE X.
01 WORK-AREA.
 05 ERROR-MESSAGE PICTURE X(30).
 05 ERROR-FLAG PICTURE X.
 88 FIELDS-IN-ERROR VALUE "E".
 05 FLASH-FLAG PICTURE X.
 88 FIRST-TIME VALUE "1".
 88 MULTI-ERROR VALUE "2".
 05 SAVE-MCS-COUNT PICTURE 9(4)
 COMPUTATIONAL-4.
 05 SAVE-MCS-FUNCTION PICTURE X.
 05 SAVE-LOCK-INDICATOR PICTURE X.
 05 FIELD-CONTROL.
 10 FIELD-X OCCURS 4 TIMES PICTURE X.
 05 LST-I PICTURE 9(4) BINARY.
 05 II PICTURE 9(4) BINARY.
 05 JJ PICTURE 9(4) BINARY.
```

```
 05 HELP-NAME PICTURE X(8).
 05 REPLY-TEXT PICTURE X(80).
 05 FILLER REDEFINES REPLY-TEXT.
 10 TXT PIC X OCCURS 80 TIMES.
 05 PROMPT-TEXT PICTURE X(80).
 05 FILLER REDEFINES PROMPT-TEXT.
 10 TXTX PIC X OCCURS 80 TIMES.
 01 CDA. COPY TC-CDA.
 /
PROCEDURE DIVISION USING PIB
 CDA
 MCS
 WORK-AREA.
INITIALIZATION.
 MOVE SPACES TO ERROR-MESSAGE.
MAIN-LOOP.
 IF CDA-PARAM (1) = " "
 CALL "TIPTITLE" USING ATITLE
 ELSE
 IF CDA-PARAM (1) NOT = "NO "
 CALL "TIPTITLE" USING CDA-TEXT
 END-IF
 END-IF
 CALL "TIPMENU" USING MENU1
 MOVE "MCS0010A" TO MCS-NAME
 MOVE SPACES TO MCS-DATA
 MOVE 0 TO MCS-COUNT
 * An 'X' indicates that control is wanted if the field changes
 MOVE ALL "X" TO FIELD-CONTROL
 PERFORM SEND-OUTPUT.
MAIN-INPUT.
MOVE " " TO REPLY-TEXT
 PERFORM GET-INPUT
 IF MCS-F-FIELD
 * One of the fields was typed into
 IF MENU-SELECT (MCS-COUNT) = "H"
 MOVE "HELPX2" TO HELP-NAME
 CALL "TIPLIST" USING HELP-NAME REPLY-TEXT
 PERFORM SHOW-REPLY
 GO TO MAIN-INPUT
 END-IF
 IF MENU-SELECT (MCS-COUNT) = "P"
 MOVE "Type something clever" TO PROMPT-TEXT
 MOVE "Hi" TO REPLY-TEXT
 CALL "TIPASK" USING REPLY-TEXT PROMPT-TEXT
 PERFORM SHOW-REPLY
 GO TO MAIN-INPUT
 END-IF
 IF MENU-SELECT (MCS-COUNT) = "W"
 MOVE "Enter Transaction for next Window"
 TO PROMPT-TEXT
 MOVE SPACES TO REPLY-TEXT
 CALL "TIPASK" USING REPLY-TEXT PROMPT-TEXT
 IF REPLY-TEXT NOT = SPACES
 MOVE SPACES TO PIB-TRID CDA PROMPT-TEXT
 PERFORM VARYING II FROM 1 BY 1
 UNTIL TXT(II) = " "
```

```
MOVE TXT(II) TO PIB-TRID(II:1)
END-PERFORM
PERFORM UNTIL TXT(II) NOT = " "
OR II > 60
ADD 1 TO II
END-PERFORM
PERFORM VARYING JJ FROM 1 BY 1
UNTIL II > 79
MOVE TXT(II) TO TXTX(JJ)
ADD 1 TO II
END-PERFORM
MOVE PROMPT-TEXT TO CDA
CALL "TIPFORKW"
END-IF
PERFORM SHOW-REPLY
GO TO MAIN-INPUT
END-IF
IF MENU-SELECT (MCS-COUNT) = "A"
MOVE "Enter Windows Application command line"
TO PROMPT-TEXT
MOVE SPACES TO REPLY-TEXT
CALL "TIPASK" USING REPLY-TEXT PROMPT-TEXT
IF REPLY-TEXT NOT = SPACES
CALL "TIPWINAP" USING REPLY-TEXT
END-IF
PERFORM SHOW-REPLY
GO TO MAIN-INPUT
END-IF
IF MENU-SELECT (MCS-COUNT) = "Q"
CALL "TIPASK" USING REPLY-TEXT
PERFORM SHOW-REPLY
GO TO MAIN-INPUT
END-IF
IF MENU-SELECT (MCS-COUNT) = "Y"
MOVE "Type something clever (Y/N)" TO PROMPT-TEXT
MOVE "Y" TO REPLY-TEXT
CALL "TIPASKYN" USING REPLY-TEXT PROMPT-TEXT
PERFORM SHOW-REPLY
GO TO MAIN-INPUT
END-IF
IF MENU-SELECT (MCS-COUNT) = "N"
MOVE "N" TO REPLY-TEXT
CALL "TIPASKYN" USING REPLY-TEXT
PERFORM SHOW-REPLY
GO TO MAIN-INPUT
END-IF
IF MENU-SELECT (MCS-COUNT) = "1"
CALL "TIPLIST" USING IN-HELP1 REPLY-TEXT IN-TEXT1
PERFORM SHOW-REPLY
GO TO MAIN-INPUT
END-IF
/
IF MENU-SELECT (MCS-COUNT) = "2"
CALL "TIPLIST" USING IN-HELP2 REPLY-TEXT IN-TEXT2
PERFORM SHOW-REPLY
GO TO MAIN-INPUT
END-IF
```

```
 IF MENU-SELECT (MCS-COUNT) = "T"
 MOVE 5 TO PIB-WAIT-TIME
 END-IF
 MOVE ALL "X" TO FIELD-CONTROL
 MOVE " " TO FIELD-X (1)
 * MOVE SPACES TO MCS-DATA
 MOVE MCS-COUNT TO MENU-POS
 MOVE "MCS0020A" TO MCS-NAME
 ADD 3 TO PIB-ALT-MCS-ROW
 ADD 6 TO PIB-ALT-MCS-COL
 IF PIB-ALT-MCS-COL > 38
 SUBTRACT 27 FROM PIB-ALT-MCS-COL
 END-IF
 IF PIB-ALT-MCS-ROW > 10
 SUBTRACT 7 FROM PIB-ALT-MCS-ROW
 END-IF
 EVALUATE TRUE
 WHEN PIB-MCS-OVERLAY > 13
 MOVE "MCS stack overflow" TO PROMPT-TEXT
 CALL "TIPERASE"
 CALL "ROLL" USING PROMPT-TEXT
 CALL "TIPRTN"
 WHEN OTHER
 MOVE "Cursor (rrr,ccc) " TO ERROR-MESSAGE
 MOVE PIB-CUR-MCS-ROW TO ERROR-MESSAGE (9:3)
 MOVE PIB-CUR-MCS-COL TO ERROR-MESSAGE (13:3)
 PERFORM SEND-OVERLAY
 END-EVALUATE
 GO TO MAIN-INPUT
 END-IF.
IF MCS-MSG-WAIT
 IF PIB-MCS-OVERLAY <= 0
 MOVE "TIP MCS window testing ending" TO PROMPT-TEXT
 CALL "TIPERASE"
 CALL "ROLL" USING PROMPT-TEXT
 CALL "TIPRTN"
 END-IF
 PERFORM POP-OVERLAY
 GO TO MAIN-INPUT
 END-IF
IF MCS-FKEY4
 MOVE "TIP MCS window testing ending" TO PROMPT-TEXT
 CALL "TIPERASE"
 CALL "ROLL" USING PROMPT-TEXT
 CALL "TIPRTN"
 END-IF
IF MCS-FKEY2
 CALL "TIPERASE"
 GO TO MAIN-LOOP
 END-IF
IF MCS-F-MENU
 MOVE "Menu Bar" TO REPLY-TEXT
 PERFORM SHOW-REPLY
 GO TO MAIN-INPUT
 END-IF.
IF MCS-FKEY5
 CALL "TIPMSGRV" USING MCS
```

```
 MOVE "Count " TO ERROR-MESSAGE
 MOVE MCS-COUNT TO ERROR-MESSAGE (12:5)
 CALL "TIPMSGE" USING MCS ERROR-MESSAGE
 MOVE SPACES TO ERROR-MESSAGE
 GO TO MAIN-INPUT
 END-IF.
IF MCS-FKEY6
 CALL "TIPMSGRV" USING MCS ROW10
 MOVE "Count Row" TO ERROR-MESSAGE
 MOVE MCS-COUNT TO ERROR-MESSAGE (12:5)
 CALL "TIPMSGE" USING MCS ERROR-MESSAGE
 MOVE SPACES TO ERROR-MESSAGE
 GO TO MAIN-INPUT
 END-IF.
IF PIB-TIMED-OUT
 MOVE SPACES TO MCS-DATA
 IF PIB-MCS-OVERLAY > 0
 PERFORM POP-OVERLAY
 END-IF
 MOVE "KeyBoard timed out" TO ERROR-MESSAGE
 PERFORM SEND-OUTPUT
 GO TO MAIN-INPUT
 END-IF.
IF PIB-MCS-OVERLAY > 0
 PERFORM POP-OVERLAY
 IF PIB-MCS-OVERLAY > 0
 MOVE SPACES TO MCS-DATA
 MOVE 0 TO MCS-COUNT
 MOVE "Window #" TO ERROR-MESSAGE
 MOVE PIB-MCS-OVERLAY TO ERROR-MESSAGE (9:3)
 PERFORM SEND-OUTPUT
 GO TO MAIN-INPUT
 END-IF
 GO TO MAIN-LOOP
 END-IF.
 MOVE "Program ending" TO PROMPT-TEXT
 MOVE PIB-ALT-MCS-ROW TO PROMPT-TEXT (30:2)
 CALL "TIPERASE"
 CALL "ROLL" USING PROMPT-TEXT
 CALL "TIPRTN".
/ **** TERMINAL INPUT/OUTPUT CALLS ****
 *-------------------------------------------------------------***
 * SEND ERROR MESSAGE: ***
 * IF NOT FIRST ERROR MESSAGE, THEN RE-FRESH FCC"S. ***
 *-------------------------------------------------------------***
 SEND-ERROR.
 IF NOT FIRST-TIME
 MOVE "R" TO MCS-FUNCTION.
 CALL "TIPMSGE" USING MCS ERROR-MESSAGE.
 MOVE SPACES TO ERROR-MESSAGE.
 MOVE "2" TO FLASH-FLAG.
 *-------------------------------------------------------------***
 * SEND THE OUTPUT SCREEN: ***
 * IF SAME SCREEN WAS USED BEFORE, THEN SEND DATA ONLY ***
 * IF ERROR MESSAGE NOT SPACES, THEN SEND ERROR AS WELL ***
 *-------------------------------------------------------------***
 SEND-OUTPUT.
```

```
MOVE MCS-COUNT TO SAVE-MCS-COUNT
MOVE MCS-FUNCTION TO SAVE-MCS-FUNCTION
IF MCS-NAME EQUAL TO PIB-LAST-MCS-NAME
MOVE "D" TO MCS-FUNCTION
ELSE
MOVE " " TO MCS-FUNCTION
END-IF
IF ERROR-MESSAGE NOT EQUAL SPACES
CALL "TIPMSGEO" USING ERROR-MESSAGE
END-IF
CALL "TIPMSGO" USING MCS
MOVE "1" TO FLASH-FLAG
MOVE SPACES TO ERROR-MESSAGE
MOVE SAVE-MCS-FUNCTION TO MCS-FUNCTION.
*-------------------------------------------------------------***
* SEND THE OVERLAY SCREEN: ***
* IF ERROR MESSAGE NOT SPACES, THEN SEND ERROR AS WELL ***
*-------------------------------------------------------------***
SEND-OVERLAY.
MOVE MCS-COUNT TO SAVE-MCS-COUNT
MOVE MCS-FUNCTION TO SAVE-MCS-FUNCTION
MOVE " " TO MCS-FUNCTION
IF ERROR-MESSAGE NOT EQUAL SPACES
CALL "TIPMSGEO" USING ERROR-MESSAGE
END-IF
CALL "TIPMSGOV" USING MCS
MOVE "1" TO FLASH-FLAG
MOVE SPACES TO ERROR-MESSAGE
MOVE SAVE-MCS-FUNCTION TO MCS-FUNCTION.
*-------------------------------------------------------------***
* POP the current overlay screen off and restore original ***
*-------------------------------------------------------------***
POP-OVERLAY.
CALL "TIPMSGRS".
MOVE PIB-LAST-MCS-NAME TO MCS-NAME.
/
*-------------------------------------------------------------***
* WAIT FOR TERMINAL"S REPLY. ***
* Ask for Field level input returned ***
*-------------------------------------------------------------***
GET-INPUT.
MOVE PIB-LOCK-INDICATOR TO SAVE-LOCK-INDICATOR.
IF PIB-WAIT-TIME = 0
MOVE -1 TO PIB-WAIT-TIME.
CALL "TIPMSGI" USING MCS FIELD-CONTROL.
*-------------------------------------------------------------***
* Show REPLY in the Error message field ***
*-------------------------------------------------------------***
SHOW-REPLY.
MOVE "F" TO ERROR-MESSAGE
MOVE PIB-MCS-KEY TO ERROR-MESSAGE (2:1)
MOVE PIB-MCS-FIELD TO ERROR-MESSAGE (4:4)
MOVE REPLY-TEXT TO ERROR-MESSAGE (10:20)
MOVE SPACES TO MCS-DATA
PERFORM SEND-OUTPUT.
```

Line Oriented Terminal I/O

The subroutines described in this section provide terminal I/O handling capabilities that programs may use to interact with the terminal on a line-by-line basis. This mode of interaction is a more primitive level of control than that offered by the TIP Message Control System (MCS), which was discussed in the previous section.

Native Mode Program

A native mode TIP program may use these subroutines to facilitate direct control of terminal input and output in situations that require low volume interaction with the user.

For example:

- Continuation prompts ("Continue Yes/No")
- Simple data entry ("Enter an account number:").

Line oriented terminal I/O operations are similar to facilities provided by many of the popular programming languages available on personal computers (such as BASIC). As the name implies, input and output operations are restricted to applications where single line prompts and replies are sufficient.

Subroutine	Description
BREAK	Check for operator break (interrupt).
PARAM	Parameterize input from terminal (or a supplied string).
PROMPT	Issue prompt and call PARAM to process reply.
PROMPTX	Issue prompt and retrieve reply (up to 64 characters) without parameterization.
PROMPTX8	Issue prompt and retrieve reply (up to 72 characters) without parameterization.
ROLL	Roll terminal display up one line and output one line on bottom row.
ROLLPT	Set roll point (number of lines to freeze at top of screen) for ROLL subroutine.
TEXT	Read line of input from terminal (up to 64 characters) without parameterization.
TEXT80	Read line of input from terminal (up to 72 characters) without parameterization.

Function Key Input

When a function key or **MSG WAIT** is pressed, *absolutely no data is transmitted from the terminal*.

To allow programs to properly process function keys, TIP translates the function key notification into a string of four characters when input is solicited by calling the Line-oriented subroutines (PROMPT, BREAK etc).

The program receives four characters in the input area (the remainder of the area is cleared to spaces). The first two characters are always "F#".

The next two characters are digits representing the function key number, for example:

- a value of F#00 represents MSG WAIT
- a value of F#01 represents F1
- a value of F#02 represents F2

...and so on.

Some terminals may be configured via a hardware or software option to signal the host computer when the terminal is reset or powered on. This is called a "Power On Confidence" signal - or POC. The signal to the host (if such a signal is received) is translated by TIP into the pseudo function key **F23**.

BREAK - Check For Operator Break

The BREAK subroutine checks for input that is already available from the terminal. This subroutine is often called to check whether or not the terminal operator has pressed the **MSG WAIT** key, a function key or the **XMIT** key to interrupt continuous ROLLed.

If an input message is not available from the terminal, the BREAK subroutine clears the result area to spaces and returns control to the calling program.

If an input message is pending at the time the program calls BREAK, the BREAK subroutine reads the input and discards it. BREAK next prompts the terminal operator with a standard TIP "break message":

```
Continue?▶Yes ▶No
```

The cursor is left in the "Yes" field, since this subroutine is often used as a mechanism to temporarily pause an otherwise continuous stream of output messages.

When the terminal operator responds, the reply is parameterized into the area specified as the first parameter to the BREAK subroutine.

Syntax:

```
CALL "BREAK" USING      param-area
```

Where:

param- area

An area - PIC X(64) - that receives the reply to the continuation query if there was an unsolicited interruption by the terminal user. This area is interpreted as eight occurrences of PIC X(8) - see also PARAM Parameterize Data for more information.

See Function Key Input on page 137 for a description of how function keys are returned.

Warning: The programmer must be careful to avoid a classic programming blunder; namely, assuming that the absence of "N" in the first position of the reply implies YES. In fact, if a function key was pressed, the first character of the result will be "F" (see *Function Key Input*).

Furthermore, the terminal operator could transmit *anything* - the program should carefully examine the result field and decide whether or not the terminal operator has correctly "interrupted" whatever processing is taking place.

PARAM - Parameterize Data

This subroutine takes an input string and breaks it into as many as eight fields of up to eight bytes each.

The input string may be a field supplied by the program or the program may choose to have PARAM prompt the terminal user for up to 80 characters of input.

PARAM recognizes the following characters as a *single* delimiter between fields:

- comma
- slash
- single space
- multiple consecutive spaces
- equal sign

If an optional second parameter is supplied, it is assumed to be the name of a 72-byte data area to be parameterized; otherwise, input is solicited from the terminal.

If input is solicited from the terminal all communications characters (DICE codes and FCC sequences) are removed from the input data before parameterization is performed.

Each alphanumeric parameter is:

- translated to uppercase
- left justified
- space padded on the right to a maximum of eight characters.

Each strictly numeric parameter (a parameter which consists of digits only) is: right justified with leading zeros to a maximum of 8 characters.

See *Function Key Input* for a description of how function keys are returned when input is obtained from the terminal.

Syntax:

```
CALL "PARAM" USING      param-area
                      [ text-area ]
```

Where:

param-area
The name of a 64-byte area to receive the parameterized data.

text-area
Optional input to the PARAM subroutine.

TEXT-AREA is a 72-byte field that is parameterized. If this parameter is omitted, up to 80 characters of input are solicited from the terminal and parameterized into "PARAM-AREA".

Example:

```
05  PARAM-AREA.
    10  PARAM OCCURS 8 TIMES PIC X(8).
05  TEXT-AREA PIC X(64).
```

The following table illustrates various input strings and the appearance of the PARAM-AREA after a call to PARAM. Double quotes in the table are present only to clearly delimit the strings; trailing parameters are not shown (they are spaces in each case):

TEXT-AREA	PARAM-AREA
"DR. John Smith III"	DR. JOHN SMITH III
"TSPUPDT 123/x PRINT"	TSPUPDT 00000123

TEXT-AREA	PARAM-AREA
	X PRINT

PROMPT - Prompt Terminal for Reply

The PROMPT subroutine "rolls" the terminal display up one line and outputs a single line prompt on the bottom line of the terminal. PROMPT then calls the PARAM subroutine (already described) to wait for and parameterize the terminal operator's reply. The calling program may provide an optional parameter that is used as the text of the prompt or may permit PROMPT to construct default prompt text.

If the prompt text is not provided, PROMPT constructs a prompt that consists of the transaction name, followed by the current execution stack level, a question mark and an SOE (►) character:

```
msgar(1)?►
```

Syntax:

```
CALL "PROMPT" USING      param-area
                       [ prompt-str ]
```

Where:

param-area
The 64-byte area where the parameterized terminal input is placed. Alphabetic data will be translated to uppercase.

See Function Key Input on page 137 for a description of how function keys are returned.

prompt-str
Optional parameter; 80 character prompt string.

If this parameter is supplied, this string (up to the last non-blank character) is used as the prompt text.

The terminal operator has only the remainder of the line to enter the response to the prompt, since prompts are output on the last line of the terminal.

If the program supplies a prompt string, either the first or the last non-blank character may be specified as a backslash character ("\"). In either case, when the prompt is output to the terminal the backslash is replaced by a start of entry character (►)

PROMPT recognizes two "special" trailing strings:

- "\YES \NO " (exactly 11 characters, all upper case)
- "\NO \YES " (exactly 11 characters, all upper case)

In each of the above cases the PROMPT subroutine does the following:

- converts the 11 character strings into YES/NO or NO/YES style prompts
- replaces backslash characters with a start of entry character (►)
- translates the words YES and NO (uppercase!) into "Yes" and "No".

The *two* spaces after each word are replaced by a TAB stop and a single space and the cursor is placed (by default) after the first choice (*hence, the need for both variations!*).

Example:

```
WORKING-STORAGE SECTION.
77  QUESTION                PICTURE X(80)
    VALUE "Enter last name: \".
 ...
LINKAGE SECTION.
 ...
01  WORK-AREA.
    05  REPLY-AREA          PICTURE X(64).
 ...
PROCEDURE DIVISION USING     PIB
                             CDA
                             MCS
                             WORK-AREA.
 ...
    CALL "PROMPT" USING      REPLY-AREA
                             QUESTION
```

This type of prompt (and an example reply) appears as follows to the terminal operator:

```
Enter last name: ►Smith
```

In this instance, the field REPLY-AREA would contain "SMITH" followed by 59 spaces.

Additional Considerations:

- The PROMPT subroutine does not directly modify the prompt string provided by the program - PROMPT constructs the appropriate prompt string elsewhere (in a work area outside the domain of the calling program).

PROMPTX - Prompt for Text

PROMPTX is identical to the PROMPT subroutine described in the previous section, with one exception: PROMPTX does *not* parameterize the user's input!

Up to 64 bytes of the input message are stored in TEXT-AREA (without parameterization). PROMPTX performs uppercase alphabetic translation if uppercase translation is enabled for the transaction.

Syntax:

```
CALL "PROMPTX" USING    text-area
                      [ prompt-str ]
```

Where:

text-area
The 64-byte area where the un-parameterized terminal input is placed.

See Function Key Input on page 137 for a description of how function keys are returned

prompt-str
Optional parameter; 80-character prompt string.

If this parameter is supplied, this string (up to the last non-blank character) is used as a prompt.

Additional Considerations:

- See PROMPT — Prompt Terminal for Reply for additional details.

PROMPTX8 - Prompt for Text

PROMPTX8 is identical to the PROMPT subroutine described in a previous section, with the following two exceptions:

- PROMPTX8 does not parameterize the user's input.
- Up to 72 bytes of text from the input message are returned.

Although the receiving area must be defined as an 80-byte area, no more than 72 bytes will be returned. PROMPTX8 performs uppercase alphabetic translation if uppercase translation is enabled for the transaction.

Syntax:

```
CALL "PROMPTX8" USING   text-area
                      [ prompt-str ]
```

Where:

text-area
The 80-byte area where the un-parameterized terminal input is placed.

See Function Key Input on page 137 for a description of how function keys are returned

prompt-str

Optional parameter; 80 character prompt string.

If this parameter is supplied, this string (up to the last non-blank character) is used as a prompt.

Additional Considerations:

- See PROMPT — Prompt Terminal for Reply for details.

ROLL - Output Line & Roll Screen

ROLL scrolls the screen up one line and sends one 80-byte line from TEXT-AREA to the bottom line of the terminal. If a second parameter is specified, ROLL automatically uses that parameter to call the "BREAK" subroutine (see description earlier) after the line is output to the terminal.

If the optional second parameter is *not* specified, the program will not be notified if terminal input is pending after this call to "ROLL".

Syntax:

```
CALL "ROLL" USING        line
                       [ param-area ]
```

Where:

line

An 80-byte text area to be rolled on the terminal. This text is not translated into uppercase by the ROLL subroutine.

param-area

Optional field used to return result from call to the "BREAK" subroutine.

Example:

```
WORKING-STORAGE SECTION.
77  HDG-LINE                PICTURE X(80)
    VALUE " Amount Tax Total".
 ...
LINKAGE SECTION.
 ...
    05  DETL-LINE.
        10  DETL-AMT         PICTURE ZZZ,ZZ9.99.
        10  FILLER           PICTURE X(3).
        10  DETL-TAX         PICTURE Z,ZZ9.99.
        10  FILLER           PICTURE X(3).
        10  DETL-TOTAL       PICTURE Z,ZZZ,ZZ9.99.
        10  FILLER           PICTURE X(44).
 ...
    CALL "ROLL" USING        HDG-LINE
```

```
MOVE SPACES              TO DETL-LINE
MOVE 1000                TO DETL-AMT
MOVE 70                  TO DETL-TAX
MOVE 1070                TO DETL-TOTAL
CALL "ROLL" USING        DETL-LINE
```

Additional Considerations:

- An important alternative to the use of ROLL is to use TIPPRINT to output data to the terminal (special destination AUX0). See the description of the TIPPRINT subroutine in the File Control System (FCS) chapter of this reference manual.
- If ROLL is called by a background program initiated from an interactive user, the output is sent back to the originator.
- If ROLL is called by a background program initiated from system startup, the output is sent to the console.

ROLLPT - Set Terminal Roll Point

The subroutines ROLL, PROMPT, PROMPTX, PROMPTX8 and BREAK all roll the terminal display from bottom to top - the top lines roll off the screen as new lines appear on the bottom line. The default is to roll the entire display.

To retain a portion of the display on the screen, the program may call this subroutine to define a new "roll point".

Syntax:

```
CALL "ROLLPT" USING     roll-point
```

Where:

roll-point

The new roll point for the terminal.

This field is a binary halfword representing the number of lines to "freeze" at the top of the terminal.

If this field contains a value of zero, the terminal roll point is reset to the default state - no lines are frozen.

Example:

```
77  FREEZE-4                PICTURE S9(3)
                            BINARY VALUE 4.
...
    CALL "ROLLPT" USING     FREEZE-4
```

Using a value of four (as in the example above) causes the top four lines of the display to remain on the screen while the lower lines are rolled as necessary.

This technique may be used to freeze information (such as headings) on the screen while detail lines are ROLLed out underneath.

TEXT - Get One Line From Terminal

The TEXT subroutine retrieves an input message of up to 64 characters without parameterization. It is assumed that the program has already output whatever information is to be used as a prompt; otherwise the terminal operator may not know that input is required!

Alphabetic characters in the data are translated to uppercase.

Syntax:

```
CALL "TEXT" USING       text-area
```

Where:

text-area
The 64-byte area where the terminal input is to be placed.

See Function Key Input on page 137 for a description of how function keys are returned

Example:

```
05  TEXT-AREA            PICTURE X(64).
...
CALL "TEXT" USING        TEXT-AREA
```

TEXT80 - Get One Line From Terminal

TEXT80 is similar to the TEXT subroutine described in the previous section, except that up to 72 characters are retrieved and no parameterization is performed.

Alphabetic characters in the data are translated to uppercase.

Syntax:

```
CALL "TEXT80" USING     text-area
```

Where:

text-area
An 80-byte area where the terminal input is to be placed (without parameterization).

This field must be defined as 80 bytes, but no more than 72 bytes of terminal data are returned

See Function Key Input on page 137 for a description of how function keys are returned

Example:

```
05  TEXT-AREA              PICTURE X(80).
...
CALL "TEXT80" USING        TEXT-AREA
```

Direct Communications I/O

Direct Communications I/O

TIP provides facilities that an online program may use to directly interface with the host computer communications sub-system. This Direct Communication I/O interface is at a primitive level - that is, it is the responsibility of the program to generate the proper control information for the devices being manipulated.

With Direct Communications I/O, the program interfaces with the operating system communications control code via calls to a TIP subroutine named "TIPTERM".

The program is responsible for:

- Issuing messages
- Including the proper control codes to produce the desired effect at the terminal.

The program must also decode all input messages and, if necessary, be prepared to filter out any imbedded terminal-dependent control codes.

Direct communication I/O is provided for relatively rare instances where the program requires direct control of a terminal or a device. Applications should take advantage of the extensive display handling capabilities of the Message Control System (MCS) and use DCIO only when the requirements cannot otherwise be met.

Message Formats

All input and output messages must begin with a fixed-format message prefix.

COBOL copybooks TC-DCINP and TC-DCOUT define the message prefixes in a COBOL program.

Each copybook defines a standard message prefix, with one exception - an extra fullword added at the beginning of each prefix is used only by TIP.

TC-DCINP copybook

The layout of the input message area is provided by the COBOL copy book TC-DCINP:

```
COPY TC-DCINP.
*-------------------------------------------------------------*
* COPY ELEMENT FOR DCIO INPUT PACKET                          *
*-------------------------------------------------------------*
    05  DCIO-INP-PKT.
        10  FILLER              PICTURE 9(8) BINARY SYNC.
        10  FILLER              PICTURE X(20).
    05  DCIO-INP-PKTR REDEFINES DCIO-INP-PKT.
        10  DCIO-INP-STATUS     PICTURE X.
            88  DCIO-INP-GOOD       VALUE SPACE.
            88  DCIO-INP-NOT-AVAIL  VALUE "E".
            88  DCIO-INP-TRUNC      VALUE "E".
            88  DCIO-INP-FKEY       VALUE "F".
        10  FILLER              PICTURE X.
        10  DCIO-INP-BUF-LEN    PICTURE 9(4) BINARY SYNC.
        10  DCIO-INP-COUNT      PICTURE 9(4) BINARY SYNC.
        10  FILLER              PICTURE X(2).
        10  DCIO-INP-TERM-ID    PICTURE X(4).
        10  FILLER              PICTURE X(4).
        10  FILLER              PICTURE X(4).
        10  FILLER              PICTURE X(4).
    05  DCIO-INP-DATA.
*-------------------------------------------------------------*
* USER INPUT DATA LAYOUT FOLLOWS                               *
*-------------------------------------------------------------*
```

The following is a description of the fields that make up the DCIO input packet:

DCIO-INP-STATUS

This field is set to the appropriate status after calling TIPTERM with an input function.

Check this field to determine the status after calling TIPTERM with an input function.

DCIO-INP-BUF-LEN

This field must be set to the length of the data area that is reserved by the program after the group item "DCIO-INP-DATA". In effect, the byte count placed in this field represents the maximum size of the largest input message that the program is willing to read into that area.

If the input message from the terminal exceeds this value TIP writes a message to the console that indicates that "Truncated Input" occurred at the noted terminal.

DCIO-INP-COUNT

On return to a call to TIPTERM with a "read input" function, this field is set to the exact byte count of the input data that is placed in DCIO-OUT-DATA.

DCIO-INP-TERM-ID

On return from a call to TIPTERM, this field is set to the name of the terminal that generated the input. This is normally the same as the terminal that is running the program.

DCIO-INP-DATA

This hanging group item is the last line of the copybook. The intention is that the programmer codes (immediately following this) whatever elementary items are needed to allow the program to examine the input message(s).

TC-DCOUT

The layout of the output message area is provided by the COBOL copy book TC-DCOUT:

TC-DCOUT copybook

```
COPY TC-DCOUT.
*-------------------------------------------------------*
* copybook FOR DCIO OUTPUT PACKET                       *
*-------------------------------------------------------*
    05  DCIO-OUT-PKT.
        10  FILLER                  PICTURE 9(8)
                                    BINARY SYNC.
        10  FILLER                  PICTURE X(16).
    05  DCIO-OUT-PKTR REDEFINES DCIO-OUT-PKT.
        10  DCIO-OUT-STATUS         PICTURE X.
            88  DCIO-OUT-GOOD           VALUE SPACE.
            88  DCIO-OUT-LINE-DOWN      VALUE "B".
            88  DCIO-OUT-TERM-DOWN      VALUE "C".
            88  DCIO-OUT-INV-DEST       VALUE "D".
            88  DCIO-OUT-NO-BUF         VALUE "E".
            88  DCIO-OUT-IO-ERR         VALUE "F".
            88  DCIO-OUT-INVALID-LEN    VALUE "G".
            88  DCIO-OUT-NOT-CONN       VALUE "N".
            88  DCIO-OUT-AUX-DOWN       VALUE "0".
            88  DCIO-OUT-NOT-OP         VALUE "1".
            88  DCIO-OUT-PAPER-OUT      VALUE "2".
            88  DCIO-OUT-EOF            VALUE "3".
            88  DCIO-OUT-NO-RESP        VALUE "4".
        10  FILLER                  PICTURE X(7).
        10  DCIO-OUT-TERM-ID        PICTURE X(4).
        10  FILLER                  PICTURE X(4).
        10  DCIO-OUT-AUX-FLD        PICTURE 9(4)
                                    BINARY SYNC.
        10  DCIO-OUT-AUXR REDEFINES DCIO-OUT-AUX-FLD.
            15 DCIO-OUT-AUX-FUNC    PICTURE X.
            15 DCIO-OUT-AUX-DVC     PICTURE X.
        10  DCIO-OUT-COUNT          PICTURE 9(4)
                                    BINARY SYNC.
    05  DCIO-OUT-DATA.
*-------------------------------------------------------*
* USER OUTPUT DATA LAYOUT FOLLOWS                       *
```

```
*-------------------------------------------------------*
```

The following is a description of the fields that make up the DCIO output packet:

DCIO-OUT-STATUS
This field is set to the appropriate status when a call is issued to TIPTERM with an output function.

Check this field to determine the status after calling TIPTERM with an output function. The status in PIB-STATUS field may indicate PIB-GOOD — that status reflects the fact that TIP accepted the output. The result returned in DCIO-OUT-STATUS represents the output delivery status.

DCIO-OUT-TERM-ID
This field must be set to the name of the desired output terminal. If it is low-values or spaces, TIPTERM assumes that output is to be sent to the terminal where the program is running.

DCIO-OUT-AUX-FUNC
This field may be set to the desired auxiliary function code.

DCIO-OUT-AUX-DVC
This field may be set to the desired auxiliary device number. In general, a binary value is placed here (that is: X'01' for AUX1 and so on).

DCIO-OUT-COUNT
This field must be set to the byte count of the output message data. The count includes any control codes that are imbedded in the text of the message

DCIO-OUT-DATA
This hanging group item is the last line of the copybook. The intention is that the programmer codes (immediately following this line) the elementary items that are need for the program to construct the desired output message text.

TIPTERM Functions

User programs request direct terminal I/O services by calling the supplied subroutine TIPTERM with parameters indicating the desired function and, for most functions, the appropriate input or output message area.

Syntax:

```
CALL "TIPTERM" USING    func
                      [ msgarea ]
```

Where:

func
The desired TIPTERM function code. See the Copy book (TC-DCIO) described below. This parameter is required for all calls to TIPTERM.

msgarea
This optional parameter references either an input message packet (as defined by the Copy book TC-DCINP) or an output message packet (as defined by the Copy book TC-DCOUT). The choice depends on whether or not the associated TIPTERM function code implies reading or writing data.

COBOL programs use the supplied copy books TC-DCINP and TC-DCOUT to define the appropriate message areas. These copy books are shown and described in the preceding section.

TC-DCIO Copy Book

COBOL programs also should include the following supplied copy book in the WORKING-STORAGE section of the program to define the corresponding function codes for calls to TIPTERM:

```
* TC-DCIO COPY BOOK FOR TIP/30 DCIO TERMINAL CONTROL
************************************************************
* THE FOLLOWING ITEMS ARE TIPTERM FUNCTION CODES          *
************************************************************
    05  T-GET                    PICTURE X VALUE "G".
    05  T-PUT                    PICTURE X VALUE "P".
    05  T-TEST                   PICTURE X VALUE "W".
    05  T-UN                     PICTURE X VALUE "U".
```

T-GET - Get Input

The TIPTERM T-GET function reads input from the terminal.

Syntax:

```
MOVE ??                  TO PIB-WAIT-TIME
MOVE ??                  TO DCIO-INP-BUF-LEN
CALL "TIPTERM" USING     T-GET
                         DCIO-INP-PKT
```

Where:

T-GET
Function code defined in the TC-DCIO copy book.

DCIO-INP-PKT
The I/O input packet defined in the TC-DCINP copy book.

PIB-WAIT-TIME

If necessary, the program may move a value to the field PIB-WAIT-TIME to instruct the TIPTERM subroutine to wait for an input message for no longer than the specified number of seconds.

If a non-zero value is provided in PIB-WAIT-TIME the PIB-STATUS field is set to "PIB-MSG-AVAIL" or "PIB-TIMED-OUT" status as appropriate. See the description of the field PIB-WAIT-TIME.

DCIO-INP-BUF-LEN

Prior to the call to TIPTERM with the T-GET function, the input message packet must be initialized. In particular, the field DCIO-INP-BUF-LEN must be set to the maximum number of bytes that the program is willing to read from the terminal. This value cannot exceed the number of bytes of space reserved after the TC-DCINP copy book - recall that the last line of that copy book was a group item.

There are often occasions when the program moves a smaller value into the field to avoid reading excess data when a small input message is expected.

This situation is exactly the scenario for the "Input Truncated" warning that sometimes occurs. For example, the program outputs a simple prompt at the top of the terminal, expects a YES or NO reply and moves say, 80, to the input buffer length. The terminal operator keys in "NO" but places the cursor in the bottom right of the screen, presses <key t="XMIT"} and sends more than 1900 bytes in as an input message.

The result: the input is duly truncated and noted on the console by TIP and the program continues with no more than the requested 80 characters.

Additional Considerations:

- After control returns from the call, the input data (if any) is placed in the area DCIO-INP-DATA and the number of bytes actually received is placed in the field DCIO-INP-COUNT.
- The program must be aware that the data received likely contains DICE codes, FCC sequences, DATA characters etc. The program is responsible for filtering through all of the various bits of data that arrive.
- The program must take care to observe the byte count in the field provided for that purpose.

Error Conditions:

STATUS	Description

STATUS	Description
DCIO-INP-FKEY	Function key received. First byte of DCIO-INP-DATA is function key code (see note which follows).
DCIO-INP-GOOD	Input message received ok.
DCIO-INP-TRUNC	Truncated input. Input Message Area was smaller than input message.

- When a function key is pressed, the first byte of the DCIO-INP-DATA field contains the code representing the function key. These codes are exactly the same codes that are used by the Message Control System (MCS) to encode function keys in the field MCS-STATUS; namely, 0 (zero) means MSG WAIT and a value of 1 means F1 etc.
- **Warning**: NO DATA from the terminal screen is returned when MSG WAIT or a function key is pressed.

T-PUT - Output Message

The TIPTERM T-PUT function is used to output a message to a terminal. If required, the message data may include appropriate cursor positioning control codes (DICE) and possibly Field Control Characters (FCC).

Syntax:

```
MOVE "????"                    TO DCIO-OUT-DATA
MOVE ??                        TO DCIO-OUT-COUNT
MOVE SPACES                    TO DCIO-OUT-TERM-ID
CALL "TIPTERM"USING            T-PUT
                               DCIO-OUT-PKT
```

Where:

T-PUT
Function code defined in the TC-DCIO copy book.

DCIO-OUT-PKT
The output packet defined in the TC-DCOUT copy book.

DCIO-OUT-COUNT
The count of the number of bytes to be output must be moved into the field DCIO-OUT-COUNT before issuing this call. This count includes control characters such as DICE codes, FCC characters etc.

DCIO-OUT-DATA
The data bytes to be output must be moved into the group item DCIO-OUT-DATA before issuing this call. Normally this group item is defined and redefined to accommodate

many different types of output messages that the program might emit.

DCIO-OUT-TERM-ID

The field DCIO-OUT-TERM-ID may be set to the terminal name of the intended destination terminal. If this field is invalid, output is sent to the terminal where the program is running.

Example:

The following output message can be used to home the cursor and clear the screen:

```
MOVE ='1001010127D4'          TO DCIO-OUT-DATA
MOVE 6                        TO DCIO-OUT-COUNT
CALL "TIPTERM"USING           T-PUT
                              DCIO-OUT-PKT
```

The hexadecimal sequence ='10010101' is a DICE code sequence to position the cursor (1001) at row and column 1,1 (0101). The hex value 27 represents the code for ESC and hex D4 represents an "M". ESC-M, when sent to a UNISCOPE terminal, causes the terminal to perform the *CLEAR PROTECTED* function (clear all unprotected and protected data).

The programming manuals for the various terminal types contain this type of detailed information about controlling the terminal.

T-TEST - Test For Input

The TIPTERM function T-TEST allows the program to determine whether input has occurred. Pressing **MSG WAIT** or **XMIT** or some function key is sometimes used as a "break" signal for programs that generate continuing output. By periodically issuing a call to TIPTERM with the T-TEST function, the program can, in effect, "listen" for input from the terminal (and may choose to interpret the arrival of such an input message as a signal to stop output).

For example, a program that displays data from a file may generate many lines of output (by rolling the screen). By testing for input after every few lines of output the program can determine if input had been generated (if the operator presses **MSG WAIT** for example) and send a message to the operator to ask if continuation is desired.

Syntax:

```
MOVE ??                       TO DCIO-INP-BUF-LEN
CALL "TIPTERM" USING          T-TEST
                              DCIO-INP-PKT
```

Where:

T-TEST

Function code defined in the TC-DCIO copy book.

DCIO-INP-PKT

The I/O input packet defined in the TC-DCINP copy book.

DCIO-INP-BUF-LEN

Prior to the call to TIPTERM with the T-TEST function, the input message packet must be initialized. The field DCIO-INP-BUF-LEN must be set to the maximum number of bytes that the program is willing to read from the terminal. This value cannot exceed the number of bytes of space reserved after the TC-DCINP copy book - recall that the last line of that copy book is a group item.

See the description of this field in the discussion of the T-GET function code above.

Additional Considerations:

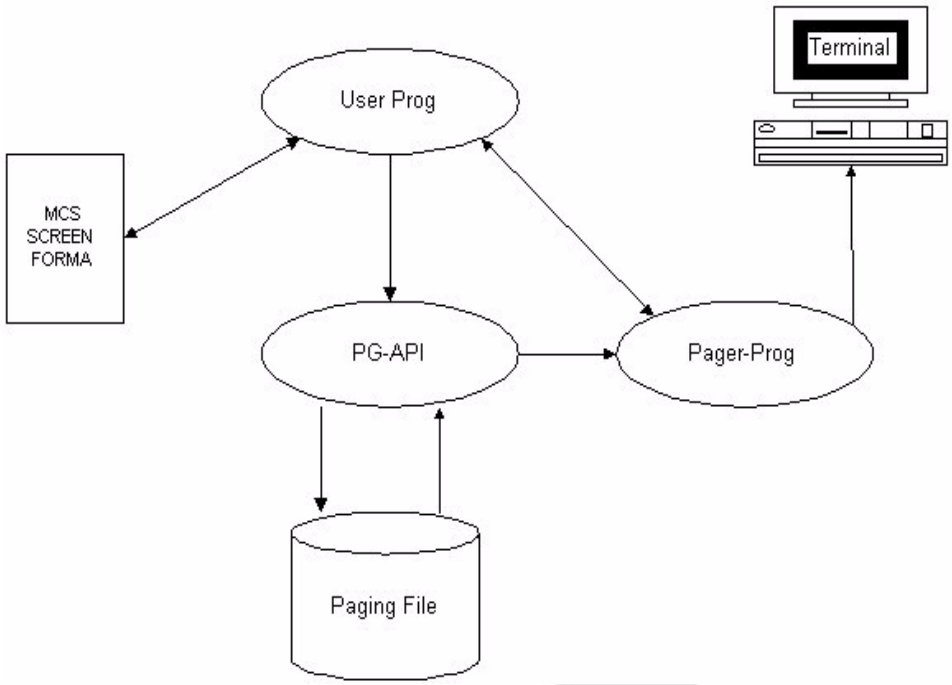
- After the call is completed, the program must check the status to determine if a message was available and was read. The T-TEST function does not wait for input; it reads an input message if one is available.

Paging API

Introduction to Terminal Paging

TIP provides paging, an efficient way to save screens (pages) into a file, and access them. Each page contains all the information necessary to repaint a full screen including the data. We provide a paging transaction, **tippager**, which enables you to browse through the paging file. This browsing transaction runs as a separate program. For details, see the *TIP Utilities* manual.

Figure 1: Architectural Overview



The paging API is called TIPPAGE. The screens are saved and retrieved in MCS formats. User applications must call FCS-OPEN before starting to page. TIP opens a paging file for that particular session, and the file is created if necessary. From then onwards every paging call will reference that file. The interface for the native TIP paging functions and the paging processor is discussed in the following sections.

Paging File

TIP creates paging files on a per TIP-session basis. As the sessions open, their corresponding paging files are created. These files are deleted as the associated TIP sessions end. The maximum length of a given paging file depends on two configurable parameters: page size, and number of pages per session. These parameters are configurable at boot time. By default, a session is limited to 20 pages in all and every page is 4K in size

Paging File Names

The paging file is created in the TIPSRC/tmpwrk directory under the name “page.*sess#*” where the extension to the file is a session number. For example: page.2536. This ensures a unique paging file name for every session.

TIPPAGE Paging API

The main interface consists of a subroutine called TIPPAGE. You pass the function to be performed to TIPPAGE as a parameter along with any

other required arguments. Thus a call to open a paging file in COBOL looks like this:

Syntax:

```
CALL "TIPPAGE" USING     FCS-OPEN
                         FILE-LFN
```

The supported functions are tabulated below.

Parameter 1 (Function)	Parameter 2	Parameter 3	Parameter 4
FCS-OPEN	FILE-LFN		
FCS-CLOSE	FILE-LFN		
FCS-ADD	FILE-LFN	MCS	
FCS-PUT	FILE-LFN	MCS	PAGE-NUM
FCS-GET	FILE-LFN	MCS	PAGE-NUM
FCS-DELETE	FILE-LFN		
FCS-GETRN	FILE-LFN	PAGE-STATUS-BUFFER	
FCS-ACCESS	FILE-LFN		

Common Parameters

As one can see from the above table there are three common data structures used by the above functions. These data structures are explained here:

FILE-LFN:

This data structure is used to hold the return status of the function call. It consists of 9 bytes of data; the 9th byte contains the status code. The first 8 bytes are reserved for the logical file name and are not used by the TIPPAGE. A value other than STS-GOOD in the FILE-STS field indicates an error. This data structure is used by every function call listed here and therefore, its explanation will not be repeated in the following calls.

```
05  FILE-LFN                PICTURE X(9).
05  FILLER REDEFINES FILE-LFN.
    07  FILE-NAME           PICTURE X(8).
    07  FILE-STATUS         PICTURE X(1).
        88  STS-GOOD            VALUE " ".
```

MCS:

This is a pointer to the MCS screen header and the data. The header contains information such as the name of the screen and the size of the screen data etc. A copy book TC-MCS is provided for application programmers. For detailed explanation on MCS structure, see the MCS section in the "**TIP Programming Reference**".

PAGE-NUM:

It is a four byte field and contains the page number in COBOL format.

```
01 PAGE-NUM PIC 9(9) COMP.
```

PAGE-STATUS-BUFFER:

This structure is used to return the status of the paging file. The size of this structure is 20 bytes and it contains information such as the number of data pages in the paging file, the current page number, and the maximum number of pages allowed etc. The structure is shown below.

```
01   PAGE-STATUS-BUFFER.
     10   STORED-DATA-PAGES    PICTURE 9(9) COMP.
     10   CURRENT-PAGE-NUMBER  PICTURE 9(9) COMP.
     10   STORED-INDEX-PAGES   PICTURE 9(9) COMP.
     10   MAX-DATA-PAGES       PICTURE 9(9) COMP.
     10   MAX-INDEX-PAGES      PICTURE 9(9) COMP.
```

Function Calls

A detailed explanation of the function calls is given below.

FCS-OPEN

Open a paging file for the current session.

Syntax:

```
CALL "TIPPAGE" USING    FCS-OPEN
                        file-lfn
```

Where:

FCS-OPEN
Function code from the TC-FCS copybook.

file-lfn
Return status of the function call.

This must be the very first call made by the user to start paging. It opens a paging file on per session basis. If necessary, the file is created. Then it is initialized.

A status condition of other than STS-GOOD in the FILE-LFN indicates a failure.

FCS-CLOSE

Close the paging file.

Syntax:

```
CALL "TIPPAGE" USING    FCS-CLOSE
                        file-lfn
```

Where:

FCS-CLOSE
Function code from the TC-FCS copybook.

file-lfn
Return status of the function call.

This function closes the paging file. It also sets a flag in the paging file's header called "stopPaging". This stops any other transaction writing into the paging file. This flag can only be cleared by the FCS-DELETE call.

Any paging function call trying to write or update a page after the FCS-CLOSE call will return an error. This function should only be called when no more paging is expected in the current session. If the paging is resumed using FCS-DELETE, all the previous data stored in the file is lost.

A status condition of other than STS-GOOD in the FILE-LFN indicates a failure.

FCS-ADD

Appends an MCS screen page to the paging file.

Syntax:

```
CALL "TIPPAGE" USING    FCS-ADD
                        file-lfn
                        mcs
```

Where:

FCS-ADD
Function code from the TC-FCS copybook.

file-lfn
Return status of the function call.

mcs The mcs interface packet.

The MCS address points to the screen information (including the data) to be written. It is an error to write a page beyond the maximum allowable limit of DPSMAXPAGES.

A status condition of other than STS-GOOD in the FILE-LFN indicates a failure.

FCS-PUT

Updates a page already written in the paging file.

Syntax:

```
CALL "TIPPAGE" USING    FCS-PUT
                        file-lfn
                        mcs
                        page-num
```

Where:

FCS-PUT
Function code from the TC-FCS copybook.

file-lfn
Return status of the function call.

mcs The mcs interface packet.

page-num
The page number in the paging file.

The new page of information is pointed by the MCS, which includes data for the screen. The PAGE-NUM contains the page number to be over written. The page number must be within the range 1 to STORED-DATA-PAGES.

A status condition of other than STS-GOOD in the FILE-LFN indicates a failure.

FCS-GET

Retrieve an MCS screen full of information from the paging file.

Syntax:

```
CALL "TIPPAGE" USING    FCS-GET
                        file-lfn
                        mcs
                        page-num
```

Where:

FCS-GET
Function code from the TC-FCS copybook.

file-lfn
Return status of the function call.

mcs The mcs interface packet.

page-num
The page number in the paging file.

The screen is retrieved into the area pointed by the MCS. The page to be retrieved is specified as a PAGE-NUM. The page number must represent one of the pages already stored i.e. it must be within the range of 1 to STORED-DATA-PAGES.

A status condition of other than STS-GOOD in the FILE-LFN indicates a failure.

FCS-DELETE

Delete all pages from the paging file and reinitialize the file header.

Syntax:

```
CALL "TIPPAGE" USING    FCS-DELETE
                        file-lfn
```

Where:

FCS-DELETE
Function code from the TC-FCS copybook.

file-lfn
Return status of the function call.

The file remains open and accessible to the user program or any following transactions.

A status condition of other than STS-GOOD in the FILE-LFN indicates a failure.

FCS-GETRN

Get the status record from the paging file.

Syntax:

```
CALL "TIPPAGE" USING    FCS-GETRN
                        file-lfn
                        page-status-buffer
```

Where:

FCS-GETRN
Function code from the TC-FCS copybook.

file-lfn
Return status of the function call.

page-status-buffer
Status buffer of the paging file as outlined above.

The status record contains information such as the current page number, the data pages stored, the maximum data pages allowed etc. See the description of the PAGE-STATUS-BUFFER above.

A status condition of other than STS-GOOD in the FILE-LFN indicates a failure.

File Control System (FCS)

This section describes the facilities of the TIP File Control System (**FCS**). FCS is the TIP component that provides the interface between transaction programs and data files.

FCS Overview

FCS and Program Access

FCS allows transaction programs to access:

- Standard indexed and relative record files and sequential files
- Standard libraries (directories)
- TIP dynamic files (files used on demand)
- TIP edit buffers.

Interface Level

The interface is implemented at the subroutine call level—all requests for file services call a supplied subroutine with appropriate parameters describing the required information.

TIP supports multiple FCS server processes to distribute I/O loads.

TIPFCS

FCS is the interface between transaction programs and online files; it provides services at the program "CALL" level. Programs call one subroutine (TIPFCS), and provide parameters that select the desired function and relevant file and record information.

Logical File Name

Programs refer to files by referencing a Logical File Name (LFN). The LFN is the name by which the file is known to TIP. An entry in the TIP definition relates a LFN to the actual physical file. All online files must have an entry in the TIP definition.

TIP$SEC and TIP$SYS Entries

The TIP system file, TIP$SYS, contains entries that define each TIP data file.

The TIP security file, TIP$SEC, contains entries for each TIP data file that specify:

- The group ownership of the data file
- And the security level required to access the data file.

Programs may only access files that are assigned to the program (either by an explicit OPEN request to FCS or an implicit OPEN.

Active File Table

All files assigned to programs have entries in the Active File Table (AFT) for the process. Files that have entries in the AFT are available (by reference to the LFN) to all programs that are run by that process until they are unassigned.

File Organizations Supported

FCS fully supports the following file organizations:

ISAM
Indexed Sequential Access Method. TIP supports indexed files with up to 10 key fields. Thus, support for MIRAM (Unisys System/80 environment) is included.

DAM
Direct Access Method (often referred to as relative files).

SAM
Sequential Access Method.

FCS also provides:

- The ability to access modules in a directory (similar in concept to a library with elements or a partitioned data set on some mainframes).
- The capability of creating (on demand) TIP Dynamic Files
- The ability to access Edit Buffers.

Record Locking

A *record locking feature* maintains file integrity. Records being updated are locked when read; locked records are unavailable to other processes until they are released.

Journaling Online Files

A user may be journal on online file by specifying a TIP definition option. This facility allows either *before images* or *after images* (or both) to be written to the TIP journal file:

Before Images

- **Before** images are often used to roll back updates that were not completed due to a hardware or software failure.

After Images

- **After** images can be used as audit trail information or applied to backup files to reprocess updates in the event of a hardware or software failure.

Dynamic Files

Dynamic files are direct access files that are managed by TIP. Programs may create or erase dynamic files, as necessary.

Setting a File in Sequential Mode

In the online environment that TIP provides, transaction programs often read indexed files in random mode—that is, by providing a specific key of the desired record.

FCS-SETL

When it is necessary for an online program to process an indexed file sequentially, a special purpose call (FCS-SETL) is made to the TIP File Control System to place the file in sequential mode. This technique is analogous to the batch COBOL verb "START".

As a side effect of setting a file in sequential mode, the program normally specifies a "starting point" by supplying a key value for one of the indices of the file.

Once a file is set in sequential mode, each call to FCS with the read request code can read forward and backward.

FCS-ESETL

When the program wishes to terminate sequential processing of a file, another special purpose call is issued (FCS-ESETL) to return to random processing mode.

Record Locking

It is generally accepted that two batch jobs should not simultaneously update the same file. Similarly, online users should be protected from the race conditions inherent in updating the same record at the same time.

Example:

To illustrate the problem, assume that JOE and TOM are working at different terminals updating FILEX and there is no record locking capability:

13. JOE displays record 500 intending to update it.
14. JOE is interrupted for a moment and TOM reads record 500, changes it at his terminal and rewrites the record in the file.
15. JOE then changes the record and rewrites it in the file; overlaying TOM's update and perhaps causing problems that may not appear until much later.

With the record locking capability provided by FCS, this situation cannot occur; the logical integrity of the updating process is maintained.

The TIP File Control System enforces the rule that a record to be updated must first be read and a lock requested (function FCS-GETUP). When the modification to the data is complete, the program may request a record rewrite (FCS-PUT).

HOLD=YES - Simple Record Locking

Specify HOLD=YES in the TIP file definition to select simple record locking. Specifying HOLD=YES implies that a program may lock a single record (at a time) from *this* file.

If the program issues a FCS-PUT without locking the record via an FCS-GETUP then the function status will be PIB-NOT-HELD and the FCS-PUT is rejected. This record holding scheme does not provide for online roll back of incomplete updates—since there is only a single record involved, the update is considered complete when the new record data is written.

HOLD=UP - Record Locking for Update

Specifying HOLD=UP in the TIP file definition parameters for a file allows a program to lock more than one record for the file at a time. The lock on each record remains in effect until that particular record is updated (via an FCS-PUT) or until that record is released (via an FCS-NOUP). This record holding scheme does not provide for online roll back.

This technique is often used in situations where there is a control record for a file (for example record 1) and that control record contains a pointer to "the next available" record.

Example:

- GETUP (rec #1)
- next available record from pointer in rec #1
- GETUP (next available record)
- move information to record area
- PUT (next available record)
- update next available pointer in rec #1
- PUT (rec #1)

If the system crashes at any point during this process, the control record remains intact and the next available record is still the next available record (although it may have different contents than it did before the attempted update).

HOLD=TR - Record Locking for Transaction

Specifying HOLD=TR in the TIP file definition causes the TIP File Control System to lock the records for that file for the duration of the transaction.

Also see the definition of transaction end in the *Program Control System (PCS)* section of this document.

This record locking scheme allows a program to lock several records for this file at one time. A program that receives a "held" status for a record in a file defined as HOLD=TR can retry the FCS-GETUP. TIP does not release any locks if an FCS-GETUP to a HOLD=TR file fails with a PIB-HELD status.

TIPFCS writes a "quick before image" of an updated record to the TIPIX.QBL file. TIP uses these quick before images to roll back the record if the transaction does *not* complete normally.

A user program can request a rollback of record updates since the last transaction end (commit point) by setting the PIB-LOCK-INDICATOR to "O" and issuing a CALL to FCS-TREN (see PIB-LOCK-INDICATOR in the Program Control System section).

Record Locking Summary

The table below compares the record locking schemes that are supported by TIP.

	HOLD=YES	**HOLD=UP**	**HOLD=TR**
Roll back Capability?	NO	NO	YES
Hold Multiple Records per File?	NO	YES	YES
When Records Released?	Update, or NOUP, or next GETUP or FCS-CLOSE*	Update, or NOUP, or FCS-CLOSE	TREN, or FCS-CLOSE*
Release on PIB-HELD Status?	YES	NO	NO
Possible Deadlock?	NO	YES	YES

Note: If an “H” is moved to the PIB-LOCK-INDICATOR before an FCS-CLOSE is performed, the file lock will be held.

Call TIPFCS - Common Parameters

Calls to TIPFCS must pass certain parameters. The first parameter is always a function code. The parameters after the first generally follow the format shown below, although there are a few minor exceptions as noted in the documentation:

Syntax:

```
CALL "TIPFCS" USING     function
                      [ file-pkt ]
                      [ record-area ]
                      [ key-value ]
                      [ index-num ]
```

Where:

Function
: The TIPFCS function code. See the next section describing the copy book TC-FCS from the TIP library.

  Warning: Passing an invalid function code to the TIPFCS subroutine results in the PIB-FUNCTION error status.

file-pkt
: The Logical file name packet. Must contain the logical file name of the file (as it is defined in the TIP definition).

record-area
: The record area. This area is where the data for a record is placed for a read operation or where the data is obtained for a write operation.

  The record area must be large enough to accommodate the largest record for the particular file.

key-value
: For an indexed file, this holds the record key. In some cases, a partial key value (that is, some key prefix) may be permitted.

  This parameter may be omitted (as documented) for some functions. For a direct (or non-indexed) file this is a binary fullword that holds the relative record number (i.e: PIC 9 BINARY).

index-num
: For indexed files, this specifies the desired index number (1 through 10 inclusive).

  Define this field as a binary halfword (PIC 999 BINARY). If this parameter is omitted, the primary key for the file. Determined from information supplied when the file is defined via the SMFILE utility is assumed.

Note: Throughout this document, references are made to the possibility of omitting parameters when calling TIPFCS. The implication in all cases is that a parameter may be omitted *only if all following parameters are also omitted*.

This is a restriction imposed by the operating system—each parameter passed on a CALL statement is identified by an address pointer. The called program (TIPFCS in this case) can only determine the *end* of the parameter list that is passed. There is no convention to identify omitted parameters.

File System Function Codes

The first parameter on every call to TIP FCS is a one-byte function code that specifies the file system operation to be performed. A copybook, TC-FCS, is supplied with TIP that COBOL programs can use to define the function codes.

Include this copybook in the WORKING-STORAGE SECTION of the COBOL program (the name selected for the 01 level item is not particularly important):

FCS Interface Packets

Two packets are used to control processing of files through FCS:

- Logical File Name Packet
- File Descriptor Packet.

All calls to TIPFCS make use of a Logical File Name packet; only calls to TIPFCS with the FCS-OPEN function use a File Descriptor Packet.

Logical File Name Packet

This is the primary control packet used for processing files. It consists of two fields:

- An eight byte field containing the Logical File Name (LFN) assigned to the file by the program. The value placed in this field is used to search the information in the TIP definition to determine which physical file is actually used by the program
- A one byte status field where FCS stores the completion status of the last call to TIPFCS for the file. This status code is the same as that returned in the PIB-STATUS field in the PIB.

Example of a Logical File Name Packet

```
05  PART-FILE.
    10  PART-LFN              PICTURE X(8).
    10  PART-FILE-STATUS      PICTURE X.
```

Example:

```
MOVE "PAYMAST"                TO PART-FILE
```

```
CALL "TIPFCS" USING          FCS-GET
                               ...
```

This example moves the logical file name "PAYMAST" to the file name packet before issuing a call to TIPFCS. FCS examines the logical file name and uses that name to search information in the TIP definition to associate the logical name with the physical name (the LFD).

Since TIPFCS modifies the status byte in this packet, the packet must be placed in the program's LINKAGE SECTION area.

File Descriptor (FDES) Packet

This packet is used during a call to FCS using the FCS-OPEN function. It establishes the relationship between a logical file name (LFN) and the real file to which I/O is to be done.

A file descriptor packet is required to open TIP Dynamic Files, TIP Edit Buffers, Library elements and, in situations where unusual processing is desired (such as opening a file with read-only access).

Fields of the copybook TC-FDES are described below:

Field	Purpose	
FDES-user id	May contain the user id or Group name to which the file belongs. If opening for:	
	Read	A complete search of the TIP definition is done.
	Output	Uses the specified value. If creating a dynamic file, this is set to the callers' user id.
FDES-CATALOG	Additional level of naming provided for dynamic files. If left as spaces or low values, this field is set to the FDES-FILE-NAME. FDES-CATALOG is ignored in TIPix when dealing with Edit buffers.	
FDES-FILE-NAME	File name for dynamic files or the defined logical file name for data management files. If left as spaces or low values, this field is set to the name in the logical file name packet (LFN).	
FDES-PASSWORD	This field may be used to assign a password to a TIP dynamic file or edit buffer. The password is established at the time the dynamic file or edit buffer is created; thereafter, all attempts to open the file must supply the password. If this field is spaces or low-values, no password is established.	

Field	Purpose	
FDES-FCS-CLASS	Class of opened file. If this field is a space or low values, TIPFCS opens the first file that it can find in the TIP definition with the supplied name.	
	E	Edit Buffer.
	P	Permanent dynamic file.
	S	Data file.
	T	Temporary dynamic file.
FDES-FCS-TYPE	Designates type of file (or element) desired:	
	C	Create new file.
	E	Open existing file.
	(space)	Access if it exists or create if it does not exist (dynamic files).
FDES-FCS-PERM	Designates type of file access:	
	R	Read only.
	W	Write only
	U	Read/write.
	(space)	Read/write.
FDES-FCS-LOCK	For Edit Buffers and TIP Dynamic files, this field may be set to a "Y" or "N" to indicate whether exclusive use of the file is required. If this field is not set to "Y", a value of "N" is assumed. PIB-LOCKED status is returned if any other process (online program) is using the Edit Buffer or Dynamic File.	

FCS Miscellaneous Functions

FCS-HOLD - Hold Resource

A program may use this function to place a user-defined value in the TIP key-holding table. Using this feature, cooperating processes can use some string of characters as a “sentinel” to implement a queuing mechanism so that only one of the processes runs at a time.

The value contained by the key-holding table is treated as a HOLD=UP type of lock — no rollback considerations apply, and the lock is not

discarded if the process receives PIB-HELD status on some other FCS-HOLD call.

The value placed in the key-holding table is the combination of the user-defined value, and a pointer to the actual physical LFD name of the associated file.

Syntax:

```
CALL 'TIPFCS' USING     FCS-HOLD
                        file-pkt
                       [hold-value]
```

Where:

FCS-HOLD
Function code from the TC-FCS copybook.

file-pkt
Logical file name packet.

This packet must contain the LFN of a file that the program has accessed

hold-value
A field containing the character string entered in the key-holding table. Exactly 4 bytes of data are entered in the table. If this parameter is omitted, a fullword containing 1 is used.

Error Conditions:

PIB-STATUS	Meaning
PIB-FUNCTION	The logical file name is not assigned to the program.
PIB-HELD	Some other process currently holds the specified value for the same physical file (LFD).
PIB-FULL	The system is unable to establish the record lock because the system resources required are not available. You may want to increase your global system shared memory (with tipinstall -M).

FCS-JOURNAL - Write User Journal Record

Transactions may use the FCS-JOURNAL function to write a *user-defined* record to the TIP Journal. These 'user' journal records are often used for accounting or audit purposes.

The format of a *user* record in the journal file is entirely at the discretion of the program writing the record. The only restriction is that the record must contain a proper record prefix (described in *Accessing TIP Journal Files).*

Syntax*:*

```
CALL "TIPFCS" USING     FCS-JOURNAL
                        dummy-file-pkt
                        jrn-record
```

Where:

FCS-JOURNAL
Function code from the TC-FCS copybook.

dummy-file-pkt
This is a dummy parameter required to maintain symmetry with other file system calls. FCS ignores the contents of this parameter.

jrn-record
The record to be written to the journal file.

This group item must be halfword aligned and must contain a proper journal record prefix including the total length of the prefix and user data.

Error Conditions:

PIB-STATUS	Meaning
PIB-IO-ERROR	An I/O error occurred while accessing the file. One cause for this error may be that the record length exceeds the system maximum record size as specified at install time.

Additional considerations:

- The copybook TC-JRN (see the section of this manual entitled "TIP Journal File") is provided as a layout of the journal record. Before issuing a call to FCS-JOURNAL, the user program must move an appropriate value to the length field (in the copybook it is named JRN-REC-LEN).
- The value placed in JRN-REC-LEN is the length of the journal record prefix plus the number of bytes of data that follows the prefix. If this length is less than or equal to zero, a PIB-IO-ERROR is returned.
- The program need not supply any other information in the prefix area since the TIP file system fills in the information before writing the USER record to the journal file.
- The copybook TC-JRNC is provided to supply key constant values for the journal record layout. These values are: the journal record prefix length, the maximum journal record data length, and the maximum journal record length.

FCS-RELEASE - Release Resource

Release an entry in the TIP key-holding table that was entered by a prior call to TIPFCS with the FCS-HOLD function.

Syntax:

```
CALL 'TIPFCS' USING     FCS-RELEASE
                        file-pkt
                       [hold-value]
```

Where:

FCS-RELEASE
Function code from the TC-FCS copybook.

file-pkt
Logical filename packet.

This packet must contain the LFN of the file that was specified when the corresponding FCS-HOLD operation was issued.

hold-value
A field containing the character string in the key-holding table that is to be released. Exactly 4 bytes of data are required.

If this parameter is omitted, a fullword containing 1 is used.

Error Conditions:

PIB-STATUS	Meaning
PIB-FUNCTION	The logical file name is not assigned to the program.
PIB-NOT-HELD	The specified key value was not found in the key-holding table.

FCS-TREN - Mark Transaction End

If a file is defined in the TIP definition parameters with HOLD=TR ("hold for transaction"), TIP automatically rolls back any updates, additions or deletions if a program aborts while updating, adding, or deleting records in that file. Records being deleted are restored and file additions are removed.

The FCS-TREN function may be used explicitly (for example) if a program updates batches of records and wishes to establish a new "roll back" point after each "batch" to limit how far automatic roll back will occur if a subsequent abort occurs.

Syntax:

```
CALL "TIPFCS" USING     FCS-TREN
```

Where:

FCS-TREN
Function code from the TC-FCS copybook.

Only one parameter is required; any other parameters are ignored.

Use of FCS-TREN signals transaction end to TIP. Another use of this function is to cause TIP to examine a value that the program has placed in the field PIB-LOCK-INDICATOR. For example, a program set the "PIB-ROLLBACK" value in that field, and then called FCS-TREN to force a transaction roll back.

Example 1; Roll back updates done so far:

```
SET PIB-ROLLBACK          TO TRUE
CALL "TIPFCS" USING       FCS-TREN
```

Example 2; Mark new roll back point:

```
MOVE SPACE TO             PIB-LOCK-INDICATOR
CALL "TIPFCS" USING       FCS-TREN
```

CALL TIPFCER - Interpret FCS Error

A special purpose subroutine is provided to enable application programs to interpret error codes that are returned by the TIP File Control System (FCS). When a program issues a call to TIPFCS, error status is returned in two places:

- The PIB (PIB-STATUS)
- The ninth byte of the file name packet (the byte after the Logical File Name).

Programs are generally coded to anticipate a subset of the possible error conditions that might occur and take the appropriate action depending on the circumstances. For example, a "not found" error might be quite reasonable for certain read operations (example: customer not on file).

If the program needs to generate a "generic" error message for rare or unexpected error conditions, the TIPFCER subroutine may be used. This subroutine returns a standard-format error message text that describes the error condition that is passed as a parameter. This standard error text can then be used to form an informational message for the terminal operator.

Syntax:

```
CALL "TIPFCER" USING          file-pkt
                              msg-area
```

Where:

file-pkt

The logical file name packet (9 bytes, consisting of an 8-byte LFN and one byte status field) that was in use at the time an error was detected.

msg-area
An 80-byte work field that is to receive the standard error message text for the error status that is found in the FILE-PKT.

Example of result text:

```
FCS Error=?, File=????????, Meaning="............."
```

Example of Using TIPFCER:

```
05  PAYMAST-LFN                PICTURE X(9).
05  FCS-ERROR-TEXT             PICTURE X(80).
     (...)
CALL "TIPFCS" USING            FCS-GET
                               PAYMAST-LFN
                                {...}
 IF NOT PIB-GOOD
    CALL "TIPFCER" USING       PAYMAST-LFN
                               FCS-ERROR-TEXT
    CALL "ROLL" USING          FCS-ERROR-TEXT
     {...}
```

This example illustrates using the result from the call to TIPFCER to output a single line on the terminal. Of course, the message text can be used in whatever fashion the program considers appropriate.

Techniques for Deleting Records

Techniques for Deleting Records

TIP supports two types of record deletion schemes:

- Physical record deletion.
- Logical record deletion (often called delete flag)

You use the **smfile** utility to specify how to delete records in a file. If you don't specify logical deletion, physical deletion is assumed. Your choice is saved in the TIP file definition for that file.

Online programs may call the TIP File Control System (TIPFCS) to delete a record. TIPFCS performs the appropriate type of delete operation for the file.

Physical Record Deletion

The file system provides a standard physical delete mechanism for files:

This form of record deletion is *not* a convention—it is a real and effectively permanent delete.

Logical Record Delete

Logical record deletion is a *convention* established when a file is defined to TIP. A specific byte in the record is identified as the "delete flag". A

specific value is designated as the "flag value". The convention that TIP follows is:

When TIP reads a record from the file that contains the specified flag value in the specified location, TIP pretends that the record does not exist.

The crucial point of the convention is contained in the word **pretends**. The record physically exists, but is flagged with a specific flag value to make it "appear to TIP" as if it was deleted. The location that is normally chosen for the delete flag is the first byte of the record that is not part of a key field; in fact, many records contain some sort of status field that may be used for this purpose.

The designated value can be any value; X'FF' is often used although displayable graphics characters are easier to recognize.

Programmers must realize that this scheme is merely a *convention* that TIP follows — the records appear perfectly normal to other programs in the system!. In particular, batch programs must be prepared to recognize such "deleted" records and take appropriate action (such as ignoring them!)

The way you specify the location and value of the delete flag has changed as of TIP 2.1. You now enter the delete flag *location* as a 0-relative value. You *don't* have to change the existing file definitions because the internal representation has not changed.

The *value* of the delete flag can be specified:

- As a single displayable character
- As 2 hex digits. The default flag value is “FF”.

TIPFCS for Indexed Files

This section describes TIP file control system operations you may specify for indexed files.

User programs may access indexed files via any of the indices defined for the file. When a multi-indexed file is defined in TIP the length, location, and the attributes of each of the keys must be stated and one of the keys must be designated as the primary key (KEY1 is the default primary key). This information is provided when the file is defined to TIP using the **smfile** utility program.

Multi-indexed files used by TIP *must* have the primary key defined as (NDUP, NCHG)—no duplicate key values and no changes allowed to this key.

FCS-ADD - Indexed: Add Record

The FCS-ADD function code adds a new record to a file. The data supplied in the record area must contain the proper key information in the appropriate location(s).

If an FCS-ADD function is issued for a record that is currently marked "logically deleted", the TIP File System allows the "add" operation by overwriting the previously logically deleted record with the new record data.

Syntax:

```
CALL "TIPFCS" USING     FCS-ADD
                        file-pkt
                        record
```

Where:

FCS-ADD
Function code from the TC-FCS copybook.

file-pkt
Logical file name packet.

record
Record area containing new record data.

Error Conditions:

PIB-STATUS	Meaning
PIB-DUP-KEY	A record with the same key already exists (for an index that does not permit duplicates).
PIB-FUNCTION	The file is not assigned to the program.
PIB-FULL	The system is unable to establish the record lock because the system resources required are not available. You may want to increase your global system shared memory (with tipinstall -M).
PIB-IO-ERROR	An I/O error occurred accessing the file.
PIB-WRONG-MODE	Write operations are not permitted for the file.

Additional Considerations:

- The FCS-ADD function can be used while the file is in sequential mode without affecting the current sequential position.

FCS-CLOSE - Indexed: Close File

The FCS-CLOSE function call indicates that a program is relinquishing access to a file. The corresponding entry for the file is removed from the Active File Table (AFT) of the issuing process. If there are no other online users of the file TIPFCS physically CLOSEs the file by issuing a UNIX kernel close request.

Syntax:

```
CALL "TIPFCS" USING     FCS-CLOSE
                        file-pkt
```

Where:

FCS-CLOSE
Function code from the TC-FCS copybook.

file-pkt
Logical file name packet.

Error Conditions:

PIB-STATUS	Meaning
PIB-FUNCTION	File is not assigned to the program

Additional Considerations:

- This function is intended for files that were opened by issuing an explicit FCS-OPEN function (not for files implicitly accessed by the program).

FCS-DELETE - Indexed: Delete Record

The FCS-DELETE function call deletes a record from the file. The TIP file system uses the applicable delete scheme as specified in the TIP definition for the file (see **smfile** in the *TIP Utilities* manual). See the description of record delete schemes in *Techniques for Deleting Records*.

The program *must* acquire a record lock (by a call to FCS-GETUP) before issuing this function call.

Syntax:

```
CALL "TIPFCS" USING     FCS-DELETE
                        file-pkt
                        record
```

Where:

FCS-DELETE
Function code from the TC-FCS copybook.

file-pkt
Logical file name packet.

record
Record area.

Error Conditions:

PIB-STATUS	Meaning
PIB-FUNCTION	The file is not assigned to the program.
PIB-IO-ERROR	An I/O error occurred on the disk
PIB-NOT-HELD	A prior FCS-GETUP for this record was not successful or the record lock has been released by TIP.

FCS-ESETL - Indexed: End Sequential Mode

Set a file to random processing mode (terminate sequential processing of the file).

Syntax:

```
CALL "TIPFCS" USING     FCS-ESETL
                        file-pkt
```

Where:

FCS-ESETL
Function code from the TC-FCS copybook.

file-pkt
Logical file name packet.

Error Conditions:

PIB-STATUS	Meaning
PIB-FUNCTION	The file is not assigned to the program.

FCS-FLUSH - Indexed: Flush File

The FCS-FLUSH function is provided for compatibility with TIP/30 programs from the Unisys System/80 environment. FCS-FLUSH is considered a no-op in the TIP environment.

Syntax:

```
CALL "TIPFCS" USING     FCS-FLUSH
                        file-pkt
```

Where:

FCS-FLUSH
Function code from the TC-FCS copybook.

file-pkt
Logical file name packet.

Error Conditions:

PIB-STATUS	Meaning
PIB-FUNCTION	The file is not assigned to the program.

FCS-GET - Indexed: Read by Key

Read a record with a specific key from a file. FCS-GET does not lock the record for update.

This section discusses the behavior of FCS-GET assuming that the file is *not* already set in sequential mode; see the following section for information about sequential mode gets.

Syntax:

```
CALL "TIPFCS" USING     FCS-GET
                        file-pkt
                        record
                      [ key ]
                      [ index-num ]
                      [ dup-count ]
```

Where:

FCS-GET
Function code from the TC-FCS copybook.

file-pkt
Logical file name packet.

record
Area where the record data is placed.

key
Record key. If this parameter is omitted, the key is taken from the record area.

index-num
Binary halfword holding the intended index number (PIC 9(3) BINARY.) If the index-num field is omitted, the default index number for the file is used.

dup-count

Binary fullword holding the ordinal number of the desired record in a set of duplicate records. For example: 100 means "return the 100th record". If this field is omitted, only the first record (of a set of duplicates) is accessed. Logically deleted records are not counted when FCS searches for the Nth duplicate record via the dup-count supplied.

Error Conditions:

PIB-STATUS	Meaning
PIB-FUNCTION	The file is not assigned to the program.
PIB-IO-ERROR	An I/O error occurred on the disk
PIB-NOT-FOUND	The record does not exist or is flagged as deleted using a logical delete flag. If the record was logically deleted, the record is returned in the specified record area.
PIB-DUPS-AHEAD	Is set in the field PIB-DETAIL-STATUS if there is another record following the retrieved record with a duplicate key. This setting can alert the program there are further records in a set of duplicates. MBP ISAM does not provide this status information, so it cannot be passed to the application.

FCS-GET - Indexed: Read Sequential Key

Read the next record from a file that has already been set in sequential mode (by a prior call to one of the various FCS-SETL-xx functions.) Using FCS-GET implies that the record is *not* locked for update.

Syntax:

```
CALL "TIPFCS" USING     FCS-GET
                        file-pkt
                        record
```

Where:

FCS-GET

Function code from the TC-FCS copybook.

file-pkt

Logical file name packet.

record
Area where record data is placed.

Error Conditions:

PIB-STATUS	Meaning
PIB-FUNCTION	The file is not assigned to the program.
PIB-IO-ERROR	An I/O error occurred on the disk
PIB-EOF	End of file is reached
PIB-NOT-FOUND	End of file is reached. This status may be returned if the file was placed in sequential mode by issuing a call to TIPFCS with the function FCS-SETL-GT.
PIB-DUPS-AHEAD	Is set in the field PIB-DETAIL-STATUS if there is another record following the retrieved record with a duplicate key. This setting can alert the program there are further records in a set of duplicates. MBP ISAM does not provide this status information, so it cannot be passed to the application.

Logically deleted records are skipped by the file system when reading in sequential mode.

FCS-GET-INDEX - Indexed: Read for Key

This function does not return a record but sets the PIB-STATUS. It is intended to be used to determine if the record key exists.

Syntax:

```
CALL "TIPFCS" USING     FCS-GET-INDEX
                        file-pkt
                        key
                      [ index-num ]
                      [ dup-count ]
```

Where:

FCS-GET-INDEX
Function code from the TC-FCS copybook

file-pkt
Logical file name packet

key
Record key.

index-num

Binary halfword holding the intended index number. If the index-num field is omitted, the default index number for the file is used. The default index is not necessarily the index for the primary key.

dup-count

Binary fullword holding the ordinal number of the desired record in a set of duplicate records. For example: 100 means "return the 100th record". If this field is omitted, only the first record (of a set of duplicates) is accessed. Logically deleted records are not counted when FCS searches for the Nth duplicate record via the dup-count supplied.

Error Conditions:

PIB-SECTION	Meaning
PIB-FUNCTION	The file is not assigned to the program.
PIB-IO-ERROR	An I/O error occurred on the disk.
PIB-NOT-FOUND	The key does not exist.
PIB-DUPS-AHEAD	Is set in the field PIB-DETAIL-STATUS if there is another record following the retrieved record with a duplicate key. This setting can alert the program there are further records in a set of duplicates. MBP ISAM does not provide this status information, so it cannot be passed to the application.

Example:

```
MOVE MCS-KEY              TO KEY-FIELD
CALL "TIPFCS" USING       FCS-GET-INDEX
                          FILE-PKT KEY-FIELD

IF PIB-GOOD
    PERFORM GETUP
     { }
    PERFORM PUT
ELSE
    PERFORM ADD
END-IF
```

FCS-GET-KEYED - Indexed: Read by Key

Read a record with a specific key from a file even if the file is in sequential mode. FCS-GET-KEYED does not lock the record for update.

This section discusses the behavior of FCS-GET-KEYED assuming that the file *may* be set in sequential mode.

This function is intended for situations where the file is in sequential mode and a random read is desired. Issuing an FCS-GET retrieves the next sequential record; this call ensures that a random read is issued.

Syntax:

```
CALL "TIPFCS" USING     FCS-GET-KEYED
                        file-pkt
                        key
                        record
                      [ index-num ]
                      [ dup-count ]
```

Where:

FCS-GET-KEYED
Function code from the TC-FCS copybook.

file-pkt
Logical file name packet.

record
Area where the record data is placed.

key
Record key. If this parameter is omitted, the key is taken from the record area.

index-num
Binary halfword holding the intended index number. If the index-num field is omitted, the default index number for the file is used. The default index is not necessarily the index for the primary key.

dup-count
Binary fullword holding the ordinal number of the desired record in a set of duplicate records. For example: 100 means "return the 100th record". If this field is omitted, only the first record (of a set of duplicates) is accessed. Logically deleted records are not counted when FCS searches for the Nth duplicate record via the dup-count supplied.

Error Conditions:

PIB-STATUS	Meaning
PIB-FUNCTION	The file is not assigned to the program.
PIB-IO-ERROR	An I/O error occurred on the disk.
PIB-NOT-FOUND	The record does not exist or is flagged as deleted using a logical delete flag. If the record was logically deleted, the record is returned in the specified record area.
PIB-DUPS-AHEAD	Is set in the field PIB-DETAIL-STATUS if there is another record following the retrieved record with a duplicate key. This setting can alert the program there are further records in a set of duplicates. MBP ISAM does not provide this status information, so it cannot be passed to the application.

FCS-GET-SEQ-LOCK - Indexed: WORKAROUND

This call is *not* implemented in TIP.

TIP/ 30
: In TIP/30 FCS-GET-SEQ-LOCK performs the following: Read a record with a specific key from a file that should already be in sequential mode. Use FCS-GET-SEQ-LOCKED to lock only one record key. If you issue this call, and then repeat it, the first lock is released.

  The FCS-GET-SEQ-LOCKED function was added to TIP/30 because a GETUP was not allowed while reading a file in sequential mode.

TIP
: However, TIP allows FCS-GETUP while reading a file in sequential mode.

FCS-GETUP always reads the record with the specified key even if the file is in sequential mode, and has no effect on the sequential position in the file. Thus, the FCS-GET-SEQ-LOCKED function has not been implemented.

This function allowed an application to read a file sequentially, locking each record as it was read, and unlocking it when the next record was read. So that if the application found the record it wanted to modify it could issue an FCS-PUT without losing the sequential file position.

When porting TIP/30 applications that use this call to TIP, change FCS-GET-SEQ-LOCK to FCS-GET and add FCS-GETUP requests as required by the application. You may have to add FCS-NOUP calls if your application needs to call FCS-GETUP for records that it does not update.

Although there is an additional FCS function call to lock the record, this technique may improve performance because there is no locking and unlocking of records the application is not interested in.

FCS-GET-SEQ-NEXT - Indexed: Read Next Record

Read the next record from a file that has already been set in sequential mode (by a prior CALL to one of the various FCS-SETL-xx functions.) This call is the same as issuing FCS-GET for a file already in sequential mode. The function name of this call is perhaps more clear to the programmer reading the code.

Syntax:

```
CALL "TIPFCS" USING     FCS-GET-SEQ-NEXT
                        file-pkt
                        record
```

Where:

FCS-GET-SEQ-NEXT
Function code from the TC-FCS copybook.

file-pkt
Logical file name packet.

record
Area where the record data is placed.

Error Conditions:

PIB-STATUS	Meaning
PIB-EOF	End of file is reached.
PIB-FUNCTION	The file is not assigned to the program.
PIB-IO-ERROR	An I/O error occurred on the disk.
PIB-NOT-FOUND	End of file is reached. This status may be returned if the file was placed in sequential mode by issuing a call to TIPFCS with the function FCS-SETL-GT. If the record was logically deleted, the record is returned in the specified record area.
PIB-DUPS-AHEAD	Is set in the field PIB-DETAIL-STATUS if

PIB-STATUS	Meaning
	there is another record following the retrieved record with a duplicate key. This setting can alert the program there are further records in a set of duplicates. MBP ISAM does not provide this status information, so it cannot be passed to the application.

FCS-GET-SEQ-PREV - Indexed: Read Previous Record

Read the previous record from a file that has already been set in sequential mode (by a prior CALL to one of the various FCS-SETL-xx functions.)

Syntax:

```
CALL "TIPFCS" USING     FCS-GET-SEQ-PREV
                        file-pkt
                        record
```

Where:

FCS-GET-SEQ-PREV
: Function code from the TC-FCS copybook.

file-pkt
: Logical file name packet.

record
: Area where the record data is placed.

Error Conditions:

PIB-STATUS	Meaning
PIB-EOF	End of file is reached
PIB-FUNCTION	The file is not assigned to the program.
PIB-IO-ERROR	An I/O error occurred on the disk.
PIB-NOT-FOUND	End of file is reached. This status may be returned if the file was placed in sequential mode by issuing a call to TIPFCS with the function FCS-SETL-GT. If the record was logically deleted, the record is returned in the specified record area.

FCS-GETRN - Indexed: Read by Record Number

Read a record from an indexed file via a record number.

Whenever a record is read from an indexed file, the file system places a unique "record number" in the field PIB-MIRAM-REL-REC-NUM. Interested programs can save this value and use it at some later time to directly retrieve the same record.

FCS-GETRN does not lock the record for update.

Syntax:

```
CALL "TIPFCS" USING     FCS-GETRN
                        file-pkt
                        record
                        rel-rec-num
```

Where:

FCS-GETRN
Function code from the TC-FCS copybook.

file-pkt
Logical file name packet.

record
Area where the record data is placed.

rel-rec-num
A binary fullword containing the record number of the record to read.

Error Conditions:

PIB-STATUS	Meaning
PIB-EOF	The requested record is beyond the last record in the file. The field PIB-MIRAM-REL-REC-NUM is set to the highest valid record number in the file.
PIB-FUNCTION	The file is not assigned to the program.
PIB-IO-ERROR	An I/O error occurred on the disk.
PIB-NOT-FOUND	The record does not exist or has been deleted. If the record is logically deleted, the record data is returned in the specified record area.

Additional Considerations:

- FCS-GETRN may be used while the file is in sequential mode without affecting the current sequential position.

FCS-GETUP - Indexed: Read With Lock

Read the record with the specified key with intent to update. The PRIMARY key of the record is placed in the TIP internal key holding table. The record is LOCKED - other processes receive an error status if an attempt is made to FCS-GETUP, FCS-LOCK or FCS-ADD the same record.

Syntax:

```
CALL "TIPFCS" USING     FCS-GETUP
                        file-pkt
                        record
                      [ key ]
                      [ index-num ]
                      [ dup-count ]
```

Where:

FCS-GETUP
Function code from the TC-FCS copybook.

file-pkt
Logical file name packet.

record
Area where the record data is placed.

key
Record key. If omitted, the key is taken from the record area.

index-num
Binary halfword holding the desired index number. If the index-num field is omitted, the default index number for the file is used. The default index is not necessarily the index for the primary key.

dup-count
Binary fullword holding the ordinal number of the desired record in a set of duplicate records. For example: 100 means "return the 100th record". If this field is omitted, only the first record (of a set of duplicates) is accessed. Logically deleted records are not counted when FCS searches for the Nth duplicate record via the dup-count supplied.

Error Conditions:

PIB-STATUS	Meaning
PIB-FUNCTION	The file is not assigned to the program.
PIB-FULL	The system is unable to establish the record lock because the system resources required are not available. You may want to increase your global system shared memory (with tipinstall -M).
PIB-IO-ERROR	An I/O error occurred on the disk.
PIB-NOT-FOUND	The record does not exist or is flagged as deleted using a logical delete flag. If the record was logically deleted, the record is returned in the specified record area.
PIB-HELD	The record is currently locked by some other process in the TIP system. Normally, programs that receive PIB-HELD retry the GETUP request.
PIB-DUPS-AHEAD	Is set in the field PIB-DETAIL-STATUS if there is another record following the retrieved record with a duplicate key. This setting can alert the program there are further records in a set of duplicates. MBP ISAM does not provide this status information, so it cannot be passed to the application.
PIB-DEADLOCK-DETECTED	The I/O system has detected an application logic deadlock condition. At this point the transaction is rolled back and the application program may take whatever corrective action is desired. The application could report an error to the end user or restart the transaction over again.

Additional Considerations:

- If a user program receives a function status of PIB-HELD in response to a FCS-GETUP (meaning the record is locked by some other process) then FCS automatically pauses the caller for a small amount of time. The program may try the GETUP again or CALL TIPTIMER to wait a little longer.
- The number of times the retry is attempted is dependent on the expected length of time the "other process" may lock the record and the probability of such conflicting attempts to update the same record.

After some reasonable number of retries, the program must consider some alternate action such as informing the terminal operator about the situation and asking whether or not the program should continue to retry.

- FCS-GETUP may be used while the file is in sequential mode without affecting the current sequential position.

NOTE: There is an inherent limit to the number of record locks that can be maintained for files that have the following characteristics:

- HOLD for Transaction is specified,
- The file is defined to use the "tipfcs" (D-ISAM) file server, and
- The file is specified as access "shared".

TIP can maintain only 200 record locks for the file in this situation.

This limitation can be worked around by declaring the file as "exclusive" access or by ensuring that less than 200 locks are requested for the file.

FCS-LOCK - Indexed: Lock Record

Read the record with the specified key but do *not* return the record. The PRIMARY key of the record is placed in the TIP internal key holding table (see separate discussion of this topic). The record is LOCKED - other processes receive an error status if an attempt is made to FCS-LOCK, FCS-GETUP or FCS-ADD the same record.

Syntax:

```
CALL "TIPFCS" USING     FCS-LOCK
                        file-pkt
                        key
                      [ index-num ]
                      [ dup-count ]
```

Where:

FCS-LOCK
Function code from the TC-FCS copybook.

file-pkt
Logical file name packet.

key
Record key.

index-num
Binary halfword holding the desired index number. If the index-num field is omitted, the default index number for the file is used. The default index is not necessarily the index for the primary key.

dup-count

Binary fullword holding the ordinal number of the desired record in a set of duplicate records. For example: 100 means "return the 100th record". If this field is omitted, only the first record (of a set of duplicates) is accessed. Logically deleted records are not counted when FCS searches for the Nth duplicate record via the dup-count supplied.

Error Conditions:

PIB-STATUS	Meaning
PIB-FUNCTION	The file is not assigned to the program.
PIB-FULL	The system is unable to establish the record lock because the system resources required are not available. You may want to increase your global system shared memory (with tipinstall -M).
PIB-IO-ERROR	An I/O error occurred on the disk.
PIB-NOT-FOUND	The record does not exist or is flagged as deleted using a logical delete flag. If the record was logically deleted, the record is returned in the specified record area.
PIB-HELD	The record is currently locked by some other process in the TIP system. Normally, programs that receive PIB-HELD will retry the LOCK request.
PIB-DUPS-AHEAD	Is set in the field PIB-DETAIL-STATUS if there is another record following the retrieved record with a duplicate key. This setting can alert the program there are further records in a set of duplicates. MBP ISAM does not provide this status information, so it cannot be passed to the application.
PIB-DEADLOCK-DETECTED	The I/O system has detected an application logic deadlock condition. At this point the transaction is rolled back and the application program may take whatever corrective action is desired. The application could report an error to the end user or restart the transaction over again.

Additional Considerations:

- If a user program receives a function status of PIB-HELD in response to a FCS-LOCK (meaning the record is locked by some other process) then FCS automatically pauses the caller for a small amount of time. The program may try the FCS-LOCK again or CALL TIPTIMER to wait a little longer.
- The number of times the retry is attempted is dependent on the expected length of time the "other process" may lock the record and the probability of such conflicting attempts to update the same record. After some reasonable number of retries, the program must consider some alternate action such as informing the terminal operator about the situation and asking whether or not the program should continue to retry.
- FCS-LOCK may be used while the file is in sequential mode without affecting the current sequential position.

FCS-NEXT - Indexed: Get Next Record

The FCS-NEXT function retrieves the next record (sequentially) from an indexed file.

Use FCS-NEXT only when one record is required at a time. If a number of records are to be read, it is more efficient to place the file in sequential mode (using a function of FCS-SETL-xx) and issuing the required number of FCS-GET functions to read the file sequentially.

Use of this call does not affect the current sequential position (if the file happens to be in sequential mode).

Syntax:

```
CALL "TIPFCS" USING     FCS-NEXT
                        file-pkt
                        record
                      [ key ]
                      [ index-num ]
                      [ dup-count ]
```

Where:

FCS-NEXT
Function code from the TC-FCS copybook.

file-pkt
Logical file name packet.

record
Record area where the data is placed.

key
Record key. If this parameter is omitted, the key is taken

from the record area. If this parameter is supplied, the actual key of the record returned is placed in this field by TIPFCS - this facilitates a subsequent call to FCS-NEXT. This action, however, alters the field and means that the field must be located in the program's LINKAGE SECTION to permit the program to run as a reentrant process.

index-num

Binary halfword holding the desired index number. If the index-num field is omitted, the default index number for the file is used. The default index is not necessarily the index for the primary key.

dup-count

Binary fullword holding the ordinal number of the desired record in a set of duplicate records. For example: 100 means "return the 100th record". If this field is omitted, only the first record (of a set of duplicates) is accessed. Logically deleted records are not counted when FCS searches for the Nth duplicate record via the dup-count supplied.

Error Conditions

PIB-STATUS	Meaning
PIB-FUNCTION	The file is not assigned to the program.
PIB-IO-ERROR	An I/O error occurred on the disk.
PIB-NOT-FOUND	The next record does not exist.
PIB-DUPS-AHEAD	Is set in the field PIB-DETAIL-STATUS if there is another record following the retrieved record with a duplicate key. This setting can alert the program there are further records in a set of duplicates. MBP ISAM does not provide this status information, so it cannot be passed to the application.

FCS-NOUP - Indexed: Cancel Update

The FCS-NOUP function call is used to "unlock" a record that has been locked via a prior call to FCS-GETUP or FCS-LOCK *provided the record has not been updated*! For example, in certain situations, a program may issue FCS-GETUP and lock a record only to later determine that an update is not desired for some reason.

Syntax:

```
CALL "TIPFCS" USING     FCS-NOUP
                        file-pkt
                      [ key ]
```

Where:

FCS-NOUP
Function code from the TC-FCS copybook.

file-pkt
Logical file name packet.

key
Specific primary key value that is to be released. If omitted, all key values currently held by this process for the specified file are released.

Error Conditions:

PIB-STATUS	Meaning
PIB-FUNCTION	The file is not assigned to the program.
PIB-NOT-HELD	The record was not held.

Additional Considerations:

- The FCS-NOUP function may be issued while the file is in sequential mode without affecting the current sequential position.

FCS-OPEN - Indexed: Open File

Make the specified file available for processing by programs at the calling terminal. An entry in the Active File Table (AFT) is created for the process issuing this call.

Files are normally automatically made available to the program by an implicit request for file names as defined in the program's TIP definition entry (smprog). If a program needs to access more files, the files may be opened by issuing explicit calls to TIPFCS with the FCS-OPEN function. Alternatively, the program may simply issue calls to access the files. (The choice is largely a matter of whether or not you prefer to get an error on the OPEN or the first use of the file).

Syntax:

```
CALL "TIPFCS" USING     FCS-OPEN
                        file-pkt
                      [ file-desc ]
```

Where:

FCS-OPEN
Function code from the TC-FCS copybook.

file-pkt
Logical file name packet.

file-desc
File descriptor packet - see separate description of the copybook "TC-FDES". If omitted, the name in the FILE-PKT is used to build a file descriptor.

Error Conditions:

PIB-STATUS	Meaning
PIB-IO-ERROR	An I/O error occurred while opening the file.
PIB-DUP-AFT-NAME	An entry already exists in the Active File Table (for the issuing process) that matches the logical file name used in the FILE-PKT field.
PIB-LOCKED	The file is closed.
PIB-NOT-FOUND	The logical file name is not defined in the active groups for this TIP session.
PIB-FUNCTION	The open function could not be performed. Check Unix permissions.

FCS-PREV - Indexed: Get Previous Record

The FCS-PREV function retrieves the previous record (sequentially) from an indexed file. Using this call does not affect current sequential position (if the file happens to be in sequential mode).

Syntax:

```
CALL "TIPFCS" USING     FCS-PREV
                        file-pkt
                        record
                      [ key ]
                      [ index-num ]
                      [ dup-count ]
```

Where:

FCS-PREV
Function code from the TC-FCS copybook.

file-pkt

Logical file name packet.

record

Record area where the data is placed.

key

Record key. If this parameter is omitted, the key is taken from the record area. It does not matter if the key specified does not exist in the file, as long as a lesser key exists in the file.

Warning:

If this parameter is supplied, the actual key of the record returned is placed in this field by TIPFCS this facilitates a subsequent call to FCS-PREV. This action, however, alters the field and means that the field must be located in the program's LINKAGE SECTION to permit the program to run as a reentrant process.

index-num

Binary halfword holding the desired index number. If the index-num field is omitted, the default index number for the file is used. The default index is not necessarily the index for the primary key.

dup-count

Binary fullword holding the ordinal number of the desired record in a set of duplicate records. For example: 100 means "return the 100th record". If this field is omitted, only the first record (of a set of duplicates) is accessed. Logically deleted records are not counted when FCS searches for the Nth duplicate record via the dup-count supplied.

Error Conditions:

PIB-STATUS	Meaning
PIB-FUNCTION	The file is not assigned to the program.
PIB-IO-ERROR	An I/O error occurred on the disk
PIB-NOT-FOUND	The previous record does not exist. This occurs if the key specified is the first key or if it precedes all existing keys.

FCS-PUT - Indexed: Rewrite Record

Rewrite (update) a record that was read and "locked for update" by a prior call to TIPFCS with the FCS-GETUP function.

Syntax:

```
CALL "TIPFCS" USING     FCS-PUT
                        file-pkt
                        record
```

Where:

FCS-PUT
Function code from the TC-FCS copybook.

file-pkt
Logical file name packet.

record
Record area containing the record contents.

Error Conditions:

PIB-STATUS	Meaning
PIB-FUNCTION	The file is not assigned to the program.
PIB-IO-ERROR	An I/O error occurred on the disk.
PIB-NOT-HELD	The primary key for the record is not currently in the TIP key holding table. This may be a result of not issuing a prior FCS-GETUP to lock the record for update or the previously acquired record lock was discarded by TIP (see discussion of record locking techniques).
PIB-WRONG-MODE	Write operations are not permitted for the file.

Additional Considerations:

- The FCS-PUT function may be issued while the file is in sequential mode without affecting the current sequential position.

FCS-SETL - Indexed: Set Sequential Mode

The FCS-SETL function sets a file in sequential processing mode beginning with the first record with a key *greater than or equal to* the key supplied.

This function does not return a record - it simply establishes a starting point for later sequential reading. Subsequent calls with a FCS-GET-SEQ-NEXT or FCS-GET-SEQ-PREV function retrieve records in sequence in the appropriate direction.

Syntax:

```
CALL "TIPFCS" USING     FCS-SETL
                        file-pkt
                      [ key ]
                      [ index-num ]
                      [ key-len ]
                      [ dup-count ]
```

Where:

FCS-SETL
Function code from the TC-FCS copybook.

file-pkt
Logical file name packet.

key
The value of the key that is to be used to set the sequential position. If this parameter is omitted, a default key of all low values (binary zero) is used. If your intention is to start at the beginning of the file use FCS-SETL-BOF; or use FCS-SETL-EOF for the end of the file.

index-num
Binary halfword holding the index number. If the index-num field is omitted, the default index number for the file is used. The default index is not necessarily the index for the primary key.

key-len
Binary fullword holding the length (in bytes) of a partial key value that is supplied in the key field. Use of this parameter implies that the key value provided is a prefix of the key desired. If this parameter is omitted, TIPFCS assumes that the value supplied as the key is a complete key.

dup-count
Binary fullword holding the ordinal number of the desired record of a duplicate set.

Error Conditions:

PIB-STATUS	Meaning
PIB-FUNCTION	The file is not assigned to the program.
PIB-IO-ERROR	An I/O error occurred on the disk.
PIB-NOT-FOUND	There are no records with a key greater than or equal to the specified key.

FCS-SETL-BOF - Indexed: Set Sequential Mode

The FCS-SETL-BOF function sets a file in sequential processing mode at the beginning of the file according to a specified index; this eliminates the need to perform an FCS-SETL-xx function with a dummy key consisting of low-values.

This function does *not* return a record - it simply establishes a starting point for sequential reading. Subsequent calls with a FCS-GET-SEQ-NEXT function will retrieve records in sequence.

Syntax:

```
CALL "TIPFCS" USING      FCS-SETL-BOF
                         file-pkt
                       [ index-num ]
```

Where:

FCS-SETL-BOF
Function code from the TC-FCS copybook.

file-pkt
Logical file name packet.

index-num
Binary halfword holding the index number. If the index-num field is omitted, the default index number for the file is used. The default index is not necessarily the index for the primary key.

Error Conditions:

PIB-STATUS	Meaning
PIB-FUNCTION	The file is not assigned to the program.
PIB-IO-ERROR	An I/O error occurred on the disk.

FCS-SETL-EOF - Indexed: Set Sequential Mode

The FCS-SETL-EOF function sets a file in sequential processing mode at the end of the file according to a specified index. This facilitates establishing a starting point for reading a file backwards.

This function does *not* return a record - it simply establishes an ending point for sequential reading. Subsequent calls with a FCS-GET-SEQ-PREV function will retrieve records in backward sequence.

Syntax:

```
CALL "TIPFCS" USING     FCS-SETL-EOF
                        file-pkt
                      [ index-num ]
```

Where:

FCS-SETL-EOF
Function code from the TC-FCS copybook.

file-pkt
Logical file name packet.

index-num
Binary halfword holding the index number. If the index-num field is omitted, the default index number for the file is used. The default index is not necessarily the index for the primary key.

Error Conditions:

PIB-STATUS	Meaning
PIB-FUNCTION	The file is not assigned to the program.
PIB-IO-ERROR	An I/O error occurred on the disk.

FCS-SETL-EQ - Indexed: Set Sequential Mode

The FCS-SETL-EQ function sets a file in sequential processing mode beginning with the first record with a key *equal to* the key supplied.

This function does *not* return a record - it simply establishes a starting point for sequential reading. Subsequent calls with a FCS-GET-SEQ-NEXT function retrieve records in sequence.

See also the description of the FCS-ESETL function.

Syntax:

```
CALL "TIPFCS" USING     FCS-SETL-EQ
                        file-pkt
                        key
                      [ index-num ]
                      [ key-len ]
                      [ dup-count ]
```

Where:

FCS-SETL-EQ
Function code from the TC-FCS copybook.

file-pkt
Logical file name packet.

key
Record key. If omitted, processing begins with the first record in the file.

index-num
Binary halfword holding the index number. If the index-num field is omitted, the default index number for the file is used. The default index is not necessarily the index for the primary key.

key-len
Binary fullword holding the length (in bytes) of a partial key value that is supplied in the key field. Use of this parameter implies that the key value is a prefix of the key desired. If this parameter is omitted, TIPFCS assumes that the key value supplied is complete.

dup-count
Binary fullword holding the ordinal number of the desired record in a set of duplicate records. For example: 100 means "return the 100th record". If this field is omitted, only the first record (of a set of duplicates) is accessed. Logically deleted records are not counted when FCS searches for the Nth duplicate record via the dup-count supplied.

Error Conditions:

PIB-STATUS	Meaning
PIB-FUNCTION	The file is not assigned to the program.
PIB-IO-ERROR	An I/O error occurred on the disk.
PIB-NOT-FOUND	The specific record does not exist.

Additional Considerations:

- TIP will return PIB-NOT-FOUND if the record is deleted.

FCS-SETL-GT - Indexed: Set Sequential Mode

The FCS-SETL-GT function sets a file in sequential processing mode beginning with the first record with a key *greater than* the key supplied.

This function does *not* return a record - it simply establishes a starting point for sequential reading. Subsequent calls with a FCS-GET function will retrieve records in sequence.

See also the description of the FCS-ESETL function.

Syntax:

```
CALL "TIPFCS" USING     FCS-SETL-GT
                        file-pkt
                      [ key ]
                      [ index-num ]
                      [ key-len ]
                      [ dup-count ]
```

Where:

FCS-SETL-GT
Function code from the TC-FCS copybook.

file-pkt
Logical file name packet.

key
Record key. If omitted, processing begins with the first record in the file.

index-num
Binary halfword holding the index number. If the index-num field is omitted, the default index number for the file is used. The default index is not necessarily the index for the primary key.

key-len
Binary fullword that holds the length (in bytes) of a partial key value that is supplied in the key field. Use of this parameter implies that the key value is a prefix of the key desired. If this parameter is omitted, TIPFCS assumes that the key value supplied is complete.

dup-count
Binary fullword holding the ordinal number of the desired record in a set of duplicate records. For example: 100 means "return the 100th record". If this field is omitted, only

the first record (of a set of duplicates) is accessed. Logically deleted records are not counted when FCS searches for the Nth duplicate record via the dup-count supplied.

Error Conditions:

PIB-STATUS	Meaning
PIB-FUNCTION	The file is not assigned to the program
PIB-IO-ERROR	An I/O error occurred on the disk.
PIB-NOT-FOUND	There are no records with a key greater than the specified key.

FCS-SKIP - Indexed: Skip Sequentially

The FCS-SKIP function is appropriate only for an indexed file set in sequential mode. A specified number of records are skipped. Subsequent calls with a FCS-GET_SEQ-NEXT function continue at the point where the FCS-SKIP ended.

This function does *not* return a record - it simply establishes a starting point for subsequent sequential reading.

In any case, deleted records are not included in the number of records skipped - FCS-SKIP skips the specified number of non-deleted records.

Syntax:

```
CALL "TIPFCS" USING     FCS-SKIP
                        file-pkt
                        skip-count
```

Where:

FCS-SKIP
Function code from the TC-FCS copybook.

file-pkt
Logical file name packet.

skip-count
Binary fullword holding the number of records to skip. Logically deleted records are not counted when FCS searches for the Nth duplicate record via the dup-count supplied.

Error Conditions:

PIB-STATUS	Meaning

PIB-STATUS	Meaning
PIB-FUNCTION	The file is not assigned to the program.
PIB-WRONG-MODE	The file is not indexed, or is not already in sequential mode.
PIB-IO-ERROR	An I/O error occurred on the disk.
PIB-EOF	End of file reached.

TIPFCS for Direct Files

This section describes TIP file control system operations you may specify for direct access files. Records are referenced by a record number relative to 1. For DAM, the relative record number is also known as the "block number".

Key Passed to TIPFCS

In all cases the key passed to **TIPFCS** is a binary fullword (PIC 9(9) BINARY) that holds the relative record number of the record to be processed.

FCS-ADD - Direct: Add Record

The FCS-ADD function adds a new record to a non-indexed file or rewrites an existing record.

Syntax:

```
CALL "TIPFCS" USING     FCS-ADD
                        file-pkt
                        record
                      [ rel-rec-num ]
```

Where:

FCS-ADD
: Function code from the TC-FCS copybook.

file-pkt
: Logical file name packet.

record
: Record area containing new record data.

rel-rec-num
: Binary fullword containing the relative record number of the record that will be added to the file. If this relative record

number is beyond the current end-of-data (EOD) pointer, TIPFCS writes the record using relative record number EOD+1.

Note: TIPFCS always updates this field to reflect the actual relative record number that was written. For this reason, this field must appear in the program's LINKAGE SECTION to permit re-entrant execution.

Error Conditions:

PIB-STATUS	Meaning
PIB-FUNCTION	The file is not assigned to the program.
PIB-IO-ERROR	An I/O error occurred on the disk.
PIB-HELD	Some other process has the specified record locked.

Additional Considerations:

- The FCS-ADD function rewrites the record if the specified record number already exists in the file. The record is rewritten and is journalized (if required) as a new record.
- Using FCS-ADD to rewrite records is in direct conflict with standard record locking facilities - some race conditions may occur if this technique is employed. The user program must ensure that the race conditions are not a problem. A popular technique is to perform a conventional FCS-GETUP on a control record before issuing such FCS-ADD operations. In this way programs essentially use the FCS-GETUP on the control record as a queuing mechanism.

FCS-CLOSE - Direct: Close File

The FCS-CLOSE function call indicates that a program is relinquishing access to a file. TIP removes the corresponding entry for the file from the Active File Table (AFT) of the issuing process.

Syntax:

```
CALL "TIPFCS" USING     FCS-CLOSE
                        file-pkt
```

Where:

FCS-CLOSE
Function code from the TC-FCS copybook.

file-pkt
Logical file name packet

Error Conditions:

PIB-STATUS	Meaning
PIB-FUNCTION	File is not assigned to the program.

Additional Considerations:

- This function is used for files that were accessed by issuing an explicit FCS-OPEN function.

FCS-DELETE - Direct: Delete Record

The FCS-DELETE function call deletes a record from the file. FCS uses the applicable delete scheme as specified in the TIP definition for the file.

A separate section of this chapter provides details about the two delete schemes (see references to "DELETE").

Before issuing this function call the program **must** first acquire the record with an update lock by issuing a prior call with the FCS-GETUP function.

Syntax:

```
CALL "TIPFCS" USING     FCS-DELETE
                        file-pkt
                        record
                      [ rel-rec-num ]
```

Where:

FCS-DELETE
Function code from the TC-FCS copybook.

file-pkt
Logical file name packet.

record
Record area.

rel-rec-num
Binary fullword that contains the relative record number of the record that is to be deleted. If you omit this parameter, the default is the last record number successfully referenced by the process.

Error Conditions:

PIB-STATUS	Meaning
PIB-FUNCTION	The file is not assigned to the program.
PIB-IO-ERROR	An I/O error occurred on the disk.

PIB-STATUS	Meaning
PIB-NOT-HELD	A prior FCS-GETUP was not successfully done for this record or the record lock was released.

FCS-FLUSH - Direct: Flush File

The FCS-FLUSH is not used by TIP and is provided for compatibility with the System/80 TIP/30 product.

Syntax:

```
CALL "TIPFCS" USING     FCS-FLUSH
                        file-pkt
```

Where:

FCS-FLUSH
: Function code from the TC-FCS copybook.

file-pkt
: Logical file name packet.

Error Conditions:

PIB-STATUS	Meaning
PIB-FUNCTION	The file is not assigned to the program.
PIB-IO-ERROR	An I/O error occurred on the disk.

FCS-GET - Direct: Read Record

Read a specific record from a direct file. The record is *not* locked for update.

Syntax:

```
CALL "TIPFCS" USING     FCS-GET
                        file-pkt
                        record
                        rel-rec-num
```

Where:

FCS-GET
: Function code from the TC-FCS copybook

file-pkt
: Logical file name packet.

record
Area where record data is to be placed.

rel-rec-num
Binary fullword containing the relative record number of the record to read.

Error Conditions:

PIB-STATUS	Meaning
PIB-EOF	The requested record is beyond the last record in the file.
PIB-FUNCTION	The file is not assigned to the program.
PIB-IO-ERROR	An I/O error occurred on the disk.
PIB-NOT-FOUND	The record does not exist or it is flagged deleted using a logical delete flag. In the latter case, the record data is returned to the program even though PIB-NOT-FOUND is set.

FCS-GETUP - Direct: Read With Lock

Read the record with the specified relative record number with intent to update. The relative record number (and the filename information) is placed in the TIP internal key holding table. The record is LOCKED - other processes receive an error status if they attempt to FCS-GETUP or FCS-ADD a record for this file with the same relative record number.

Syntax:

```
CALL "TIPFCS" USING     FCS-GETUP
                        file-pkt
                        record
                        rel-rec-num
```

Where:

FCS-GETUP
Function code from the TC-FCS copybook.

file-pkt
Logical file name packet.

record
Area where the record data is placed.

rel-rec-num
Binary fullword that contains the relative record number of the record to be read.

Error Conditions:

PIB-STATUS	Meaning
PIB-FUNCTION	The file is not assigned to the program.
PIB-IO-ERROR	An I/O error occurred on the disk.
PIB-NOT-FOUND	The record does not exist or is flagged as logically deleted.
PIB-HELD	The record is currently locked by some other process in the TIP system.

Additional Considerations:

- If the program receives the error status "PIB-HELD", the program probably should retry the FCS-GETUP function (possibly after a brief delay via TIPTIMER). The number of times the retry is attempted is application-dependent, after some number of retries, consider some alternate action.

FCS-NOUP - Direct: Cancel Update

You may use the FCS-NOUP function call to unlock a record that was locked via a previous call to FCS-GETUP *provided that the record has not been updated*.

In certain situations, a program may issue a FCS-GETUP and lock a record only to later determine that an update is not appropriate.

Syntax:

```
CALL "TIPFCS" USING     FCS-NOUP
                        file-pkt
                      [ rel-rec-num ]
```

Where:

FCS-NOUP
Function code from the TC-FCS copybook.

file-pkt
logical file name packet.

rel-rec-num
Binary fullword containing the relative record number of the specific record to be released. If omitted, all records currently held by this process for the specified file are released.

Error Conditions:

PIB-STATUS	Meaning
PIB-FUNCTION	The file is not assigned to the program.
PIB-NOT-HELD	The record was not held.
PIB-EOF	The record number is not valid.

FCS-OPEN - Direct: Open File

Make the specified file available for processing by programs at the calling terminal. TIP creates an entry in the Active File Table (AFT) for the process issuing this call.

This function is needed only for files which are not implicitly opened as result of the FILES= keyword in the program's TIP definition entry.

Syntax:

```
CALL "TIPFCS" USING     FCS-OPEN
                        file-pkt
                      [ file-desc ]
```

Where:

FCS-OPEN
Function code from the TC-FCS copybook.

file-pkt
Logical file name packet.

file-desc
File descriptor packet - see separate description of the copybook TC-FDES. If this parameter is omitted, TIP uses the name in the FILE-PKT to build a file descriptor.

Error Conditions:

PIB-STATUS	Meaning
PIB-IO-ERROR	An I/O error occurred while opening the file.
PIB-DUP-AFT-NAME	An entry already exists in the Active File Table (for the issuing process) that matches the logical file name used in the FILE-PKT field.
PIB-LOCKED	The file is closed.
PIB-NOT-FOUND	The logical file name is not defined in the

PIB-STATUS	Meaning
	active groups for this TIP session.
PIB-FUNCTION	The open function could not be performed. Check Unix permissions.

FCS-PUT - Direct: Update Record

Update (rewrite) a record obtained by a previous FCS-GETUP.

Syntax:

```
CALL "TIPFCS" USING     FCS-PUT
                        file-pkt
                        record
                      [ rel-rec-num ]
```

Where:

FCS-PUT
Function code from the TC-FCS copybook.

file-pkt
Logical file name packet.

record
Record area containing the new record contents.

rel-rec-num
Binary fullword that contains the relative record number of the record TIPFCS is to update. If this parameter is omitted, the default is the last record number referenced by the process for this file.

Error Conditions:

PIB-STATUS	Meaning
PIB-FUNCTION	The file is not assigned to the program.
PIB-IO-ERROR	An I/O error occurred on the disk.
PIB-NOT-HELD	The relative record number is not currently in the TIP key holding table. This may be a result of not issuing a prior FCS-GETUP to lock the record for update or TIP has discarded the previously acquired record lock (see discussion of record locking elsewhere in this manual).

TIPFCS for Sequential Files

TIPFCS for Sequential Files

This section describes TIP file control system operations for sequential files. Sequential files include:

- Sequential
- PRINT
- TAPE.

A sequential file must be designated in the TIP file definition as either an INPUT or OUTPUT file (INOUT is not available for sequential processing).

Sequential files may not be assigned to a program by specifying the filename in the program's TIP catalogue entry. Sequential files must be explicitly opened and closed by the program by issuing FCS-OPEN and FCS-CLOSE function CALLs to the TIP File Control System (TIPFCS).

The operating system's spooling facilities normally process printer files. This spooling activity is transparent to FCS.

Print files use a standard variable length print line. The layout of the print line is the same as the layout required by the TIP printing interface "TIPPRINT". (See the TIPPRINT section of this document or the copy book TC-PLINE).

FCS-CLOSE - Sequential: Close File

The FCS-CLOSE function indicates that a program is relinquishing access to a file. TIP removes the corresponding entry for the file from the Active File Table (AFT) of the issuing process.

If there are no other on-line users of the file *and* the file was generated with OPEN=NO, TIPFCS will physically CLOSE the file by issuing a "CLOSE".

Syntax:

```
CALL "TIPFCS" USING     FCS-CLOSE
                        file-pkt
```

Where:

FCS-CLOSE
Function code from the TC-FCS copybook.

file-pkt
Logical file name packet.

Error Conditions:

PIB-STATUS	Meaning
PIB-FUNCTION	File is not assigned to the program.

Additional Considerations:

- Issue this function only for files that were opened by issuing a FCS-OPEN function (files explicitly accessed by the program).

FCS-GET - Sequential: Read Record

Read the next record from a sequential input file.

Attempts to have more than one program simultaneously read the same input file can result in interleaved read operations (each program will "miss" whatever records the other programs read).

Furthermore, there is no provision for specifying a particular starting position - an FCS-GET issued for a sequential file obtains the next record in the file - regardless of who read the last record.

For this reason, it is recommended that sequential input files be declared as OPEN=NO in the TIP definition for the file and steps be taken to ensure that only one program reads the file at a time.

One way to do this is to make use of the TIPFLAGS subroutine (see documentation for the TIPFLAGS subroutine in the PCS section of this manual) or by using the FCS-HOLD and FCS-RELEASE function CALLs of TIPFCS.

Syntax:

```
CALL "TIPFCS" USING     FCS-GET
                        file-pkt
                        record
```

Where:

FCS-GET
Function code from the TC-FCS copybook.

file-pkt
Logical file name packet.

record
Area where record data is to be placed

Error Conditions:

PIB-STATUS	Meaning
PIB-FUNCTION	The File is not assigned to the program.
PIB-IO-ERROR	An I/O error occurred on the disk.
PIB-EOF	End of file has been reached.
PIB-WRONG-MODE	File is not defined as an input file.
PIB-NOT-FOUND	The record does not exist or is flagged deleted using a logical delete flag. If the record is logically deleted, the record data is returned in the specified record area

FCS-OPEN - Sequential: Open File

Make the specified file available for processing by programs at the calling terminal. TIP creates an entry in the Active File Table (AFT) for the process issuing this call.

If there are no other users of the file and the file was specified in the TIP definition as "OPEN=NO", TIPFCS will physically OPEN the file.

For the FCS-OPEN function to be successful, the file to be opened must be:

- defined in the TIP Catalogue (this is where the connection is made between a logical file name (LFN) and the physical file name (LFD)
- defined in the TIP FILE definition.

Syntax:

```
CALL "TIPFCS" USING     FCS-OPEN
                        file-pkt
                      [ file-desc ]
```

Where:

FCS-OPEN
: Function code from the TC-FCS copybook.

file-pkt
: Logical file name packet.

file-desc
: File descriptor packet - see separate description of the copybook "TC-FDES".If omitted, the name in the FILE-PKT parameter is used to build a file descriptor.

Error Conditions:

PIB-STATUS	Meaning
PIB-IO-ERROR	An I/O error occurred while opening the file.
PIB-DUP-AFT-NAME	An entry already exists in the Active File Table (for the issuing process) that matches the logical file name used in the FILE-PKT field.

FCS-PUT - Sequential: Write A Record

Write a record to a sequential output file.

FCS permits multiple concurrent writers for an output sequential file. Each program appends a new record to the file - in other words, the write operations might be interleaved.

Syntax:

```
CALL "TIPFCS" USING      FCS-PUT
                         file-pkt
                         record
```

Where:

FCS-PUT
Function code from the TC-FCS copybook.

file-pkt
Logical file name packet.

record
Record area containing data for the record to be added.

Note: If the output file is a printer file (generation type "PRINT"), the first 5 bytes of the record must be a properly constructed header containing the length of the record area and the printer spacing control code. See the description of the structure of print line records in the section describing TIPPRINT and the supplied copybook TIP/TC-PLINE.

Error Conditions:

PIB-STATUS	Meaning
PIB-FUNCTION	The File is not assigned to the program.
PIB-IO-ERROR	An I/O error occurred on the disk.
PIB-EOF	The file is full and cannot be extended.
PIB-WRONG-MODE	The file is not defined as on output file, or for FCS-PUT to a printer file, the printer spacing code (in the 5 byte header) is not a valid spacing code.

TIPFCS for Dynamic Files

TIP supports a file organization known as a "dynamic file". Dynamic files have the following characteristics:

- Dynamic files may be created and scratched on demand by TIP programs.
- Record size is fixed at 512 bytes.
- Records are referenced by a relative record number (in a similar manner as a direct access file).
- Dynamic File names consist of three sections (each name may be up to eight characters long) — an example is: EDP/BATCH/007 TIP edit buffers are stored in the $TIPROOT directory under the directory name "tipfiles/dynamic". The first two names of a dynamic file represent further directory names under "tipfiles/dynamic". The final (3rd) part of the dynamic file name are individual file names there under. For example, a dynamic file named "EDP/TEST/PAY020" appears in the Unix file:
  `$TIPROOT/tipfiles/dynamic/EDP/TEST/PAY020`

Note: The dynamic file name is forced to all uppercase.

- Programs can dynamically create records in any sequence desired; for example, if only 40 records exist at the moment and the program specifies a read or a write of record 87, the file system will allocate more blocks for the file and then access block (record) 87.
- To allow maximum flexibility, TIPFCS allows the program to read or write multiple (sequential) records with a single operation. This, for example, allows a program to simulate a record size of 1024 by always writing two records at a time - blocks 1 and 2, then blocks 3 and 4, and so on.
- Dynamic files may be created as "permanent" or "temporary" files - temporary files are automatically scratched when the program terminates; programs must explicitly scratch permanent dynamic files.

Dynamic File Functions:

Dynamic files support the following functions (the function names refer to function codes defined by the copy book TC-FCS).

Function	Description
FCS-ACCESS	Open an existing file.
FCS-ASSIGN	Open file; create if necessary.
FCS-CLOSE	Close a file.
FCS-CREATE	Create a new file.
FCS-GET	Read record(s) from the file.
FCS-OPEN	Open file (choice of ACCESS, ASSIGN or CREATE).
FCS-PUT	Write record(s) to the file.
FCS-SCRATCH	Scratch a file.

FCS-ACCESS - Dynamic: Access File

Before an application program can perform I/O to an *existing* dynamic file, the file must be assigned to the program. Use the function FCS-ACCESS to open an *existing* dynamic file.

Syntax:

```
CALL "TIPFCS" USING     FCS-ACCESS
                        file-pkt
                        file-desc
```

Where:

FCS-ACCESS
Function code from the TC-FCS copybook.

file-pkt
Logical file name packet.

file-desc
File descriptor packet. See the description of the copybook TC-FDES.

Example of Accessing an Existing Dynamic File:

To access an existing dynamic file named: EDP/TAX/TABLES (for read-only operations):

```
02  TAXTABLE-LFN        PICTURE X(9).
```

```
02  TAXTABLE-FDES.       COPY TC-FDES.
     ...
    MOVE "TAXTABLE"      TO TAXTABLE-LFN
    MOVE "EDP"           TO FDES-user id
    MOVE "TAX"           TO FDES-CATALOG
    MOVE "TABLES"        TO FDES-FILE-NAME
    MOVE SPACES          TO FDES-PASSWORD
    MOVE FCS-CLASS-PERM  TO FDES-FCS-CLASS
    MOVE SPACE           TO FDES-FCS-TYPE
    MOVE FCS-PERM-READONLY
                         TO FDES-FCS-PERM
    MOVE FCS-LOCK-NO     TO FDES-FCS-LOCK
    CALL "TIPFCS" USING  FCS-ACCESS
                         TAXTABLE-LFN
                         TAXTABLE-FDES
```

Error Conditions:

PIB-STATUS	Meaning
PIB-DUP-AFT-NAME	A file with the logical file name specified in the FILE-PKT parameter is already present in the active file table (AFT) for the process.
PIB-NOT-FOUND	The requested file does not exist.

FCS-ASSIGN - Dynamic: Assign File

This FCS function will assign an existing Dynamic file for use by the calling program. If the file does not exist, TIPFCS will automatically create a new file according to the specifications given in the FILE-DESCRIPTOR packet.

Syntax:

```
CALL "TIPFCS" USING     FCS-ASSIGN
                        file-pkt
                        file-desc
```

Where:

FCS-ASSIGN
Function code from the TC-FCS copybook.

file-pkt
Logical file name packet.

file-desc
File descriptor packet. See the description earlier of the copybook TC-FDES.

Error Conditions:

PIB-STATUS	Meaning
PIB-DUP-AFT-NAME	A file with the logical file name specified in the FILE-PKT parameter is already present in the active file table (AFT) for the process.

FCS-CLOSE - Dynamic: Close File

When an application program is finished with a dynamic file it should remove the file from the Active File Table by issuing a FCS-CLOSE. If the program created a dynamic file as a "temporary" file, this operation will scratch the file. If the file was created as a permanent dynamic file, the FCS-CLOSE operation only removes the file from the Active File Table.

Syntax:

```
CALL "TIPFCS" USING      FCS-CLOSE
                         file-pkt
```

Where:

FCS-CLOSE
Function code from the TC-FCS copybook.

file-pkt
Logical file name packet.

Error Conditions:

PIB-STATUS	Meaning
PIB-FUNCTION	The file is not assigned to the program.

FCS-CREATE - Dynamic Create File

This function creates new dynamic files, either temporary or permanent. The application program must first fill in the fields of the FILE-DESCRIPTOR with appropriate values.

If the field FDES-user id is spaces or low-values, TIPFCS will use the user id from the PIB (PIB-UID).

If the field FDES-CATALOG is SPACES or low-values, TIPFCS will construct a unique name consisting of the terminal-id (PIB-TID) and program execution stack level (PIB-LEVEL).

Set the field FDES-FCS-TYPE to the value FCS-TYPE-NEW.

Set FDES-FCS-CLASS to FCS-CLASS-PERM or FCS-CLASS-TEMP to create a permanent or temporary file.

Set FDES-FCS-LOCK to FCS-LOCK-YES or FCS-LOCK-NO to indicate whether the program desires exclusive use of this dynamic file.

Syntax:

```
CALL "TIPFCS" USING     FCS-CREATE
                        FILE-PKT
                        file-desc
```

Where:

FCS-CREATE
Function code from the TC-FCS copybook.

FILE-PKT
Logical file name packet

file-desc
File descriptor packet. See the earlier description of the copybook TC-FDES.

Error Conditions:

PIB-STATUS	Meaning
PIB-DUP-AFT-NAME	A file of the name given in the FILE-PKT is already assigned to the process.
PIB-NOT-FOUND	The requested file already exists.

FCS-GET - Dynamic: Read Record(s)

Records in FCS Dynamic files are referenced by relative (to 1) record number. The program specifies a relative record number (as a fullword) to read.

If the optional parameter REC-COUNT is specified, FCS reads that many records (starting with the relative record indicated by REC-NUM) into the record area.

If the optional parameter REC-COUNT is *not* specified, FCS reads a single record into the record area specified.

You must fullword align the record area; it must also be large enough to accommodate the number of records requested by REC-COUNT (that is, REC-COUNT * 512 bytes).

If a requested record is beyond the current allocation of blocks, TIPFCS will allocate more blocks to the file, up to the maximum allowable limit for a dynamic file.

When TIPFCS returns blocks to the calling program, data in the blocks *is not initialized*; the program must take responsibility for the contents of the blocks.

Syntax:

```
CALL "TIPFCS" USING     FCS-GET
                        file-pkt
                        record
                        rec-num
                      [ rec-count ]
```

Where:

FCS-GET
Function code from the TC-FCS copybook.

file-pkt
Logical file name packet.

record
Record area of (512 x REC-COUNT) bytes. This area must be fullword aligned.

rec-num
Binary fullword that specifies the relative record number of the first record to read.

rec-count
Optional fullword that specifies how many records to read. Default is one record.

Error Conditions:

PIB-STATUS	Meaning
PIB-EOF	The requested record is beyond the last record in the file.
PIB-FUNCTION	The file is not assigned to the program.
PIB-IO-ERROR	An I/O error occurred on the disk.

Example:

```
05  LFN-PKT                 PICTURE X(9).
```

```
05  REL-REC-NUM                PICTURE 9(8)
                               COMP SYNC.
05  REC-COUNT                  PICTURE 9(8)
                               COMP SYNC.
05  DYN-REC.
    10  FILLER                 PICTURE 9(8)
                               COMP SYNC.
    10  FILLER                 PIC X(1020).
          ... ...
    MOVE 1                     TO REL-REC-NUM
    MOVE 2                     TO REC-COUNT
    CALL "TIPFCS" USING        FCS-GET
                               LFN-PKT
                               DYN-REC
                               REL-REC-NUM
                               REC-COUNT
```

In the example above, the program must next increment the REL-REC-NUM field by REC-COUNT to read the next set of records.

FCS-OPEN - Dynamic: Open File

Use the FCS-OPEN function to open *any* dynamic file. The file descriptor supplied with the call and any existing TIP record information is used to determine what type of file is to be opened:

To open an existing file set FDES-FCS-TYPE to FCS-TYPE-OLD. If the FDES-FCS-TYPE is left as a space and the file exists, it is opened. If the file does not exist, it is created.

To create a new file set FDES-FCS-TYPE to FCS-TYPE-NEW.

Thus, depending on the values set in the file descriptor, FCS-OPEN can perform the same functions as FCS-ACCESS, FCS-ASSIGN and FCS-CREATE.

Syntax:

```
CALL "TIPFCS" USING        FCS-OPEN
                           file-pkt
                           file-desc
```

Where:

FCS-OPEN
Function code from the TC-FCS copybook.

file-pkt
Logical file name packet.

file-desc
File descriptor packet. See the earlier description of the copybook TC-FDES.

Error Conditions:

PIB-STATUS	Meaning
PIB-DUP-AFT-NAME	A file of the name given in the FILE-PKT is already assigned to the process.
PIB-NOT-FOUND	The requested file does not exist.

FCS-PUT - Dynamic: Write Record(s)

Dynamic file records are a fixed size of 512 bytes. The FCS-PUT function allows the program to write one or more records (in sequence) to a dynamic file.

If a RECORD-NUMBER is specified that is beyond the current limit of the file, FCS will expand the file to accept that record up to the maximum file size allowed for a dynamic file.

Syntax:

```
CALL "TIPFCS" USING     FCS-PUT
                        file-pkt
                        record
                        rec-num
                      [ rec-count ]
```

Where:

FCS-PUT
Function code from the TC-FCS copybook.

file-pkt
Logical file name packet.

record
Record area. The size of this area must be 512 bytes times the value of REC-COUNT. This field must be fullword aligned.

rec-num
Binary fullword that specifies the relative record number of the first record to be written.

rec-count
Optional fullword that specifies the number of records to be written. Default is one record.

Error Conditions:

PIB-STATUS	Meaning
PIB-EOF	The requested records are invalid.
PIB-FUNCTION	The file is not assigned to the program.
PIB-IO-ERROR	An I/O error occurred on the disk.

FCS-SCRATCH - Dynamic: Scratch File

The FCS-SCRATCH function deletes either "temporary" or "permanent" dynamic files from the FCS system.

A file must be assigned before it can be scratched. Temporary Dynamic files are automatically scratched if TIP terminates abnormally or if the transaction aborts.

Syntax:

```
CALL "TIPFCS" USING     FCS-SCRATCH
                        file-pkt
```

Where:

FCS-SCRATCH
Function code from the TC-FCS copybook.

file-pkt
Logical file name packet

Error Conditions:

PIB-STATUS	Meaning
PIB-FUNCTION	The file is not assigned to the program.
PIB-WRONG-MODE	The file is not a dynamic file.

TIPFCS for Edit Buffers

TIPFCS for Edit Buffers

The text editors supplied with TIP do all editing in a special purpose file called an "edit buffer". Edit buffers are named files containing lines of text. Each line is referenced by relative line number, starting with 1.

If a record is deleted all following records move up (their line number decreases by one). If a record is added all following records move down (their line number increases by one).

TIP uses a two part name to name edit buffers. Each part of the name may be from one to 8 characters. For example:

```
EDP/PAY020
```

The first part of the name is normally determined by the group membership of the user who creates the edit buffer. This is the assumption made by the TIP text editors; however, this is not a hard and fast rule. TIP edit buffers are stored in the $TIPROOT directory under the directory name "tipfiles/dynamic". Each user group represents a further directory name under "tipfiles/dynamic" and buffers are individual file names there under. For example, an edit buffer named "EDP/PAY020" appears in the Unix file:
$TIPROOT/tipfiles/dynamic/EDP/PAY020

Notice that the group name (EDP) and the buffer name (PAY020) are forced to all uppercase.

FCS-ADD - Edit: Add/Insert Line

The FCS-ADD function adds or inserts a new record to an edit buffer.

Syntax:

```
CALL "TIPFCS" USING     FCS-ADD
                        file-pkt
                        record
                        line-num
```

Where:

FCS-ADD
Function code from the TC-FCS copybook.

file-pkt
Logical file name packet.

record
Record area.

line-num
Binary fullword holding the relative record number that is to be added.

The supplied record becomes the new contents of the specified line number. All records that follow this line number will have their record number increased by 1.

If this field contains a value that is higher than the current last line number, this record is added at the end of the edit buffer and TIPFCS modifies the field to reflect the resulting actual line number of the added record.

Error Conditions:

PIB-STATUS	Meaning
PIB-EOF	The line number is invalid.
PIB-FUNCTION	The edit buffer is not assigned to the process.

Additional Considerations:

- The record is written to the file at the specified position. TIP shifts any records currently at that position or higher to the next higher position by altering the index to reflect their new logical position in the file.

FCS-CLOSE - Edit: Close Buffer

The FCS-CLOSE function closes an edit buffer and removes the entry for the edit buffer from the Active File Table (AFT) of the process.

Before issuing this call, the program must be certain to issue an FCS-FLUSH function (see description of this function), otherwise some changes to the edit buffer may not be written to the disk.

Syntax:

```
CALL "TIPFCS" USING     FCS-CLOSE
                        file-pkt
```

Where:

FCS-CLOSE
Function code from the TC-FCS copybook.

file-pkt
Logical file name packet.

Error Conditions:

PIB-STATUS	Meaning
PIB-FUNCTION	The edit buffer is not assigned to the process.

FCS-DELETE - Edit: Delete Line

The FCS-DELETE function deletes a line from an edit buffer.

Syntax:

```
CALL "TIPFCS" USING     FCS-DELETE
                        file-pkt
                        record
                        line-num
```

Where:

FCS-DELETE
Function code from the TC-FCS copybook.

file-pkt
Logical file name packet.

record
Record area.

This parameter is a dummy parameter to maintain symmetry with other calls to TIPFCS.

line-num
Binary fullword holding the relative line number to be deleted.

Error Conditions:

PIB-STATUS	Meaning
PIB-FUNCTION	The edit buffer is not assigned to the process.
PIB-EOF	The line number is invalid.

Additional Considerations:

- TIP deletes the record at the specified position from the file. Any records with a higher line number are shifted down one line number by changing the index to reflect their new logical position in the file.

Note: The specification of a line number that is out of bounds (for example, past end of file) will *not* result in an error status!

FCS-FLUSH - Edit: Flush Buffer

TIPFCS assumes responsibility for the maintenance of the index for an edit buffer. Updated blocks are not written to disk unless TIPFCS determines that they need to be written to make space in the I/O buffer that is maintained in memory.

Edit buffers are implemented as Windows NT memory mapped files. The operating system will page the file in and out of memory as required. The FCS-FLUSH command is used by Windows NT to flush any updated areas of the file to disk immediately. There is no need to issue a FCS-FLUSH prior to issuing a CLOSE. The TIP Application Server will do this automatically.

Syntax:

```
CALL "TIPFCS" USING     FCS-FLUSH
                        file-pkt
```

Where:

FCS-FLUSH
Function code from the TC-FCS copybook.

file-pkt
Logical file name packet.

Error Conditions:

PIB-STATUS	Meaning
PIB-FUNCTION	The edit buffer is not assigned to the process.

FCS-GET - Edit: Read Line

The FCS-GET function reads a line from an edit buffer.

Syntax:

```
CALL "TIPFCS" USING     FCS-GET
                        file-pkt
                        record
                        line-num
```

Where:

FCS-GET
Function code from the TC-FCS copybook.

file-pkt
Logical file name packet.

record
Record area. This area must be large enough to hold a record from the edit buffer.

line-num
Binary fullword holding the relative line number to read.

This parameter is optional. If omitted, you will receive the next line.

Error Conditions:

PIB-STATUS	Meaning
PIB-EOF	The line number is invalid.
PIB-FUNCTION	The edit buffer is not assigned to the process.

FCS-OPEN - Edit: Open Buffer

Edit buffers are "line-oriented" in the sense that they manipulate "lines" of data. The most common implementation of edit buffers (used by TIP editors) specifies a line length of 85 characters. TIP always references the lines in an edit buffer by positive whole numbers that range from one in increments of one.

TIP reserves bytes 82 through 85. Records in an edit buffer are accessed by a line number *relative to one*.

Syntax:

```
CALL "TIPFCS" USING     FCS-OPEN
                        file-pkt
                        file-desc
                      [ buffer ]
                      [ num-buffers ]
```

Where:

FCS-OPEN
Function code from the TC-FCS copybook.

file-pkt
Logical file name packet.

file-desc
File descriptor packet (see copybook TC-FDES). The example that follows illustrates additional details.

buffer
A work area that TIPFCS may use as an I/O buffer for the file. If this parameter is omitted, FCS attempts to allocate a

dynamic buffer (of 1536 bytes) from the TIP free memory pool.

Note: The second halfword of this buffer always contains a binary number representing the number of lines currently in the edit buffer. The user program must never modify the contents of this buffer - only TIPFCS is intended to access this buffer.

num-buffers

A halfword that indicates the number of 512 byte data blocks that immediately follow the mandatory initial index block in the "buffer" specified in the previous parameter.

Minimum specified value is two (implying that "buffer" is 512 + (2*512) bytes

Maximum specified value is 12 (implying that "buffer" is 512 + (12*512) bytes

The larger the number of data blocks allocated in the buffer, the more potential work can be accomplished in memory (rather than performing disk I/O).

line-length

A halfword containing the desired line length for this edit buffer.

Range: 64 through 512 bytes inclusive.

Default value (if parameter is omitted or is out of the allowable range) is 81. The TIP editors default to creating edit buffers that have a line length of 81 characters (80 bytes of user data plus 1 byte of control information).

Error Conditions:

PIB-STATUS	Meaning
PIB-DUP-AFT-NAME	The logical file name is already in use by the process.
PIB-NOT-FOUND	The edit buffer is not assigned to the process.
PIB-IO-ERROR	An I/O error occurred while opening the file.

Example:

```
... in the program's WORKING-STORAGE ...
77  NUM-BUFFERS                 PICTURE 9(3)
                                BINARY VALUE 3.
 ... in the program's work area ...
02  EDIT-BUF-DESC.              COPY TC-FDES.
```

```
       05  EDIT-BUF-LFN            PICTURE X(9).
       05  EDIT-WORKAREA.
           10  EDIT-BUFFER-WORD1   PICTURE 9(8)
                                   BINARY.
           10 EDIT-BUFFER-WORD1R REDEFINES
                                   EDIT-BUFFER-WORD1.
               15  FILLER          PICTURE X(2).
               15  EDIT-LINES      PICTURE 9(4)
                                   BINARY.
           10  FILLER              PICTURE X(508).
           10  FILLER              PICTURE X(512).
           10  FILLER              PICTURE X(512).
           10  FILLER              PICTURE X(512).
    ... in the PROCEDURE DIVISION ...
       MOVE "WORKFIL"               TO EDIT-BUF-LFN
       MOVE "EDP"                   TO FDES-user id
       MOVE "SOMEDATA"              TO FDES-FILE-NAME
       MOVE SPACES                  TO FDES-PASSWORD
       MOVE FCS-CLASS-QED           TO FDES-FCS-CLASS
       MOVE SPACE                   TO FDES-FCS-TYPE
                                       FDES-FCS-PERM
                                       FDES-FCS-LOCK
       CALL "TIPFCS" USING          FCS-OPEN
                                    EDIT-BUF-LFN
                                    EDIT-BUF-DESC
                                    EDIT-WORKAREA
                                    NUM-BUFFERS
```

This example opens an edit buffer named "EDP/SOMEDATA". Since FDES-FCS-TYPE is space, it will either open an existing edit buffer or (if necessary) create one by that name.

Note: In the definition of the edit work area the first word is defined as a binary synchronized item to force proper alignment of the group item! The second halfword is redefined to allow interrogation of the number of lines in the edit buffer.

The stated number of buffers is three; therefore, three filler items that are 512 bytes each follow the first block of 512 bytes.

FCS-PUT - Edit: Replace Line

The FCS-PUT function **replaces** or rewrites an existing line in an edit buffer.

Syntax:

```
CALL "TIPFCS" USING      FCS-PUT
                         file-pkt
                         record
                         line-num
```

Where:

FCS-PUT
Function code from the TC-FCS copybook.

file-pkt
Logical file name packet.

record
Record area.

line-num
Binary fullword holding the relative record number to be replaced.

Error Conditions:

PIB-STATUS	Meaning
PIB-EOF	The line number is out of bounds.
PIB-FUNCTION	The edit buffer is not assigned to the process.

FCS-SCRATCH - Edit: Scratch Buffer

Use the FCS-SCRATCH function to erase (scratch) an edit buffer that has already been opened by the program.

Syntax:

```
CALL "TIPFCS" USING     FCS-SCRATCH
                        file-pkt
```

Where:

FCS-SCRATCH
Function code from the TC-FCS copybook.

file-pkt
Logical file name packet identifying the edit buffer.

Error Conditions:

PIB-STATUS	Meaning
PIB-FUNCTION	The edit buffer is not assigned to the process.

Additional Considerations:

- TIP creates edit buffers as "permanent" files - this prevents their disappearance if a system crash occurs. Since they are permanent files, they must be explicitly scratched to erase them.

TIPFCS for Library Files

TIP implements a "library" access method that is functionally compatible with mainframe libraries. A mainframe library is a *partitioned data set* that contains *elements* or *members*. Each member contains data that is stored in fixed length lines (usually simple text data).

In the TIP implementation, each named element is a separate Unix file (the element name is used as the filename). Element names are limited to 8 characters (for compatibility with the mainframe).

Libraries are defined as logical files in the TIP catalog by using the **smfile** program to associate a library name (again 8 character limit) with a particular Unix file directory name. Elements created within that "library" appear as file within that directory.

When a library element is opened for reading, the corresponding Unix file is opened for input. When a library element is opened for writing, a new, empty temporary file is created and all data is written to this file. When the user subsequently closes this temporary file, it will replace the library element (file). Users currently accessing the element are not affected. This is analogous to the mainframe scheme where the latest element is flagged “active” and the old element is flagged “deleted”.

In TIP/30 it was only possible to have one element in a library open for output at a time. In TIP it is possible to have many elements in a library open for output. Only one user may have any particular element open for output at time.

Library elements contain lines of data that are up to 128 bytes in length. TIP assumes that the record area that is designated by the program for read or write operations is 128 bytes long. Lines are physically stored with trailing spaces removed and a carriage-return and linefeed character appended to each line.

TIP supports the following function codes for library elements:

Function	Description
FCS-OPEN	Open library element.
FCS-CLOSE	Close library element. If reading: de-access file. If writing: the old element is replaced by the latest element, this activates the latest element.
FCS-GET	Get next input record (line).
FCS-PUT	Write next output record (line).
FCS-NOUP	Close library element and cancel update. If reading: de-access file. If writing: old element is unchanged, latest element is removed, the latest

Function	Description
	element is never activated.

Library File Descriptor

The layout of the FILE-DESCRIPTOR packet for library files is described in the copybook TC-FDES in the TIP library:

TC-FDES copybook

The following is a description of the fields in the TC-FDES copybook:

FDES-user id

The group name (or user name) associated with the TIP definition for the library file.

If this field is spaces or low-values, TIPFCS will perform a "standard order of search" for the correct definition entry to reference.

FDES-CATALOG

Logical file name for the library.

This field normally contains the same value as the following field (FDES-FILE-NAME), although the library open routines will accept a logical file name in either this field or the next.

If both fields are empty, the name in the file name packet, file-pkt, is used on the FCS-OPEN call.

FDES-FILE-NAME

Logical file name for the library.

FDES-PASSWORD

This field is not implemented for the library access method and is ignored.

FDES-FCS-CLASS

This field is not implemented for the library access method and is ignored.

FDES-FCS-TYPE

Library element type codes. This field in not implemented for TIP library access and is ignored. It was used in TIP/30 to allow a library directory read or a library element read. Because a library is implemented as a directory, it is easy to get a directory listing via Unix or the TLIB transaction.

FDES-FCS-PERM
Specified when the element is opened. The default access is read.

If read access desired, set to "R".

If write access desired, set to "W".

FDES-ELEMENT
Element (module) name within library. This field contains the actual element name.

FDES-COMMENTS
Comments (stored in element header record).

FDES-DATE
Date module was created: "YY/MM/DD" format.

FDES-TIME
Time module was created: "HH:MM" format

FCS-CLOSE - Library: Close Element

The FCS-CLOSE function closes the specified library element and removes the entry for the file from the Active File Table (AFT) for the process. If the element was opened for output, it becomes active and replaces any old copies of the element.

Syntax:

```
CALL "TIPFCS" USING     FCS-CLOSE
                        file-pkt
```

Where:

FCS-CLOSE
Function code from the TC-FCS copybook.

file-pkt
Logical file name packet.

Error Conditions:

PIB-STATUS	Meaning
PIB-FUNCTION	File is not assigned to the program.

FCS-GET - Library: Read Next Line

The FCS-GET function reads the next line of an input library element.

Syntax:

```
CALL "TIPFCS" USING     FCS-GET
                        file-pkt
                        record
```

Where:

FCS-GET
Function code from the TC-FCS copybook.

file-pkt
Logical file name packet.

record
Record area where line data is placed.

The record area is a fixed size of 128 bytes. Records are padded at the end with sufficient spaces to fill 128 characters (the carriage-return and linefeed characters are removed when the data is placed in the program's record area).

Error Conditions:

PIB-STATUS	Meaning
PIB-EOF	The end of the element has been reached.
PIB-FUNCTION	The file is not assigned to the program.
PIB-IO-ERROR	An I/O error occurred on the disk.
PIB-WRONG-MODE	The library file was not opened for input processing.

FCS-NOUP - Library: Close Element (No update)

The FCS-NOUP function is similar to a FCS-CLOSE function. If the element is currently open with "Write" access, the element will *not* be written - the previous version of the element, if any, remains the current element.

Syntax:

```
CALL "TIPFCS" USING     FCS-NOUP
                        file-pkt
```

Where:

FCS-NOUP
Function code from the TC-FCS copybook.

file-pkt
Logical file name packet.

Error Conditions:

PIB-STATUS	Meaning
PIB-FUNCTION	File is not assigned to the program.

FCS-OPEN - Library: Open Element

The FCS-OPEN function assigns the library to the issuing process and makes an appropriate entry for the logical file in the Active File Table (AFT).

Syntax:

```
CALL "TIPFCS" USING     FCS-OPEN
                        file-pkt
                        file-desc
```

Where:

FCS-OPEN
Function code from the TC-FCS copybook.

file-pkt
Logical file name packet.

file-desc
File descriptor packet (see earlier description of the TC-FDES library descriptor packet).

Example:

```
... in the program's work area ...
    02  LIB-FDES.           COPY TC-FDES.
    02  LIB-LFN             PICTURE X(9).
    02  LIB-REC             PICTURE X(128).
    02  LIB-HEADER          PICTURE X(256).
     ... in the PROCEDURE DIVISION ...
    MOVE SPACES              TO LIB-FDES
    MOVE "INFILE"            TO LIB-LFN
    MOVE "TIP"               TO FDES-FILE-NAME
    MOVE "R"                 TO FDES-FCS-PERM
    MOVE "S"                 TO FDES-FCS-TYPE
    MOVE "TC-FDES"           TO FDES-ELEMENT
    CALL "TIPFCS" USING      FCS-OPEN
                             LIB-LFN
                             LIB-FDES
```

The above example opens the element "TC-FDES" in the library named "TIP" for read operations (FDES-FCS-PERM).

Note: The name in the LFN file packet ("INFILE") can be any name the programmer chooses — the TIP file system uses the name to determine which file the program is referring to during subsequent CALLs to TIPFCS.

Error Conditions:

PIB-STATUS	Meaning
PIB-IO-ERROR	An I/O error occurred while opening the file.
PIB-DUP-AFT-NAME	A file with the name given in FILE-PKT is already assigned to the terminal.
PIB-DUP-KEY	An element of that name already exists in the library. This warning is given when an existing element is opened with "Write" access. The program may choose to ignore this error - and thereby update an existing element when the FCS-CLOSE is issued later.
PIB-FUNCTION	An attempt was made to open a file that is not a library.
PIB-LOCKED	"Write" access is being requested and the element has already been opened for output.
PIB-NOT-FOUND	The requested library has not been defined in TIP. (Use smfile to define it.)

FCS-PUT - Library: Write Line

The FCS-PUT function will output the next line (sequential fashion) to the library element. Trailing spaces are removed from the data supplied by the program (128 bytes!) and a carriage return and linefeed character are appended to the line in the file.

Syntax:

```
CALL "TIPFCS" USING     FCS-PUT
                        file-pkt
                        record
```

Where:

FCS-PUT
Function code from the TC-FCS copybook.

file-pkt
Logical file name packet.

record
128 character record area.

Error Conditions:

PIB-STATUS	Meaning
PIB-FUNCTION	The File is not assigned to the program.
PIB-IO-ERROR	An I/O error occurred on the file.
PIB-WRONG-MODE	The library file was not opened for output processing.

Transaction Suspend (TIPSUSPEND)

This call has the ability to suspend a running transaction and then later rejoin that transaction and complete it. You can only suspend a transaction if you have actually started one by locking some data record. And you can only rejoin a previously suspended transaction if your program has done no locking or updates.

The important data fields are as follows:

```
WORKING-STORAGE or LINKAGE SECTION.

05  TRAN-TIME-OUT    PIC 9(8) BINARY.
05  TRANS-ID         PIC X(24).
```

To suspend the current transaction:

```
CALL "TIPSUSPEND" USING FCS-PUT, TRANS-ID, TRAN-TIME-OUT
```

This will put the current transaction into a suspended state for up the number of seconds in TRAN-TIME-OUT. If the transaction is not resumed within that time frame it will be assumed to be aborted and the data will be rolled back. The 'transacciton Id' is returned in TRANS-ID as a text string and this valued must be used later to resume the transaction.

To rejoin a suspended transaction:

```
CALL "TIPSUSPEND" USING FCS-GET, TRANS-ID
```

Possible status codes.

PIB-NOT-HELD	no transaction active so nothing to suspend
PIB-NOT-FOUND	could not find a suspended transaction with the given identifier
PIB-FULL	Can not join a suspended transaction because the program has already started a new transaction.

The tipix.conf parameter TRANSUSPEND can be used to specify the default maximum time to suspend a transaction. Without that the default is 15 seconds.

TIP Print Facility (TIPPRINT)

TIP native mode programs may call the subroutine **TIPPRINT** to perform printing functions. TIPPRINT directs print lines to any of the following "destinations":

- an auxiliary (attached) printer
- a terminal (in full screen display mode)
- a logical printer (printers are defined to TIP using the smprint utility program; the "printer" may, in fact, be redirected to other programs or devices in the Unix system.

The user interface with TIPPRINT is similar to the interface used in standard TIPFCS CALLs. The first three parameters — *function-code*, *filename*, and *record* — are common to both interfaces - the fourth parameter of TIPPRINT, however, supplies the name of a user supplied work area that TIPPRINT uses as a buffer and/or work area. TIPPRINT uses the filename (2nd parameter) to determine the destination of the print file (see list above).

A TIP native mode program issues CALLs to the TIPPRINT subroutine to perform the following functions:

Function	Description
OPEN	Initialize the interface to TIPPRINT.
PUT	Pass a single print line image to TIPPRINT.
FLUSH	Force TIPPRINT to empty its internal buffer.
CLOSE	Terminate the interface with TIPPRINT

The program provides the print lines and TIPPRINT ensures their delivery to the specified printer.

The following sections describe the various CALLs to TIPPRINT. The CALLs are described in the sequence that they are normally encountered in a program; namely, OPEN, PUT, FLUSH, and CLOSE.

TIPPRINT Print Destinations

Printers are defined in the TIP system by using the **smprint** utility program. Printers are assigned logical names and these logical names are defined in terms of the actual final routing of the print lines. For example, a printer PRNTR must be defined for use by the system; however, the definition of "PRNTR" may be defined to route the print lines (as standard input or via a pipe) to a specific Unix system command (for example: **lp** or some other spooling system).

The TIPPRINT subroutine can direct print lines to a number of potential destinations. Programs that call TIPPRINT provide printer destination information when opening the interface to specify where the print lines are to be sent. This section describes the various supported destinations and also discusses special information that may interest the programmer.

The second parameter on all CALLs to TIPPRINT is a standard nine-byte filename packet. This filename packet is the primary place where the program can indicate the desired destination of the output. In some cases, the program may choose to supply additional destination information in the "INFO-PKT" that is the third parameter passed on a FCS-OPEN function call.

ROLL - Line by Line Terminal Output

Specifying the character string "ROLL" as a destination tells TIPPRINT to "roll" (scroll) the generated lines of output to the terminal that is calling TIPPRINT. In this case, TIPPRINT passes the data portion of the generated print lines to the standard TIP output routine "ROLL".

ROLL will move the contents of the terminal up one line (the top line disappears off the screen) and then outputs the current data to the last line of the terminal. This print destination is often used to test or debug print programs when a printer is not available. The ROLL subroutine is described in the MCS section of this documentation. ROLL handles only 80 characters of data; using TIPPRINT with a destination of "ROLL" results in print lines being truncated at 80 characters.

AUX0 - Full Screen Output

Specifying a destination of "AUX0" tells TIPPRINT to output the print data one screen full at a time. TIPPRINT accumulates print lines until there are N-1 lines (N is the number of rows on the terminal).

TIPPRINT then outputs the N-1 lines of data (truncated to the width of the screen if necessary) on lines two through N of the terminal where the program is executing and automatically displays a continuation prompt on the first line of the terminal:

```
Continue? ▶Yes ▶No
```

Reply:

- Yes or F2 if you wish to see the next screen full of information,
- No or MSG WAIT to return PIB-BREAK status to the calling program and thus (presumably) halt printing as soon as possible.

AUX0 is the default print destination for many of the TIP utility programs that display information on the terminal.

AUX1 - Auxiliary Device

You may direct TIPPRINT output to a terminal auxiliary printer by using the device name AUX1.

TIPPRINT examines the TIPPRINTAUX environment variable to determine the local terminal printer type:

Type	Description
HP	for a Hewlett-Packard LaserJet printer
EPSON	for an EPSON dot-matrix printer
D630	for a Diablo 630 printer
LP	for a Unix spooler lp If an application uses TIPPRINT to output to a logical printer named "AUX1", and TIPPRINTAUX=LP is specified, then TIPPRINT uses the TIP printer definition for "PRNTR" (as defined with smprint).
PS[=filename]	for a Postscript printer. You may also specify a filename to send ahead of the print data by specifying PS=filename. If no postscript filename is supplied, the default postscript file - named "land" - supplied with TIP is used.

Example: TIPPRINTAUX=PS=arcland

You can also specify the following options. (Separate options with a colon.)

Option	Meaning
TOF	Top of Form. TIPPRINT will supply an initial form feed if the application doesn't.
BOF	Bottom of Form. TIPPRINT will supply a final form feed if the application doesn't.
LPP=n	Lines Per Page. Specify the number of lines on a logical page.

Example:

```
TIPPRINTAUX=PS=arcland:TOF:BOF:LPP=43
```

FCS-CLOSE - Close TIPPRINT Interface

When the program has finished generating print lines it must close the interface to TIPPRINT. The CLOSE function automatically performs the FLUSH function (see previous section).

Syntax:

```
CALL "TIPPRINT" USING   FCS-CLOSE
                        file-pkt
                        dummy
                        buffer
```

Where:

FCS-CLOSE
This function code (normally defined via the supplied copybook TC-FCS) indicates that the desired function is to CLOSE the TIPPRINT interface.

file-pkt
Standard (9-byte) filename packet that specifies the printer that TIPPRINT is to use. This is the same packet as described in the previous section (TIPPRINT: Open).

dummy
The third parameter is a dummy parameter - you could use the usual line packet. It is present only to preserve symmetry with the other calls to TIPPRINT.

You cannot supply a line of print data on a CLOSE call as TIPPRINT ignores the parameter.

buffer
The buffer that is assigned for TIPPRINT's use as described in the previous section (see FCS-OPEN Open TIPPRINT Interface for more information).

Additional Considerations:

- The CLOSE operation delivers any buffered print data that is in the TIPPRINT buffer. You do not need to explicitly FLUSH the buffer before calling the CLOSE function. Once the interface to TIPPRINT is closed you may reopen it with a different printer specification.

Error Conditions:

PIB-STATUS	Meaning
PIB-BREAK	There may be a problem with the printer (see previous section TIPPRINT: Put).

FCS-FLUSH - Flush TIPPRINT Buffer

Since TIPPRINT may be buffering the print lines that are being passed by the user program, your program may need to force a FLUSH of the TIPPRINT buffer.

An example of this situation occurs during the printing of cheques. The program may "print" several lines (via TIPPRINT) and proceed to update a master file to indicate that a cheque was printed for the customer. If it could not be verified that the cheque was printed and a system crash occurred before the cheque was actually printed, the master file would not reflect the real world situation. In this situation, the program can issue a FLUSH request to TIPPRINT after every complete cheque is printed and in effect "wait" to be certain that printing was complete.

The FLUSH request should be issued after each complete cheque and *not* after every line of the cheque!

TIPPRINT automatically performs a FLUSH operation whenever the TIPPRINT interface is closed by the program. It is not necessary for your program to issue a FLUSH before issuing a CLOSE to TIPPRINT.

Syntax:

```
CALL "TIPPRINT" USING   FCS-FLUSH
                        file-pkt
                        dummy
                        buffer
```

Where:

FCS-FLUSH
This function code (normally defined via the supplied copybook TC-FCS) indicates that you wish to FLUSH the TIPPRINT buffer.

file-pkt
Standard (9-byte) filename packet that is used to specify the printer that TIPPRINT is to use. This is the same packet as described in the previous section (see FCS-OPEN Open TIPPRINT Interface for more information.)

dummy
The third parameter is a dummy parameter (the usual line packet could be used) that is present only to preserve symmetry with the other calls to TIPPRINT.

You cannot supply a line of print data on a FLUSH call - TIPPRINT ignores the parameter.

buffer
The buffer that is assigned for TIPPRINT's use as described in the previous section (see FCS-OPEN Open TIPPRINT Interface for more information.)

Additional Considerations:

- The FLUSH operation delivers any buffered print data that is in the TIPPRINT buffer (this normally occurs when the buffer is full).
- There is no need to FLUSH the buffer unless your program must be certain that the print lines that have been passed across the TIPPRINT interface have in fact been printed.

Error Conditions:

PIB-STATUS	Meaning
PIB-BREAK	There may be a problem with the printer (see section TIPPRINT: Put for more details).

FCS-OPEN - Open TIPPRINT Interface

The program must first establish the interface to the TIPPRINT subroutine by issuing a call to TIPPRINT with a function code of "FCS-OPEN". This call:

- Initializes the TIPPRINT interface
- Establishes the desired print destination and
- Specifies printing options that you require.

Syntax:

```
CALL "TIPPRINT" USING   FCS-OPEN
                        file-pkt
                        info-pkt
                        buffer
```

Where:

FCS-OPEN

This function code (normally defined via the supplied copybook TC-FCS) indicates that the desired function is to OPEN the interface.

file-pkt

Standard (8+1 byte) filename packet that is used to specify the output device that TIPPRINT is to use.

See also description of TIPPRINT Print Destinations on page 242 for further details.

The filename may be the name of a batch print file (for example, PRNTR) or may be the name of an auxiliary print device.

info-pkt

Information packet required ONLY on the call to TIPPRINT with the FCS-OPEN function.

The supplied copybook TC-PRINT defines the format of the information packet:

buffer

The fourth parameter on the call to TIPPRINT with a FCS-OPEN function code identifies the buffer that the user program must provide for TIPPRINT to use.

This buffer must be a minimum of 1,280 bytes and may be a maximum of 3,584 bytes (any additional space is wasted!).

Note: You must fullword align this buffer.

The program need not initialize this buffer - TIPPRINT manages this area directly.

The program must not modify any field in this buffer from the time it passes an OPEN function to TIPPRINT to the time it passes a CLOSE function to TIPPRINT.

Warning:

Programs should not make any assumptions about any observed contents of this buffer!

The supplied copybook TC-PBUFR defines the format of the buffer.

TC-PRINT copybook

```
05  INFO-PKT                COPY TC-PRINT.
*
* COPY ELEMENT FOR TIPPRINT INFORMATION PACKET
   10  PRINT-BUF-LEN        PICTURE 9(4) BINARY SYNC.
   10  PRINT-PAG-LEN        PICTURE 9(4) BINARY SYNC.
   10  PRINT-ERR-TERM       PICTURE X(4).
   10  PRINT-TOP-OF-FORM    PICTURE X.
   10  PRINT-LINE-FEED      PICTURE X.
   10  PRINT-NOW-PRINTING   PICTURE X.
   10  PRINT-UPPER-CASE     PICTURE X.
   10  PRINT-RESERVED       PICTURE X.
   10  PRINT-VFB-INFO       PICTURE X.
   10  PRINT-FULL-FILE-INFO PICTURE X.
   10  PRINT-TITLE          PICTURE X.
   10  PRINT-SUBJECT        PICTURE X(20).
   10  PRINT-FULL-FILE-NAME PICTURE X(20).
   10  PRINT-MS-DOS-FILE-NAME
           REDEFINES PRINT-FULL-FILE-NAME.
       15 PRINT-MS-DOS-DRIVE       PICTURE X.
       15 PRINT-MS-DOS-FILE        PICTURE X(16).
       15 PRINT-MS-DOS-EXTENSION   PICTURE X(3).
   10 PRINT-VFB-CHANNEL    PICTURE 9(4) BINARY SYNC
```

```
                                        OCCURS 15 TIMES.
```

The following is a description of the fields that make up the TC-PRINT copybook:

PRINT-BUF-LEN

Used to specify the length of the buffer that the user program is providing for TIPPRINT to use to buffer print lines (the fourth parameter).

The program must move the length of the buffer that has been reserved for TIPPRINT's use to this field before issuing the FCS-OPEN function.

The minimum buffer size is 1,024 bytes; the maximum (usable) buffer size is 3584.

PRINT-PAG-LEN

The desired number of lines per page.

TIPPRINT will return the status PIB-OVERFLOW whenever the TIPPRINT interface has printed this many lines.

Your program may ignore this overflow status OR may use it as a signal to output headings.

PRINT-ERR-TERM

Specifies the name of the terminal that is to receive an error message if an error condition occurs.

Default: terminal that is invoking TIPPRINT.

The value specified may be:

term	The name of a valid terminal in the network
*CON	To indicate the UNIX operator console or,
*RET	To indicate that no error message is to be sent. TIPPRINT will not output any error message and will simply return to the calling program with PIB-BREAK status in the PIB.

PRINT-TOP-OF-FORM

You can now specify the following options:

Y	TIPPRINT ensures that there is a skip to top of form both before printing output and after completion of printing. If there are no skips to top of form present at the beginning and ending of output, TIPPRINT will insert them. If there are skips to top of form present at the beginning and ending of output, nothing is inserted.

N TIPPRINT ensures that, if the first output is a skip to top of form, it is removed. If the last output is a skip to top of form, TIPPRINT removes it.

T TIPPRINT ensures that the first output is a skip to top of form and that there is no skip to top of form at the ending of output.

B TIPPRINT ensures that there is no skip to top of form at the beginning of output and that there is a skip to top of form at the ending of output.

space The value specified in the "Form Feed Requirements" of the terminal definition will be used as the PRINT-TOP-OF-FORM value if a terminal definition (see smterm) exists for the current terminal and if the value of Form Feed Requirements is non-blank. Otherwise, the value of the system parameter PRINT-TOP-OF-FORM from the tipix.conf file will be used.

PRINT-LINE-FEED

Specify as either "Y" or "N" or space. Some printers automatically provide a "free" line feed character whenever a full screen of data is transferred from the terminal to the printer.

Set this field to "Y" or "N" to indicate whether TIPPRINT is to insert a line feed character at the end of every data transfer to the communications printer.

PRINT-NOW-PRINTING

Specify as either "N" or "Y" (default).

If this field is not an "N", TIPPRINT displays the "NOW PRINTING" message on the terminal when the call to OPEN TIPPRINT is issued. The message will be erased when the TIPPRINT interface is CLOSEd.

If you set this field to "N", the "NOW PRINTING" message will not be displayed on the terminal.

The NOW PRINTING message is more than a convenience message. If the NOW PRINTING message is suppressed there is no way you can interrupt the printing (i.e. by pressing MSG WAIT).

The NOW PRINTING message is often suppressed because the program is using a screen format. In this case, the program normally issues its own version of this message (for example by using a call to TIPMSGE).

PRINT-UPPER-CASE

Specify as either "Y" or "N" or space.

Y Indicates that TIPPRINT is to translate alphabetic characters to uppercase.

N Indicates that no translation is to occur

space Indicates that the system default is to be taken (no translation).

PRINT-RESERVED

This field is reserved for future use and is currently not examined by TIPPRINT.

PRINT-FULL-FILE-INFO

When printing to a MS-DOS file:

If a six-character filename is enough, put the disk drive letter, a colon, and the 6-character filename in the FILE-PKT field. For example "C:FI.EXT"If you don't specify an extension, it is assumed to be ".PRN". For example, "C:FILNAM" implies "C:FILNAM.PRN".

If a six-character name is not enough, move spaces to FILE-PKT, and code a "D" in PRINT-FULL-FILE-INFO.

D Print to a MS-DOS file using a long filename in the PRINT-FULL-FILE- NAME field.

In either case, TIP/fe must be running in smart mode.

PRINT-TITLE

If you set this field to a "Y", TIPPRINT expects to find up to 20 characters of program supplied data in the field PRINT-SUBJECT.

TIPPRINT will generate a title page (similar to a WRTSML that includes the subject and user id, etc.)

If you set this field to an "S" (indicating that data is in the PRINT-SUBJECT field), TIPPRINT will suppress the title page for non-batch destinations and will generate a title page for batch printer destinations.

If you use any other value in this field the contents of PRINT-SUBJECT will be ignored and no title page will be produced.

PRINT-SUBJECT

See prior description of PRINT-TITLE.

PRINT-FULL-FILE-NAME

This field is used when printing to an MS-DOS file with a long filename. It contains smaller fields containing:

1. the disk drive letter (1 byte)
2. path and filename (16 bytes)
3. extension (3 bytes).

This field is only used if PRINT-FULL-FILE-INFO contains a "D" and FILE-PKT does not contain a ":".

TC-PBUFR copybook

You may use the supplied copybook TC-PBUFR in your program's work area to define the buffer:

```
COPY TC-PBUFR.
* TIPPRINT BUFFER PACKET
* --- USER PROGRAM SHOULD NOT MODIFY THESE FIELDS ---
05  TIPPRINT-BUF.
    10  FILLER              PICTURE 9(8) BINARY SYNC.
    10  FILLER              PICTURE X(2556).
05  TIPPRINT-BUFFER REDEFINES TIPPRINT-BUF.
    10  BU-PAGE-LENGTH      PICTURE 9(4) BINARY SYNC.
    10  BU-ICAM-STATUS      PICTURE X.
    10  FILLER              PICTURE X(2557).
```

The following is a description of the fields that make up the TC-PBUFR copybook:

BU-PAGE-LENGTH

While the TIPPRINT interface is open, this field contains the number of lines per page that TIPPRINT has determined.

This field is intended for informational purposes only; do not modify it in your program.

BU-ICAM-STATUS

When PIB-BREAK status is set, this field may contain a detailed status code that indicates the reason for delivery notification failure:

0	Device Down.
1	Read/Write Error.
2	Out of Forms.
3	End of Tape.
4	Device Off line.
B	Line Down.
C	Terminal Down.
D	Invalid Destination.
E	No Network Buffers.
F	Disk Error.
G	Wrong Buffer Length.

? Unknown Status.

Error Conditions:

PIB-STATUS	Meaning
PIB-FUNCTION	Invalid parameter. Function code is not FCS-OPEN, FCS-PUT, FCS-FLUSH or FCS-CLOSE.
PIB-IO-ERROR	Invalid parameter list.
PIB-LOCKED	TIP may return this status if your program specified "*RET" in the PRINT-ERR-TERM field in the info packet and the destination printer is locked when the FCS-OPEN is issued. If you do not set PRINT-ERR-TERM to "*RET", TIPPRINT will display the message "Waiting for printer" on the terminal and will wait for the printer to be available before returning to the program.
PIB-NO-MEM	Buffer length too small (less than 1024 bytes).
PIB-NOT-FOUND	Destination or error terminal not found.

FCS-PUT - Output Print Line

User programs call the TIPPRINT subroutine with a function of FCS-PUT to output each print line. A description appears below of the format of the print line. The program formats the print line with whatever text is desired, inserts an appropriate device independent carriage control code and passes the print line to TIPPRINT for delivery.

You must be aware that TIPPRINT *may be* accumulating print lines in the buffer that is provided as a TIPPRINT work area. This means that the line that is passed on a PUT call to TIPPRINT *may not be physically printed* at the time the call is issued (see also the description of the FCS-FLUSH function call to TIPPRINT).

Syntax:

```
CALL "TIPPRINT" USING   FCS-PUT
                        file-pkt
                        print-line
                        buffer.
```

Where:

FCS-PUT
This function code (normally defined via the supplied copybook TC-FCS) indicates that the desired function is to output a print line to the interface.

file-pkt
Use this standard (nine-byte) filename packet to specify the printer that TIPPRINT is to use. This is the same packet as described in the previous section (TIPPRINT: open).

print-line
This is the print line (packet) that contains the data to be printed (and the carriage control to use).

You may use the supplied TC-PLINE copybook to define this area:

buffer
This is the buffer that your program provides for TIPPRINT.

TC-PLINE copybook

```
COPY TC-PLINE.
*
* COPY ELEMENT FOR TIPPRINT LINE PACKET
05  PRINT-LINE.
    10  LI-LENGTH        PICTURE 9(4)
                          BINARY.
    10  FILLER           PICTURE XX.
    10  LI-DI-CONTROL    PICTURE X.
        88  LI-DI-HOME       VALUE X'27'.
        88  LI-DI-SPACE1     VALUE X'01'.
        88  LI-DI-SPACE2     VALUE X'02'.
        88  LI-DI-SPACE3     VALUE X'03'.
    10 LI-DATA PICTURE X(250).
```

The following is a description of the fields that make up the TC-PLINE copybook:

PRINT-LINE
A variable length record containing a length field, a DI code (for carriage control), and the data to be printed.

The above copybook defines the print line as a fixed length area for convenience only.

Your program may establish several print lines of varying length for specific purposes (for example, headings).

LI-LENGTH

The length of the entire print line packet.

The length specified must include the five bytes preceding the actual data. In the copybook for example, you would move 137 to LI-LENGTH.

TIPPRINT supports a maximum length of 250 bytes for data to be printed.

If the field LI-LENGTH contains a value greater than 255 (250+5), TIPPRINT truncates the print line to 250 characters.

The minimum specification for this field is a value of five bytes. Some carriage control codes cannot specify data at the same time; that is, skip to top of page: X'27'.

LI-DI-CONTROL

This field contains the Device-Independent Control character that indicates the desired type of printer spacing used when printing this print line.

Standard FORTRAN skip codes are:

space	Single space
0	Double space
-	Triple space
I	Skip to the top of a new page

Additional special codes are:

B Not implemented in TIP. Output the data portion of the print line using BLOCK characters (to create title or separator pages).

Within your line of data, a carriage return (X'0D'), means begin a new line of block characters.

V Output the contents of the print line to the device without any translation or other interpretation.

This allows you to send arbitrary character codes to a printer - some printers' react to character sequences to perform advanced functions.

If the print destination is AUX0 (the screen) or ROLL (the screen with scrolling), the print line is discarded and PIB-GOOD is returned.

LI-DATA

This field contains the text of the print line.

TC-DI copybook:

The supplied copybook TC-DI defines some commonly used carriage control codes. Since this copybook contains VALUE clauses you must place it in the WORKING-STORAGE SECTION of your program.

```
COPY TC-DI.
*
* DEFINE CODES USED FOR PRINTER CARRIAGE CONTROL
05 TC-DI-CODES.
    10  TC-DI-HOME              PICTURE X VALUE X"27".
    10  TC-DI-PRINT-SPACE1      PICTURE X VALUE X"01".
    10  TC-DI-PRINT-SPACE2      PICTURE X VALUE X"02".
    10  TC-DI-PRINT-SPACE3      PICTURE X VALUE X"03".
    10  TC-DI-PRINT-SPACE4      PICTURE X VALUE X"04".
    10  TC-DI-PRINT-SPACE5      PICTURE X VALUE X"05".
    10  TC-DI-PRINT-SPACE6      PICTURE X VALUE X"05".
    10  TC-DI-PRINT-SPACE7      PICTURE X VALUE X"07".
    10  TC-DI-PRINT-SPACE8      PICTURE X VALUE X"08".
    10  TC-DI-PRINT-SPACE9      PICTURE X VALUE X"09".
    10  TC-DI-PRINT-SPACE10     PICTURE X VALUE X"0A".
    10  TC-DI-PRINT-NO-SPACE    PICTURE X VALUE X"10".
    10  TC-DI-SPACE1            PICTURE X VALUE X"51".
    10  TC-DI-SPACE2            PICTURE X VALUE X"52".
    10  TC-DI-SPACE3            PICTURE X VALUE X"53".
    10  TC-DI-SPACE4            PICTURE X VALUE X"54".
    10  TC-DI-SPACE5            PICTURE X VALUE X"55".
    10  TC-DI-SPACE6            PICTURE X VALUE X"55".
    10  TC-DI-SPACE7            PICTURE X VALUE X"57".
    10  TC-DI-SPACE8            PICTURE X VALUE X"58".
```

Error Conditions:

PIB-STATUS	Meaning
PIB-FULL	TIP returns this status if your program has a serial resource locked and this FCS-PUT caused the TIPPRINT buffer to be full. Even though this status was returned, TIPPRINT accepted the print line and placed it in the buffer. Normally TIPPRINT would flush the buffer to the device at this point, but since there is a serial resource locked, TIPPRINT warns the program that it cannot flush the buffer right now. If the program issues another FCS-PUT with serial resources still locked, that FCS-PUT will be rejected with PIB-LOCKED status. If an FCS-FLUSH or FCS-CLOSE is issued with serial resources locked, TIPPRINT will go ahead and flush the buffer and TIP will probably abort the program and issue a

PIB-STATUS	Meaning
	resources locked, waiting message. This PIB-STATUS is applicable only to non-batch print destinations.
PIB-LOCKED	TIP returns this status if the program issues further FCS-PUT functions when the TIPPRINT buffer is already full and serial resources are locked. TIPPRINT ignores the print line. This PIB-STATUS is applicable only to non-batch print destinations.
PIB-NOT-FOUND	TIP returns this status if the device-independent carriage control character in the print-line is an unrecognized value. TIPPRINT ignores the print line.
PIB-OVERFLOW	TIP returns this status if TIPPRINT determines that the number of lines per page (as declared in the INFO packet on the preceding OPEN) has been exceeded. PIB-OVERFLOW status will continue to be returned on subsequent FCS-PUT requests until such time as TIPPRINT receives a command to home the paper. This is not an error condition - merely overflow notification. Your program may choose to ignore this event as it may be counting its own lines or may use this as a signal to output page headings.
PIB-BREAK	TIP returns this status when the printer is no longer available due to a delivery notification error or because you have interrupted the TIPPRINT process and have indicated that you do not want it to continue. You may interrupt TIPPRINT processing (presuming that a "NOW PRINTING" message has been displayed!) by pressing MSG-WAIT. TIPPRINT will interrupt (break) before the next data transfer to the print device.

TIPPRINT displays the BREAK prompt as follows:

```
Break - Continue? ▶YES ▶NO
```

If you enter "NO" to this break message, TIP returns a PIB-BREAK status to the program.

Note: It is imperative that the program check for PIB-BREAK status after every FCS-PUT.

TIP also returns PIB-BREAK status if, for example, the printer is out of paper. The program should take appropriate action; it should stop printing in any case.

If an I/O error occurs on an auxiliary device, TIP sends a message to the error reporting terminal (as specified in the information packet supplied at the time TIPPRINT was OPENed). The message identifies the error and the name of the terminal associated with the error.

The text of the message is as follows:

```
PRINT ERROR AT ____, ERROR = '_______'
```

If this condition occurs, PIB-BREAK status is returned to the program to indicate that the printed output has been broken.

Accessing TIP Journal Files

The TIP File Control System (FCS) automatically writes BEFORE and/or AFTER images of updated records to the TIP Journal file. Parameters specified for each file with TIP Enterprise Manager (See the Defining Files section of the TIP/em documentation) control the writing of before and after images.

To *write user* records to the journal file from an *online transaction*, use the TIPFCS FCS-JOURNAL function. Such user records can be written to the journal file, for example, to mark certain exceptional events or to be able to monitor transaction usage.

To *read* all journal or QBL file records from a *batch program*, see TIPJRNOP, TIPJRNCL, and TIPJRNGT.

The format of a *user* record in the journal file is entirely at the discretion of the program writing the record. The only restriction is that the record must contain a proper record prefix (described in the following section).

Journal and QBL File Record Format

The TIP Journal file contains variable length records. Each record has a journal prefix that contains a record length field, record type field and assorted information about the data that may or may not follow the prefix. Some journal records contain NO data - they are simply a prefix

The copybook TC-JRNC is provided to supply key constant values for the journal record layout. Namely the journal record prefix length, the maximum journal record data length and the maximum journal record length.

The Quick Before Look (QBL) files have the same format as the journal files.

TC-JRNC copybook:

```
* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
*                                                               *
* TIP JOURNAL FILE RECORD CONSTANTS                             *
* Based on copybook TC-JRN                                      *
*                                                               *
* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *

* JRN-PREFIX-LENGTH LENGTH OF JOURNAL/QBL RECORD HEADERS
* JRN-MAX_DATA-LENGTH LENGTH OF MAXIMUM RECORD DATA STORAGE
* JRN-MAX_REC-LENGTH LENGTH OF MAXIMUM RECORD STORAGE

05  JRN-PREFIX-LENGTH       PICTURE 9(8) COMP-4 VALUE 74.
05  JRN-MAX-DATA-LENGTH     PICTURE 9(8) COMP-4 VALUE 8184.
05  JRN-MAX-REC-LENGTH      PICTURE 9(8) COMP-4 VALUE 8250.
```

TC-JRN copybook:

The supplied copybook TC-JRN describes the layout of the various records that may appear in the journal file.

```
* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
*                                                               *
* TIP JOURNAL FILE RECORD DEFINITION                            *
*                                                               *
* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *

05  JRN-RECORD.
    10  JRN-PREFIX.
*
* JRN-REC-LEN LENGTH OF RECORD (INCL PREFIX)
* ZERO LENGTH IMPLIES END-OF-FILE
*
* JRN-REC-TYPE TYPE OF JOURNAL RECORD -
*
* AFTR - IMAGE OF A DATA RECORD AFTER UPDATE
*        (INCLUDING LOGICAL DELETE)
* BEFR - IMAGE OF A DATA RECORD BEFORE UPDATE
* CKPT - NOTIFICATION A DATA FILE WAS CLOSED
* - NOTIFICATION A LIBRARY ELEMENT WAS
* READ -OR- WRITTEN
* DELT - IMAGE OF A MIRAM RECORD DELETED VIA RCB
* LGOF - TIP USER LOGOFF
* LGON - TIP USER LOGON
* NEW  - IMAGE OF A NEW DATA RECORD THAT WAS ADDED
* PREN - NOTIFICATION OF END OF TRANSACTION PROG
* PRST - NOTIFICATION OF START OF TRANSACTION PROG
* STAT - TIP STATISTICS RECORD
```

```
* TREN - END OF TRANSACTION MARKER
* USER - USER WRITTEN JOURNAL RECORD
* - FORMAT DEFINED BY PROGRAM WHICH WRITES
* THE RECORD TO THE JOURNAL FILE
* PREP - NEW FOR 2 PHASE COMMIT
* RDYC - NEW FOR 2 PHASE COMMIT
* RDYA - NEW FOR 2 PHASE COMMIT
* COMM - NEW FOR 2 PHASE COMMIT
* ABRT - NEW FOR 2 PHASE COMMIT
* HDR - NEW EXPLICIT NAME FOR FILE HEADER/CONTROL RECORD
*
* JRN-UID - RECORD WRITTEN FOR THIS user id

* JRN-TRID - EXECUTING THIS TRANSACTION PROGRAM
* JRN-LFD - FILE LFD NAME
* JRN-FULL-DATE - DATE STAMP (YYYYMMDD) NEW 4 DIGIT YEAR
* JRN-TIME - TIME STAMP (HHMMSS)
* JRN-TID - RECORD WRITTEN FOR THIS TERMINAL
* JRN-DIRECT-BLK-NO - BLOCK NUMBER ( RELATIVE FILES ONLY)
* JRN-ACCT - USER'S LOGON ACCOUNT NUMBER
* JRN-STATE
* RLBK - SET TO "R" IF THIS JOURNAL
* - RECORD WAS WRITTEN AS A RESULT OF
* - ONLINE ROLLBACK (TRANSACTION
* - ABORTED OR REQUESTED
* - ROLLBACK VIA PIB-LOCK-INDICATOR)
* NORM - SET FOR RECORD WRITTEN FOR NORMAL PROCESSING
/
        15  JRN-REC-LEN             PICTURE 9(8)
                                     COMP-4 SYNC.
        15  JRN-GTRAN-ID.
            20  JRN-GTRAN-LOCAP     PICTURE X(8).
            20  JRN-GTRAN-TRAN-NUM  PICTURE 9(8) COMP-4.
        15  JRN-DIRECT-BLK-NO       PICTURE 9(8) COMP-4.
        15  JRN-REC-TYPE            PICTURE X.
            88  JRN-AFTR                VALUE "A".
            88  JRN-BEFR                VALUE "B".
            88  JRN-CKPT                VALUE "C".
            88  JRN-DELT                VALUE "D".
            88  JRN-LGOF                VALUE "F".
            88  JRN-LGON                VALUE "O".
            88  JRN-NEW                 VALUE "N ".
            88  JRN-PREN                VALUE "E".
            88  JRN-PRST                VALUE "G".
            88  JRN-STAT                VALUE "S".
            88  JRN-TREN                VALUE "T".
            88  JRN-USER                VALUE "U".
            88  JRN-PREP                VALUE "P".
            88  JRN-RDYC                VALUE "R".
            88  JRN-RDYA                VALUE "S".
            88  JRN-COMM                VALUE "X".
            88  JRN-ABRT                VALUE "Y".
            88  JRN-HDR                 VALUE "H".
        15  JRN-UID                 PICTURE X(8).
        15  JRN-TRID                PICTURE X(8).
        15  JRN-LFD
            20  JRN-LFN             PICTURE X(8).
```

```
           15  JRN-FULL-DATE           PICTURE 9(8).
           15  FILLER REDEFINES JRN-FULL-DATE.
               20  JRN-CENTURY         PICTURE 99.
               20  JRN-DATE            PICTURE 9(6).
               20  FILLER REDEFINES JRN-DATE.
                   25  JRN-DATE-YY     PICTURE 99.
                   25  JRN-DATE-MM     PICTURE 99.
                   25  JRN-DATE-DD     PICTURE 99.
           15  JRN-TIME                PICTURE 9(6).
           15  FILLER REDEFINES JRN-TIME.
               20  JRN-TIME-HH         PICTURE 99.
               20  JRN-TIME-MM         PICTURE 99.
               20  JRN-TIME-SS         PICTURE 99.
           15  JRN-TID                 PICTURE X(8).
           15  JRN-ACCT                PICTURE X(4).
           15  JRN-STATE               PICTURE X.
               88  JRN-RLBK                VALUE "R".
               88  JRN-NORM                VALUE " ".
           15  FILLER                  PICTURE X(2).

       10  JRN-DATA.
           15  FILLER                  PICTURE X(4092).
           15  FILLER                  PICTURE X(4092).
      /
      *    10  JRN-STAT-REC REDEFINES JRN-DATA.
      *        15  JRN-STAT-MSG-IN         PICTURE 9(8) COMP-4.
      *        15  JRN-STAT-MSG-OUT        PICTURE 9(8) COMP-4.
      *        15  JRN-STAT-LEN-IN         PICTURE 9(8) COMP-4.
      *        15  JRN-STAT-LEN-OUT        PICTURE 9(8) COMP-4.
      *        15  JRN-STAT-SWAP           PICTURE 9(8) COMP-4.
      *        15  JRN-STAT-CAT-REQ        PICTURE 9(8) COMP-4.
      *        15  JRN-STAT-LOADM-REQ      PICTURE 9(8) COMP-4.
      *        15  JRN-STAT-LOADM-ACT      PICTURE 9(8) COMP-4.
      *        15  JRN-STAT-TOT-RESP       PICTURE 9(8) COMP-4.
      *        15  JRN-STAT-TOT-SCHED      PICTURE 9(8) COMP-4.
      *        15  JRN-STAT-TOT-COMM       PICTURE 9(8) COMP-4.
      *        15  JRN-STAT-FILE-SWAP      PICTURE 9(8) COMP-4.
      *        15  JRN-STAT-CAT-ACT        PICTURE 9(8) COMP-4.
      *        15  JRN-STAT-MCS-ACT        PICTURE 9(8) COMP-4.
      *        15  JRN-STAT-DYN-ACT        PICTURE 9(8) COMP-4.
      *        15  JRN-STAT-ALL-BUSY       PICTURE 9(8) COMP-4.
      *        15  JRN-STAT-MCS-REQ        PICTURE 9(8) COMP-4.
      *        15  JRN-STAT-TOT-IO         PICTURE 9(8) COMP-4.
      *        15  JRN-STAT-TOT-BUSY       PICTURE 9(8) COMP-4.
      *        15  JRN-STAT-BUSY-10        PICTURE 9(8) COMP-4.
      *        15  JRN-STAT-BUSY-15        PICTURE 9(8) COMP-4.
      *        15  JRN-STAT-BUSY-20        PICTURE 9(8) COMP-4.
      *        15  JRN-STAT-DBMS-IO        PICTURE 9(8) COMP-4.
      *        15  JRN-STAT-DBMS-IMP       PICTURE 9(8) COMP-4.
      *        15  JRN-STAT-MAX-B4         PICTURE 9(8) COMP-4.
      *        15  JRN-STAT-PRNTR-IO       PICTURE 9(8) COMP-4.
      *        15  JRN-STAT-BLK-ACT        PICTURE 9(8) COMP-4.
      *        15  JRN-STAT-JRN-ACT        PICTURE 9(8) COMP-4.
      *        15  JRN-STAT-B4-ACT         PICTURE 9(8) COMP-4.
      *        15  JRN-STAT-RESERVED       PICTURE 9(8) COMP-4.
      *        15  FILLER                  PICTURE X(3972).
      *        15  FILLER                  PICTURE X(4092).
```

```
/
*
* LOGOFF RECORD:
*
* JRN-LGOF-LGON-HH HOURS LOGGED ON
* JRN-LGOF-LGON-MM MINUTES LOGGED ON
* JRN-LGOF-LGON-SS SECONDS LOGGED ON
* JRN-LGOF-WALL-MSEC TOTAL TIME LOGGED ON (MILLISEC)
* JRN-LGOF-I-O TOTAL NO I/O'S ISSUED
* JRN-LGOF-MSGIN TOTAL INPUT MESSAGES
* JRN-LGOF-MSGOUT TOTAL OUTPUT MESSAGES
* JRN-LGOF-LGON-FULL-DATE DATE OF LOGON (YYYYMMDD)
* JRN-LGOF-LGON-TIME TIME OF LOGON (HHMMSS)
* JRN-LGOF-FULL-DATE DATE OF LOGOFF (YYYYMMDD)
* JRN-LGOF-TIME TIME OF LOGOFF (HHMMSS)
* JRN-LGOF-AVG-RESP AVERAGE RESPONSE TIME (MILLISEC)
*
*
    10  JRN-LGOF-REC REDEFINES JRN-DATA.
        15  FILLER                  PICTURE X(2).
        15  JRN-LGOF-LGON-HH        PICTURE 9(2).
        15  JRN-LGOF-LGON-MM        PICTURE 9(2).
        15  JRN-LGOF-LGON-SS        PICTURE 9(2).
        15  JRN-LGOF-WALL-MSEC      PICTURE 9(8) COMP-4.
        15  JRN-LGOF-I-O            PICTURE 9(8) COMP-4.
        15  JRN-LGOF-MSGIN          PICTURE 9(8) COMP-4.
        15  JRN-LGOF-MSGOUT         PICTURE 9(8) COMP-4.
        15  JRN-LGOF-LGON-FULL-DATE PICTURE 9(8).
        15  FILLER REDEFINES JRN-LGOF-LGON-FULL-DATE.
            20  JRN-LGOF-LGON-CENTURY       PICTURE 99.
            20  JRN-LGOF-LGON-DATE          PICTURE 9(6).
            20  FILLER REDEFINES JRN-LGOF-LGON-DATE.
                25  JRN-LGOF-LGON-DATE-YY   PICTURE 99.
                25  JRN-LGOF-LGON-DATE-MM   PICTURE 99.
                25  JRN-LGOF-LGON-DATE-DD   PICTURE 99.
        15  JRN-LGOF-LGON-TIME              PICTURE 9(6).
        15  FILLER REDEFINES JRN-LGOF-LGON-TIME.
            20  JRN-LGOF-LGON-TIME-HH       PICTURE 99.
            20  JRN-LGOF-LGON-TIME-MM       PICTURE 99.
            20  JRN-LGOF-LGON-TIME-SS       PICTURE 99.
        15  JRN-LGOF-FULL-DATE              PICTURE 9(8).
        15  FILLER REDEFINES JRN-LGOF-FULL-DATE.
            20  JRN-LGOF-CENTURY            PICTURE 99.
            20  JRN-LGOF-DATE               PICTURE 9(6).
            20  FILLER REDEFINES JRN-LGOF-DATE.
                25  JRN-LGOF-DATE-YY        PICTURE 99.
                25  JRN-LGOF-DATE-MM        PICTURE 99.
                25  JRN-LGOF-DATE-DD        PICTURE 99.
        15  JRN-LGOF-TIME                   PICTURE 9(6).
        15  FILLER REDEFINES JRN-LGOF-TIME.
            20  JRN-LGOF-TIME-HH    PICTURE 99.
            20  JRN-LGOF-TIME-MM    PICTURE 99.
            20  JRN-LGOF-TIME-SS    PICTURE 99.
        15  JRN-LGOF-AVG-RESP       PICTURE 9(8) COMP-4.
        15  FILLER                  PICTURE X(4040).
        15  FILLER                  PICTURE X(4092).
*
```

```
* LOGON RECORD:
*
* JRN-LGON-FULL-DATE DATE OF LOGON (YYYYMMDD)
* JRN-LGON-TIME TIME OF LOGON (HHMMSS)
*
    10  JRN-LGON-REC REDEFINES JRN-DATA.
        15  FILLER                  PICTURE X(1).
        15  JRN-LGON-STATUS         PICTURE X(1).
            88  JRN-LGON-ERROR          VALUE "E".
            88  JRN-LGON-OK             VALUE " ".
        15  FILLER                  PICTURE X(18).
        15  JRN-LGON-FULL-DATE      PICTURE 9(8).
        15  FILLER REDEFINES JRN-LGON-FULL-DATE.
            20  JRN-LGON-CENTURY        PICTURE 99.
            20  JRN-LGON-DATE           PICTURE 9(6).
            20  FILLER REDEFINES JRN-LGON-DATE.
                25  JRN-LGON-DATE-YY    PICTURE 99.
                25  JRN-LGON-DATE-MM    PICTURE 99.
                25  JRN-LGON-DATE-DD    PICTURE 99.
        15  JRN-LGON-TIME           PICTURE 9(6).
        15  FILLER REDEFINES JRN-LGON-TIME.
            20  JRN-LGON-TIME-HH    PICTURE 99.
            20  JRN-LGON-TIME-MM    PICTURE 99.
            20  JRN-LGON-TIME-SS    PICTURE 99.
        15  FILLER                  PICTURE 99.
        15  FILLER                  PICTURE X(4056).
        15  FILLER                  PICTURE X(4092).
*
* CKPT RECORD: (FOR LIBRARY ELEMENT READ OR WRITE)
*
* JRN-CKPT-ELT-NAME ELEMENT NAME
* JRN-CKPT-ELT-TYPE ELEMENT TYPE (S-OURCE M-ACRO ETC)
* JRN-CKPT-ACCESS READ / WRITE ACCESS BY USER
*
    10  JRN-CKPT-REC REDEFINES JRN-DATA.
        15  JRN-CKPT-ELT-NAME       PICTURE X(8).
        15  JRN-CKPT-ELT-TYPE       PICTURE X.
        15  JRN-CKPT-ACCESS         PICTURE X.
            88  JRN-CKPT-READ           VALUE "R".
            88  JRN-CKPT-WRITE          VALUE "W".
        15  FILLER                  PICTURE X(4082).
        15  FILLER                  PICTURE X(4092).
*
* JOURNAL RECORD TYPES: AFTR, BEFR, DELT, NEW, USER
* CONTAIN A VARIABLE AMOUNT OF DATA IN JRN-DATA
* -- DEPENDING ON THE RECORD SIZE OF THE
*    FILE TO WHICH THE IMAGE APPLIES
*
* JOURNAL RECORD TYPES: CKPT (EXCEPTING LIBRARIES), TREN,
*                       PRST, PREN
* CONTAIN ---NO--- DATA OTHER THAN THE PREFIX.
*
* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
```

The following is a description of the fields that make up the TC-JRN copybook:

JRN-PREFIX
A fixed length prefix that appears on the front of ALL records in the journal file.

JRN-REC-LEN
Binary fullword containing the length of the journal file record. This length includes the number of bytes in the record prefix.

JRN-REC-TYPE
The type of journal file record.

JRN-UID
This journal record was written on behalf of this TIP user.

JRN-TRID
The TIP transaction name that was executing when this record was written.

JRN-LFD
The applicable file LFD name (applies to file related journal records).

JRN-FULL-DATE
The date stamp of this record in YYYYMMDD format.

JRN-TIME
The time stamp of this record in HHMMSS format.

JRN-TID
The name of the terminal related to this journal record.

JRN-GTRAN-LOCAP
The name of the LOCAP related to this journal record.

JRN-GTRAN-TRAN-NUM
A transaction number unique for each transaction on a LOCAP.

JRN-DIRECT-BLK-NO
The relative record number if this journal record is a before (BEFR) or after (AFTR) image of a direct (non-indexed) file.

JRN-ACCT
The logon account number of the user to which this journal record pertains. This field contains the account number that was specified when the user logged on TIP.

JRN-STATE
This field contains the character string "R" if this journal record was written by TIP online roll back.

JRN-DATA
A group item indicating the start of the "data" portion of the journal record.

Note: The copybook reserves a great deal of space to accommodate a fairly large record—the record length of the journal record can be large.

Additional considerations:

- The LOGON transaction writes a record to the journal file after four consecutive failed logon attempts (at the LOGON screen.) The logon record in the journal file has a status (JRN-LGON-STATUS) that indicates acceptance or rejection of the logon.

Batch Journal File Access

The TIP library *libtip.a* contains subroutines that provide I/O services to journal files for the batch program.

You may write a batch program to use these supplied subroutines to read the TIP journal file or the QBL file.

TIPJRNOP - Batch Journal File

This subroutine OPENs the input "journal" file. The subroutine OPENs the first file that it finds from the following list:

- File described by TIPJRNIN environment variable
- Journal file
- QBL file

Syntax:

```
CALL "TIPJRNOP"
```

No parameters required.

Additional considerations:

- TIP does not provide an error status.

TIPJRNCL - Batch Journal File Close

This subroutine CLOSEs the input "journal" file. The subroutine CLOSEs whatever file was previously OPENed via a call to TIPJRNOP (see previous description).

Your program should *not* attempt a call to this subroutine *unless* it has completed a prior call to TIPJRNOP.

Syntax:

```
CALL "TIPJRNCL"
```

No parameters required.

TIPJRNGT - Batch Journal File Read

This subroutine READs the next record from the input file and moves it to the area specified as the (only) parameter on the CALL statement. The file header or control record is meant for system use only and will not be returned.

Syntax:

```
CALL "TIPJRNGT" USING   JRN-RECORD
```

Where:

JRN-RECORD

Parameter indicating where the subroutine is to place the next record from the input file.

You should use the previously described copybook (TC-JRN) to define this area.

Additional considerations:

- If the record length (JRN-REC-LEN) is zero after a call to TIPJRNGT, the program must treat this as an end of file indication.
- The batch journal file access routines return a variable length record. In particular, there may be very large records in the input file (for example, BEFR and AFTR images of user data records).
- If your program has no interest in a particular record type, the record can be ignored when it is delivered by the call to the TIPJRNGT subroutine; however, the program must allow sufficient space in the definition of the record area (JRN-RECORD) to house the largest possible journal record!
- This is the reason for the rather generous FILLER items that are defined as part of the group item "JRN-DATA" in the supplied copybook.

FCS Batch Interface

The FCS Batch Interface allows batch programs to access and update files that are managed by TIP.

Batch programs call TIP to Read, Read for Update, and Write records. TIP performs these operations, looks after record locking, saves before-images of records during updates, and journals completed record updates. The preceding actions depend on how the file is defined to TIP.

The TIP FCS Batch Interface consists of:

- tipbatsv — a batch program
- tipbatpi.o — an interface subroutine found in libbat.a.

Each batch application that wants to access TIP must initially CALL BATACTIV to start its own copy of **tipbatsv.** The **tipbatsv** batch interface program runs as a separate process.

Your batch application uses **tipbatpi.o** to pass messages to your **tipbatsv** process that in turn communicates with TIP.

Through this interface, batch programs can access most file types supported by TIP including indexed, direct and sequential.

Prepare to use batch Interface Routine

You must be sure that the correct value for the TIPROOT environment variable has been set and that $TIPROOT/bin is declared in the PATH environment variable.

The interface routine needs to know how many parameters have been passed on a CALL. The method of computing this is different for each COBOL compiler. The genmain utility will create a small library with the correct code for your COBOL compiler.

- If you are using MBP COBOL, run genmain -vn.
- If you are running Micro Focus COBOL, run genmain -mn.

This will create a library called libnargs.a. When you compile and link batch programs (that use this interface) specify these linker options:

```
-L$TIPROOT/lib -lbat -lnargs .
```

tipbatpi.o Interface Subroutine

The tipbatpi.o routine has several entry points. The names of the calls to tipbatpi.o begin with "**BAT**" (for batch) to avoid confusion with the on-line "**TIP**" FCS calls.

BATACTIV

To find out if TIP is active, the batch program can call **BATACTIV**.

This call establishes the connection with TIP and must be issued before any other BAT calls.

Example:

```
05  BATFLG                PICTURE X.
    88  SERVER-ACTIVE         VALUE "Y".
    88  SERVER-INACTIVE       VALUE "N".
    CALL "BATACTIV" USING    BATFLG
```

BATFCS, BATPIB

When you call BATFCS you may pass any valid parameters that TIPFCS will accept. Use the function codes in the TC-FCS copy book. The status codes are the same, as well.

Example:

```
CALL "BATFCS" USING     FCS-OPEN
                        MYLFN.

CALL "BATFCS" USING     FCS-GET
                        MYLFN
                        MYREC
                        MYKEY.

CALL "BATFCS" USING     FCS-CLOSE
                        MYLFN.
```

The function codes, logical file name packet, record area and key area typically are defined in the batch program's WORKING-STORAGE SECTION.

Status codes are returned in the 9th byte of the file name packet. To get the usual PIB status as well, define a PIB in WORKING-STORAGE with the TC-PIB copy book. Then inform BATFCS you wish to use the PIB by calling BATPIB once, early in the program execution.

Example:

```
01  MYPIB.                 COPY TC-PIB OF TIP.
 ....
    CALL "BATPIB" USING     MYPIB
```

Batch Commit and Rollback

By default, BATFCS commits each record update (ADD, PUT, DELETE) as it occurs (by calling FCS-TREN).

A logical transaction (such as moving money from one account to another) may update several records. Taking money out of the first account without adding it to the second account tends to make customers very angry. You want to commit both accounts at the same time - or rollback both of them. Fortunately, your batch applications can do this with BATCOMIT and BATROLBK.

Your application should call "BATCOMIT" to establish the start of a *"transaction"* (update sequence). The initial call to BATCOMIT turns off the automatic commit behavior of BATFCS. Your application can now perform record ADD, PUT, and DELETE operations and issue BATCOMIT or BATROLBK as required. The intended use of this option is for casual updates of files that TIP is managing. If you have a large

number of records to process it is recommended that you close the file in TIP and then run your batch update program without using BATFCS.

BATCOMIT:

Commit record updates since the last commit or rollback and mark the beginning of a new transaction or rollback point. Once updates have been committed they cannot be rolled back.

```
CALL "BATCOMIT"
```

BATROLBK:

Roll back (reverse) any record updates since the last commit or rollback and mark the beginning of a new transaction or rollback point.

```
CALL "BATROLBK"
```

Additional Considerations:

- The purpose of BATCOMIT is to maintain the integrity of related records. Your application should call BATCOMIT when a logical unit of processing has completed.
- Do not try to commit all records for a large file as a single transaction - the key holding table for TIP could be filled to capacity. The key holding table is maintained in TIP's Global System shared memory.
- To see how much Global System shared memory is currently being used by TIP, run "status s". The last status line will be of the form:

```
Of -M memory: Current free: 172K Most used: 27K
```

Performance:

Batch programs using the TIP FCS batch interface may execute more slowly. TIP performs additional I/O to insure the integrity and consistency of the data. TIP checks every record update request against all other requests from the on-line system. TIP may also hold before-images during updating. In addition, increased I/O may occur from journalizing of updates.

If your batch program updates many records and the file can be closed to TIP, you may want to continue that practice. If only a few updates are done and file closing is hard to schedule, the added overhead of the batch interface may be worth the assurance of data integrity and the ease of operation.

Journalizing both on-line and batch updates will count as an overall improvement in system operation. File recovery from a single source and with a single method is easier and safer.

Security:

When a batch program uses the TIP FCS batch interface, the user id of the person running the batch program is used to log into TIP. It is best if the user is properly defined to TIP. All appropriate TIP security and group

searching is then enforced. If updates are journalized, this information quickly identifies the source of any file change.

PCXFER - PC File Transfer

TIP provides reentrant subroutines that TIP native mode programs may call to transfer data to and from a Personal Computer running MS-DOS. The subroutines are:

- TIPH2P - Copy from HOST to the computer.
- TIPP2H - Copy from computer to the HOST.

These TIP native mode programs that use these routines must be running either the MS-DOS or MS-Windows based versions of TIP/fe.

The user interface with PCXFER is similar to that used by TIPFCS calls (that is: function-code, filename, record). However, a fourth parameter is required that indicates a user supplied work area which PCXFER uses as a buffer. The filename (2nd parameter) is used to indicate the MS-DOS filename (source or destination).

PCXFER handles variable length records with no maximum length. A TIP native mode program issues calls to the computer transfer subroutines to perform the following functions:

- OPEN Initiate the interface to TIPH2P or TIPP2H.
- GET Retrieve a record image from an MS-DOS file (TIPP2H)
- PUT Pass a single record image to an MS-DOS file (TIPH2P).
- FLUSH Force TIPH2P to empty its internal buffer.
- CLOSE Terminate the interface with TIPH2P or TIPP2H.

File Transfer Interface copybooks

There are copybooks defined which can be used when passing parameters to the PCXFER functions. They are as follows:

TC-PCFIL (File Packet) copybook

The format of the file packet is defined by the copybook TIP/TC-PCFIL and should be included in the LINKAGE SECTION of the program.

```
05  FILE-PKT.                  COPY TC-PCFIL
                               OF TIP.
*
* FILE PACKET FOR HOST/computer TRANSFER
*
    10  PCFIL-DRIVE            PICTURE X.
    10  PCFIL-FILE-NAME        PICTURE X(8).
```

```
10  PCFIL-EXTENSION     PICTURE X(3).
10  FILLER              PICTURE X(4).
10  PCFIL-STATUS        PICTURE X.
10  PCFIL-ACKNOWLEDGE   PICTURE X(4).
```

The following is a description of the fields that make up the TC-PCFIL copybook:

PCFIL-DRIVE
Specifies the drive designator on which the MS-DOS file is to be read or written. Specify a drive letter between "A" through "Z" (inclusive).

PCFIL-FILE-NAME
The filename of the MS-DOS file to be accessed and must conform to MS-DOS rules.

PCFIL-EXTENSION
The three character extension name used for this file.

PCFIL-STATUS
A status byte that is set to the same return status value as PIB-STATUS (except during FCS-OPEN when the FILE-PKT existence has not been conclusively established).

PCFIL-ACKNOWLEDGE
Not used at this time.

TC-PCINF (Info Packet) copybook

The format of the information packet is defined by the copybook TIP/TC-PCINF and should be included in the LINKAGE SECTION of the program. This packet is required only with the call to the FCS-OPEN function.

```
05 INFO-PKT.                    COPY TC-PCINF
                                    OF TIP.
 *
 * COPY ELEMENT FOR MSDOS TRANSFER INFO PACKET
   10  PCINF-BUF-LEN          PICTURE 9(4)
                              COMP-4 SYNC.
   10  PCINF-ERR-TERM         PICTURE X(4).
   10  PCINF-INDEX            PICTURE X.
   10  PCINF-OPTIONS          PICTURE X(8).
   10  PCINF-SEPARATOR        PICTURE X(2).
   10  PCINF-END-OF-FILE.
       15  PCINF-PROMPT       PICTURE X(2).
       15  PCINF-MAX-REC-LEN PICTURE 9(4).
       15  FILLER             PICTURE X(10).
   10  PCINF-CONTROL-CODE     PICTURE X.
       88  PCINF-SPACE-SUPR    VALUE " ".
       88  PCINF-NO-SUPR       VALUE "N".
       88  PCINF-HEX-WITH-SS   VALUE "B".
```

```
           88  PCINF-HEX-WOUT-SS   VALUE "H".
           88  PCINF-TRANSLATE     VALUE "T".
       10  FILLER                PICTURE X(10).
       10  PCINF-COMMENTS        PICTURE X(60).
       10  PCINF-COMPRESS        PICTURE X.
       10  PCINF-RESERVED        PICTURE X(23).
```

The following is a description of the fields that make up the TC-PCINF copybook:

PCINF-BUF-LEN

Specifies the length of a buffering area in the user program into which the FCS-OPEN function blocks record images into screen images for efficient data communication transfer.

This is a numerical value, which is the length of the buffering area. Set this field before issuing the FCS-OPEN function. The minimum buffer size is 768 bytes; the recommended buffer size is 2560. In general, the larger the buffer, the greater the efficiency of the transfer subroutines.

PCINF-ERR-TERM

Not used at this time.

PCINF-INDEX

Not used at this time.

PCINF-OPTIONS

Not used at this time.

PCINF-SEPARATOR

Not used at this time.

PCINF-END-OF-FILE

Not used at this time.

PCINF-PROMPT

Not used at this time.

PCINF-MAX-REC-LEN

This field is only used on transfers from an MS-DOS file to the host. This field should contain the length of the largest record expected from the MS-DOS file.

PCINF-CONTROL-CODE

This character determines the type of transfer to take place.

The following values are recognized:

space Used when transferring purely graphic character data. In this mode of operation, spaces at the end of a line are suppressed.

N Used when transferring purely graphic character data. In this mode of operation, NO trailing space suppression is to take place.

ALL other values, including 'B', 'H' and 'T', result in the data being transferred in binary mode (the exact data is transferred between TIP and MS-DOS).

PCINF-COMMENTS
Not used at this time.

PCINF-COMPRESS
Not used at this time.

PCINF-RESERVED
This field is reserved for future use.

TC-PCBUF (Transfer Buffer Packet) copybook

The format of the transfer buffer packet is defined by the copybook TIP/TC-PCBUF and should be included in the LINKAGE SECTION of the program.

```
COPY TC-PCBUF OF TIP.
*
* COPY ELEMENT FOR MSDOS TRANSFER BUFFER PACKET
* USER PROGRAM SHOULD NOT MODIFY THESE FIELDS
    10  PCBUF-AREA.
        15  PCBUF-LENGTH    PICTURE 9(8)
                             COMP-4 SYNC.
        15  FILLER          PICTURE X(2556).
```

The following is a description of the fields that make up the TC-PCBUF copybook:

PCBUF-LENGTH
While the transfer interface is open, this field contains the length of the buffer. This field is for informational purposes only and must not be modified by the user program.

Record Area Packet copybook

The format of the record area packet is defined by the copybook TIP/TC-PCREC and should be included in the LINKAGE SECTION of the program.

The record packet is a variable length record containing a length and the data transferred to and from the MS-DOS file. The program MUST define the appropriate record fields immediately after PCREC-DATA. Multiple record types are handled by redefinition.

```
05 RECORD-PKT.              COPY TC-PCREC
```

```
                                        OF TIP.
*
* COPY ELEMENT FOR MSDOS TRANSFER RECORD PACKET
    10  PCREC-LENGTH          PICTURE 9(4)
                              COMP-4 SYNC.
    10  PCREC-CONTROL         PICTURE X.
    10  FILLER                PICTURE X.
    10  PCREC-DATA.
*
* USER SUPPLIED RECORD LAYOUT FOLLOWS HERE
```

The following is a description of the fields that make up the TC-PCREC copybook:

PCREC-LENGTH

The length of the entire record packet.

For FCS-GET:
There is no maximum record length if you set the PCINF-OPTION field to spaces. This field must be explicitly set before each call to FCS-GET. The length specified must include the 4 bytes preceding the actual data. After a call to FCS-GET this field is set by the PCXFER interface to the actual record length (including four byte header) returned from the MS-DOS file.

For FCS-PUT:
The actual length of the PUT (e.g., minus trailing spaces) is returned. This field should be specified explicitly before each call to FCS-PUT.

The length specified must include the four bytes preceding the actual data. For example, if the record length was 256 bytes long, move 260 to PCREC-LENGTH. There is no maximum record length if you set the PCINF-OPTION field to spaces.

PCREC-CONTROL

If set to "M", to indicate record masking, the record is ignored since record masking is not required.

PCREC-DATA

This field contains the data of the record to be transferred.

PCXFER Masking

File transfer between TIP and MS-DOS does *not* require any masking functionality. This functionality previously was needed to handle EBCDIC or Binary data, but is not needed since data on both TIP and MS-DOS is in ASCII format and is transferred in binary image. Any masking functionality supported on TIP/30 will still be maintained but it is ignored.

Transfer from/to MS-DOS File

File transfer is in either graphical text mode or in binary image mode. In graphical text mode, set PCINF-CONTROL-CODE to either a space or ‘N’ depending on whether trailing space suppression is required or not. Then, each record will be returned from the MS-DOS file as indicated by line feed separators or will be written to the MS-DOS file with a line feed separator appended to the record.

ALL other values for PCINF-CONTROL-CODE will transfer the file in binary image mode with no concern for line feed separators. The data will be returned exactly as it is in the MS-DOS file or will be written exactly as in the record being written to the MS-DOS file.

PCXFER Compression

Data compression is handled internally by the PCXFER routines. Any previous support for compression for TIP/30 is maintained but ignored. The computer-COMP routine on MS-DOS is not required for file transfer between TIP and MS-DOS.

FCS-OPEN - Open PCXFER Interface

Establish the interface to the file transfer subroutines by issuing a call to the specific subroutine with a function code of "FCS-OPEN". This call serves to initialize the transfer facility. It is used to establish the desired MS-DOS file destination or source and to specify transfer options that are required.

A header record may be written to the MS-DOS file during the open. (See PCINF-COMMENT field for further details).

Syntax:

```
CALL "TIPH2P" USING     FCS-OPEN
                        file-pkt
                        info-pkt
                        pc-buffer

CALL "TIPP2H" USING     FCS-OPEN
                        file-pkt
                        info-pkt
                        pc-buffer
```

Where:

FCS-OPEN
This function code (normally defined via the supplied

copybook TIP/TC-FCS) indicates that the desired function is to OPEN the interface.

file-pkt

Use this filename packet to specify the MS-DOS file and drive that TIPH2P or TIPP2H is to use. When transferring records to the computer, if the filename and extension match an existing file, the data in that file is overwritten. The program does not receive any notification if this occurs and the file is allocated if it does not already exist.

See the description of the copybook TIP/TC-PCFIL for further information.

info-pkt

Information packet required only on the call to TIPH2P or TIPP2H with the FCS-OPEN function.

ee the description of the copybook TIP/TC-PCINF for further information.

pc-buffer

The fourth parameter on the call with a FCS-OPEN function code identifies the buffer that the user program provides for PCXFER subroutines to use. This buffer must be a minimum of 768 bytes and may be as large as 2560 bytes. This area must be fullword aligned. The program need not initialize this buffer.

he program should not modify any field in this buffer from the time an FCS-OPEN function is issued to the time an FCS-CLOSE function is issued.

ee the description of the copybook TIP/TC-PCBUF for further information.

Error Conditions:

PIB-STATUS	Meaning
PIB-FUNCTION	An IO Error has occurred opening the MS-DOS file or one of the parameters is invalid. (not one of: FCS-OPEN, FCS-PUT, FCS-FLUSH or FCS-CLOSE)
PIB-BREAK	Buffer not initialized, not opened or error retrieving first buffer of data from the computer.
PIB-IO-ERROR	Invalid parameter list.
PIB-NO-MEM	Buffer length too small (less than 1024 bytes).

PIB-STATUS	Meaning
PIB-NOT-FOUND	Destination or error terminal not found.
PIB-WRONG-MODE	Invalid screen number (only 1 through 8 is valid), or non-graphic data found in the field PCINF-COMMENTS.
PIB-EOF	The DOS file does not exist.

Additional considerations:

- Once a successful FCS-OPEN function is performed, the program should not terminate without issuing an FCS-CLOSE function for the PCXFER interface. Failing to properly close the interface can leave buffered data that has not been written to the output file.

Note: After this function is called, the output to the screen may interfere with the operation of following calls to FCS-GET. Therefore you should avoid any output to the screen between FCS-OPEN and FCS-CLOSE of TIPP2H.

FCS-GET - Input Record from computer

Call the TIPP2H subroutine repeatedly when you need to input records. The format of the record that is passed is described below. The program issues the FCS-GET function and receives the data from the MS-DOS file in the designated RECORD-PKT area.

Syntax:

```
CALL "TIPP2H" USING     FCS-GET
                        file-pkt
                        record-pkt
                        buffer
```

Where:

FCS-GET
This function code, as defined by the supplied copybook TIP/TC-FCS, indicates that you wish to retrieve a record from the interface.

file-pkt
File name packet that specifies the drive and MS-DOS file that TIPP2H is to use.

See the description of the copybook TIP/TC-PCFIL for further information.

record-pkt
This is the record area into which FCS-GET returns the MS-DOS data.

See the description of the copybook TIP/TC-PCREC for further information.

buffer

The buffer that is assigned for use by TIPP2H as given on the FCS-OPEN function call.

See the description of the copybook TIP/TC-PCBUF for further information

Error Conditions:

PIB-STATUS	Meaning
PIB-EOF	This status is returned at end of file.
PIB-NO-MEM	This status is returned if the record length is longer than the buffer provided. This applies only if the default options for PCINF-OPTIONS are not used.
PIB-BREAK	This status is returned if an error has been detected at the computer

FCS-PUT - Output Record to computer

The TIPH2P subroutine is called to output each record. The format of the record that is passed is described below. The program issues the FCS-PUT function to deliver the record to the MS-DOS file from the RECORD-PKT.

The programmer must keep in mind that TIPH2P is blocking the records into the transfer buffer to build a screen full of data. This means that the line that is passed with a FCS-PUT function to TIPH2P may not be physically transferred at the time the call is issued! Also see the description of the FCS-FLUSH function of TIPH2P.

Syntax:

```
CALL "TIPH2P" USING     FCS-PUT
                        file-pkt
                        record-pkt
                        buffer
```

Where:

FCS-PUT

This function code, as defined by the supplied copybook TIP/TC-FCS, indicates that the desired function is to output a record to the MS-DOS.

file-pkt

Use this file name packet to specify the drive and MS-DOS file that TIPH2P is to use. See the description of the copybook TIP/TC-PCFIL for further information.

record-pkt

This is the record packet that contains the data to be transferred. See the description of the copybook TIP/TC-PCREC for further information.

buffer

The buffer that is assigned for use by TIPH2P as given on the FCS-OPEN function call. See the description of the copybook TIP/TC-PCBUF for further information.

Error Conditions:

PIB-STATUS	Meaning
PIB-NO-MEM	This status is returned if the record length is longer than the buffer provided.
PIB-WRONG-MODE	This status is returned if non-displayable characters are detected in the record data. If this status appears when not expected, double check the parameter list supplied on the call; the TIPH2P subroutine may not be examining the same data area that you think is being examined!
PIB-BREAK	This status is returned if an error has been detected at the computer.

FCS-FLUSH - Flush PCXFER Buffer

Since TIPH2P is buffering the records that the user program is passing, the program may need to flush the content of the TIPH2P buffer prematurely. Normally you need not consider issuing the FCS-FLUSH operation, an automatic flush occurs when the buffer fills and when a close is issued.

Syntax:

```
CALL "TIPH2P" USING     FCS-FLUSH
                        file-pkt
                        dummy
                        buffer
```

Where:

FCS-FLUSH
This function code, defined by the supplied copybook TIP/TC-FCS, indicates that the desired function is to flush the TIPH2P buffer.

file-pkt
This file name packet specifies the drive and MS-DOS file that TIPH2P is to use.

See the description of the copybook TIP/TC-PCFIL for further information.

dummy
The third parameter is a dummy parameter (the usual record packet could be used) that is present only to preserve symmetry with the other calls to TIPH2P. Record data cannot be provided with a call to the FCS-FLUSH function - it is ignored.

buffer
The buffer that is assigned for use by TIPH2P as given on the FCS-OPEN function call.

See the description of the copybook TIP/TC-PCBUF for further information.

Error Conditions:

PIB-STATUS	Meaning
PIB-LOCKED	This status is returned if the buffer is about to be flushed, serial resources are locked (file in sequential mode) and this PUT was rejected. The CALL should be re-submitted when the file is taken out of sequential mode.

Additional considerations:

- The FLUSH operation delivers any buffered record data that is in the TIPH2P buffer. This normally only occurs when the buffer is full. Since flushing defeats blocking and increases communication overhead, perform this operation only when your program must be certain that terminal I/O occurs at a specific time (for example, when your program is awaiting further input to a background process).

FCS-CLOSE - Close PCXFER Interface

When your program has finished transferring records, the program must close the interface to PCXFER. The FCS-CLOSE function automatically

flushes any buffered data (see description of the FCS-FLUSH function in the previous section).

If the program does not issue an FCS-CLOSE function to TIPH2P or TIPP2H, unpredictable results may occur; one real possibility is the potential loss of the last buffer of data.

Syntax:

```
CALL "TIPH2P" USING     FCS-CLOSE
                        file-pkt
                        dummy
                        buffer

CALL "TIPP2H" USING     FCS-CLOSE
                        file-pkt
                        dummy
                        buffer
```

Where:

FCS-CLOSE

This function code, as defined by the supplied copybook TIP/TC-FCS, indicates that the desired function is to close the PCXFER subroutine interface

file-pkt

File name packet that specifies the drive and MS-DOS file that TIPH2P and TIPP2H are to use.

See the description of the copybook TIP/TC-PCFIL for further information

dummy

The third parameter is a dummy parameter (the usual line packet could be used) that is present only to preserve symmetry with the other calls. Record data cannot be supplied with a call to the FCS-CLOSE function - it is ignored.

buffer

The buffer that is assigned for the subroutines use as described in the previous section.

See the description of the copybook TIP/TC-PCBUF for further information.

Error Conditions:

PIB-STATUS	Meaning
PIB-EOF	For TIPH2P, this status is returned if an I/O error is detected on an implied FLUSH or an EOF string was not detected. For TIPP2H, this status

PIB-STATUS	Meaning
	is returned if an I/O error is detected while attempting to close the PCXFER interface.

Additional considerations:

- The CLOSE operation delivers any buffered data that is in the transfer buffer. There is no need to flush the buffer explicitly before issuing the close function.
- Once the CLOSE operation closes the interface, the program may reopen the interface and start another transfer.
- The CLOSE operation also guarantees that the computer software/hardware is notified that the file transfer operation is complete - this can prevent subsequent file transfer attempts from having problems.

Compiling and Testing Application Programs

Supported COBOL Compilers

Inglenet has verified two COBOL compilers for use with TIP/ix on UNIX/Linux:

- Micro Focus COBOL Server Express
- OpenCOBOL 1.1 (open source)
- COBOL-IT Enterprise Edition (supported and updated version of OpenCOBOL)

Some platforms do not support both COBOL compilers. Check the TIP/ix Release Notice to see which compilers are supported on your platform.

TIP/ix supplies makefiles (in **$TIPROOT/src/tip**) for these compilers. Inglenet recommends that you use these files as examples to construct your own make files.

Most UNIX systems provide a **make** utility and a C compiler.

Micro Focus COBOL

The Micro Focus COBOL compiler provides a high degree of compatibility with the OS/3 COBOL-74 and COBOL-85 compilers.

The Micro Focus COBOL compiler needs the following environment variables at *compile* time:

```
COBCPY=$TIPROOT/include;dir2;dir3...
```

4. Specify the directories holding copybooks. This should include $TIPROOT/include.

On some platforms, the following environment variables are required at *run* time:

COBDIR=directory

5. Specify the directory where the compiler is installed. Usually:
   "/usr/lib/cobol".

   Note: If you use Micro Focus COBOL in a dynamic linking environment and:re-entrant programs, or background transactions, or distributed transaction processing (DTP) transactions, or IMS programs with REUSE option then the tipix.conf file must contain an COBDIR entry.

LD_LIBRARY_PATH=$COBDIR/coblib

6. If you want to use re-entrant programs, you must code this environment variable in the tipix.conf file to specify the location of the Cobol libraries.

LD_RUN_PATH=$COBDIR/coblib

7. Specify the run path if it is required on your platform.

For details, see your Unix and Micro Focus documentation. For a combined list of TIP, HSP/80, HSP/22, and other related environment variables, browse the file:

$TIPROOT/arm/scripts/arm.tipsetenv.

Types of Executables

Micro Focus COBOL can generate two kinds of executable programs:

- statically linked executables
- dynamically linked executables

For the statically linked type, the executable contains everything it needs to run.

For the dynamically linked type, the executable depends on a shared run-time object library supplied by Micro Focus. When you execute a program, Unix loads the program into your session's memory (address space), and attempts to resolve references to subroutines stored in any shared object libraries. If all the references are resolved, the program is given permission to run. Otherwise, the program is aborted.

The advantage of dynamic linking is that you get smaller executables.

The disadvantages of dynamic linking are that some of the newer security features of TIP are sacrificed, and that executables take longer to load into memory.

TIP release 1.7 and higher may be installed as either a secured or unsecured OLTP system. For details, see **fixperms** in the *TIP Utilities* manual.

Some of the security functions in TIP are implemented with a Unix feature called "setuid".

Essentially, certain special executables are owned by a designated privileged Unix user, and when these programs are executed on behalf of an unprivileged user, some restricted access is granted (temporarily) to the otherwise unprivileged user.

However, Unix refuses to run "setuid" executables which dynamically link in a shared object library at run-time. This is a necessary measure to bar any Trojan Horse (in the form of a malicious shared object library module) from infiltrating the Unix system.

COBOL Makefiles

The following makefiles show how to compile on-line or batch COBOL programs (with or without debugging).

The makefiles presume the following meanings for file name extensions:

Extension	TIP COBOL transaction
cbl	TIP COBOL transaction
ims	IMS COBOL transaction
bat	Batch COBOL program
rpg	RPG program (RPG source code is converted to COBOL and then compiled). This option requires the Heritage Support Package (HSP).
tip	COBOL program that calls 1100/2200 APIs.

The makefiles, distributed with TIP in the directory $TIPROOT/src/tip, are only examples. They contain comments to guide you when editing them to meet your needs. The Unix **make** program is a complex utility with many features. If you intend to use it, study your Unix documentation thoroughly. See your COBOL compiler documentation for compiler-specific information beyond the scope of this book.

There are two main make files (**make.mf** for Micro Focus and **make.oc** for OpenCOBOL and COBOL-IT). When the Unix **make** utility is invoked, it looks for a file called **makefile**. When TIP is installed, it creates a Unix

file link that points the file **makefile** to either **make.oc** or **make.mf** depending on which COBOL compiler is installed.

To use these make files, copy them to the directory that contains your source programs and invoke the **make** utility with the appropriate makefile:

```
make [-f xxxxx.xxx] target
```

Where:

-f This option tell the make utility to use a specific makefile. Normally this option is not required, which results in the file makefile being used. When TIP is installed this file will be linked to either make.mbp or make.mf.

xxxxx.xxx
The makefile to use (make.mf or make.mbp).

target
The name (without extension) of the target executable to "make". The make program infers the name of the source components needed to create the target.

Example:

If your source program is called PAY020.cob you can invoke the Micro Focus makefile with this command:

```
make -f make.mf PAY020
```

If the file **makefile** is linked to **make.mf**, then the program PAY020 can be compiled with the following statement.

```
make PAY020
```

The distributed makefiles put the executable in the location defined by the BIN symbol. You may have to fine tune (edit) the value of BIN in the makefiles.

NOTE: These sample makefiles are provided as working examples and you may need to make changes for your local setup and conventions.

Micro Focus COBOL Makefile (make.mf)

```
#
# Makefile for compiling TIP application programs using MF COBOL.
#
# This 'makefile' is provided as an example. Please edit it
# to suit your local situation.
#
# Unless you are using multiple COBOL compilers and need to easily
# switch between them, it is more convenient to rename this file to
# Makefile or makefile. (i.e. cp make.mf Makefile).
#
# If you are using MF COBOL, the install script has linked makefile
# to make.mf. This lets you execute a make command without having
# to specify the make file.
#
# Uncomment the following definition if you want to use multiple COBOL
```

```
# compilers, or have not renamed this file to be either makefile or
Makefile,
# or have removed the link of makefile to make.mf.
#
# MAKEFILE= -f make.mf
#
# TIP COBOL transactions should be suffixed with .cbl or .cob
# TIP C transactions should be suffixed with .c
# IMS/90 COBOL transactions should be suffixed with .ims
# Batch COBOL programs should be suffixed with .bat
# RPG programs should be suffixed with .rpg
#

.SUFFIXES: .c .ims .cbl .cob .bat .rpg .tip
#
# Set up some macros to make any future changes easier.
#
TIPINC = $(TIPROOT)/include
TIPLPATH= -L$(TIPROOT)/lib
LIBBAT = -lbat
LIBIMS = -lims
LIBTIP = -ltip
TIPISAM = -ldisam
#
# By default, any binaries created will be moved into $TIPROOT/bin.
# If you want to put your binaries into a different directory then change
# the following definition of BIN.
#
BIN = $(TIPROOT)/bin
#
# If you choose to put copybooks in a directory other than
# $TIPROOT/include then set SITEINC to that directory. You may
# enter multiple directories but be sure to follow the format
# of the template supplied.
#
# Note: when entering the directory name remove the "<>" characters.
# For an example, see the definition of TIPINC above.
# Be sure to add "$(SITEINC)" to the rules that compile your COBOL
# programs.
#
# SITEINC =:<enter your include directory here>
#
# If you choose to put create object libraries in a directory other than
# $TIPROOT/lib then set SITELIB to that directory.
#
# Note: when entering the directory name remove the "<>" characters.
# For an example, see the definition of TIPLPATH above.
#
#SITELPATH = -L<enter your library directory here>
LIBPATH = $(TIPLPATH) $(SITELPATH)
COBOL = cob
#
# By default this make file compiles programs without including any
# debugging information. With out this information source level debugging
# will not work. If you want to have debugging on by default then
uncomment
# the MFOPT definition that contains the debug option.
# In addition, debugging can be invoked on a per compile basis by issuing
# the following command:
# make MFOPT=-gUa <program>
# where <program> is the name of your source file without the extension.
#
# MFOPT may be used to set any Micro Focus COBOL parameters that you
would
# like to use. A number of examples follow.
#
# Generic debugging compile options
#MFOPT = -gUa
# Online compile options for debugging with animation
#MFOPT = -gUa -I TIPFCS -I TIPMSGO -I TIPMSG -I TIPH2P -I TIPSUB
# Batch compile options for debugging with animation
```

```
#MFOPT = -gUa -I BATFCS -I INITFTAB
# Default compile options for no debugging.
MFOPT = -Ox
MFCOB = -P -X mFFH -X ADIS -C "NOWARNING VSC2 IBMCOMP REF XREF"
MFBAT = -C "NOWARNING VSC2 IBMCOMP REF XREF NODETECTLOCK"
# FILESHARE should prevent opening a file for exclusive use
#MFBAT = -C "NOWARNING VSC2 IBMCOMP REF XREF NODETECTLOCK FILESHARE"
#
# If compiling on a Data General DG/UX system set the environment
# variable TARGET_BINARY_INTERFACE to m88kdguxcoff
#
#TARGET_BINARY_INTERFACE="m88kdguxcoff"
MV = mv -f
RM = rm -f
CC = cc
CLIB = -lcurses -lc
GENMAIN = $(TIPROOT)/bin/genmain
ARMRPG = $(TIPROOT)/bin/armrpg
ARMOPT = -mf
# Following is for TIP transactions using Micro Focus COBOL
.cob .cbl:
    $(COBOL) $(MFCOB) $(MFOPT) -c -k $<
    $(GENMAIN) -tm $* main$*.c
    $(COBOL) $(MFCOB) $(MFOPT) main$*.c $*.o -o $* $(LIBPATH) $(LIBTIP)
    $(CLIB)
    $(MV) $* $(BIN)
    $(RM) $*.idy $*.int $*.lst $*.o main$*.o main$*.c
# Following is for IMS transactions using Micro Focus COBOL
.ims:
    $(COBOL) $(MFCOB) $(MFOPT) -c -k $<
    $(GENMAIN) -im $* main$*.c
    $(COBOL) $(MFCOB) $(MFOPT) main$*.c $*.o -o $* $(LIBPATH) $(LIBIMS)
    $(CLIB)
    $(MV) $* $(BIN)
    $(RM) $*.idy $*.int $*.lst $*.o main$*.o main$*.c
# Following is for 1100 transactions using Micro Focus COBOL
.tip:
    $(COBOL) $(MFCOB) $(MFOPT) -c -k $<
    $(GENMAIN) -1m $* main$*.c
    $(COBOL) $(MFCOB) $(MFOPT) main$*.c $*.o -o $* $(LIBPATH) $(LIBS2200)
    $(CLIB)
    $(MV) $* $(BIN)
    $(RM) $*.idy $*.int $*.lst $*.o main$*.o main$*.c
# Following is for batch Cobol programs
.bat:
    $(COBOL) $(MFBAT) $(MFOPT) -c -k $<
    $(COBOL) $(MFBAT) $(MFOPT) $*.o -o $* $(LIBPATH) $(LIBBAT) $(CLIB)
    $(MV) $* $(BIN)
    $(RM) $*.o
# Following is for RPG to COBOL
.rpg:
    $(ARMRPG) $(ARMOPT) $(@F).rpg
    make $(MAKEFILE) $(@F)
# Following is for TIP transactions written in C language
.c:
    $(GENMAIN) -cm $* main$*.c
    $(CC) $(CFLAGS) main$*.c $< -o $* $(LIBPATH) $(LIBTIP) $(CLIB)
    $(MV) $* $(BIN)
    $(RM) $*.o main$*.o main$*.c
```

Debugging on-line programs

Finding out why your program isn't performing can be difficult. TIP supports a number of tools and techniques to help software developers' track down elusive bugs. These tools fall into three categories:

- Debug logs
- Embedded debugging statements
- Compiler-supplied source level debugging

TIP provides a facility that will create log files containing information (in ASCII) about the execution of an on-line program, or suite of on-line programs. Since on-line programs execute under control of the TIP system and make many requests into TIP (with COBOL CALL statements), the log contains information related to each of these CALLs. The log files do not normally contain any information about what your program is doing between CALLs to TIP. If the standard log files are not sufficient to track down a problem, then you may want to look at the additional facilities outlined in the following sections.

Activating the log file

The TIP debug log can be created several ways:

`tipix -[a]d`

A lowercase "d" means create a separate log file for each transaction program that is executed.

The log files are called "log.TRID" where TRID is the transaction code used to schedule each program.

The "-a" option specifies that the log is to contain "all" information. This results in a more detailed log file

For details about tipix command options, see the TIP Utilities.

`tipix -[a]D filespec`

A capital "D" means create a single log file called "filespec" containing all the debug log information for the entire user session (from entering tipix through to the fin command).

The "-a" option indicates that the log is to contain "all" information. This results is a more detailed log file. The filespec value can either be a filename (in which case the file is created in the user's home directory), or a full path name specifying the log file to create.

`*TRID`

Entering an asterisk (*) character immediately preceding the transaction code activates the debug log for the execution of this transaction only.

`SMPROG`

When you define a transaction to TIP/ix with the SMPROG utility, you can set the logging attribute to "Min or All" to create a debug log every time the on-line program is scheduled. This is useful when you are debugging a program that is either called by another program (such as

TIPSUB, TIPXCTL, TIPFORK etc.) or is started automatically by the TIP system (by a TIPQUEUE server program).

Examples:

- This example starts the TIP shell with transaction logging turned on. A detailed log (because of the “a” option) will be created for every transaction that is run.

```
tipix -da
```

- This example starts the TIP shell with session logging turned on. A summary log (because no -a option was specified) will be created. All of the log information will be written to a single file called log.session. Since no path information is given, the log file will be created in the user’s home directory.

```
tipix -D log.session
```

- This example assumes that the TIP shell is already active. It runs the transaction quereq with logging turned on. The log file, log.QUEREQ, is stored in the user’s home directory. By default, the log is a summary log unless you started the TIP shell with the “-a” option.

```
quereq mainque,12,monday
```

- This example starts the TIP shell with logging turned off. However, the “-a” option indicates that if any log files are created later, they will be created as “all” style logs (meaning that they will contain more detailed information than a summary log does). These log files may be created using the *trid feature or may be created directly by user programs using the TIPLOG FCS-OPEN function.

```
tipix -a
```

Embedded Debugging Statements

Sometimes a programmer may want to add application-oriented information to the TIP log file to track down a problem that cannot be uncovered using the standard log information. TIP provides the TIPLOG, TIPDUMP and TIPSNAP functions to help track down these problems.

TIPLOG - Updating the Log File

TIPLOG gives you the ability to add extra information to the log file to help you track down an application error. TIPLOG can write a single line of textual information to the log, or dump an area of memory to the log in hexadecimal format.

Besides writing information to the log file, TIPLOG can also open and close the log file. This can be useful if an abnormal condition occurs and the application program wants to write information to a log file — even if a log file was not requested. In such cases, the program would open the log file, write the appropriate information to the log, and then close the log. If

an application does create a log file dynamically like this, it should inform the end user of the event. An error message such as the following might be appropriate:

```
Severe error occurred - trace information written to log file
```

Open the Log File

This function opens a debugging log file in the user's home directory. The file is called "log.XXXXXXXX" (where XXXXXXXX is the transaction code of the program)

Syntax:

```
CALL "TIPLOG" USING     FCS-OPEN
```

Where:

FCS-OPEN
Function code from the TC-FCS copy book

Additional Considerations:

- This call will be ignored if the program is defined with the "Log Never" option in SMPROG.
- This function does not return an error if the debug log is already open.
- If this function opens a new log file, it does not check to see if the new log file will overwrite an existing log file of the same name.

Close the Log File

This function closes the currently open log file.

Syntax:

```
CALL "TIPLOG" USING     FCS-CLOSE
```

Where:

FCS-CLOSE
Function code from the TC-FCS copybook

Additional Considerations:

- This function does not return an error if the debug log is not currently open.
- If the TIP shell was started with the "-D filespec" option (specifying a single log file for the entire session) then this function is ignored and the log file is not closed.
- If the program closes the current log file, then subsequently re-opens the log, the previous log file is overwritten.

Write a Text Message to the Log File

This function writes a single text message to the currently open log file.

Syntax:

```
05  RECORD                  PICTURE X(80).
CALL "TIPLOG" USING         FCS-PUT
                            RECORD
```

Where:

FCS-PUT
Function code from the TC-FCS copy book

RECORD
This is an 80-character record area that contains the character data to be written to the log file.

Example:

```
05 TEXT-MESSAGE.
    10  LOG-LITERAL-1        PICTURE X(30) VALUE
        "CREDIT BALANCE IN ERROR, CUST=".
    10  LOG-CUST-NUMBER      PICTURE 9(8).
    10  LOG-LITERAL-2        PICTURE X(10)
        VALUE ", BALANCE=".
    10  LOG-CUST-BALANCE     PICTURE
                              $ZZ,ZZZ.99CR.
    10  LOG-LITERAL-3        PICTURE X(20)
                              VALUE ".".
    MOVE REC-CUST-NUMBER      TO LOG-CUST-NUMBER
    MOVE REC-CUST-BALANCE     TO LOG-CUST-BALANCE
    CALL "TIPLOG" USING       FCS-PUT
                              TEXT-MESSAGE
```

Additional Considerations:

- This call is ignored if the program was defined with the "Log Never" option in SMPROG.
- This function does not return an error if the debug log is not open. Since no error is returned if the log file is not open, programmers may be tempted to leave these statements in programs even after they have been debugged. This is a matter of common sense and the approach taken may differ from case to case. The following points are items to consider when addressing this issue.
- Although the overhead associated with the TIPLOG function call is low, especially when the log file is not open, there is still some overhead. This means that code that is executed many times (such as the inner loop of a nested loop structure) should not contain TIPLOG CALLs if they are not actually used to log information.
- The TIPLOG function can provide useful debugging information. It is very handy to simply have the program run with the log option turned on and obtain useful information to track down a problem. If the TIPLOG statements are left in a production program, then they will be activated any time the program is run with an active log file.

Dump Memory to the Log File

This function dumps a section of memory in hexadecimal format to the currently open log file.

Syntax:

```
CALL "TIPLOG" USING      FCS-FLUSH
                         Start-1 End-1
                       [ Start-2 End-2 ]
                       [ Start-3 End-3 ]
                       [ Start-4 End-4 ]
```

Where:

FCS-FLUSH
Function code from the TC-FCS copybook

Start-n and End-n
identify the starting and ending locations of an area to be dumped in hexadecimal format to the log file. The dumped area includes the first byte of the Start-n field up to but not including the first byte of the End-n field.

Up to four pairs of parameters may be passed; each pair represents the starting and ending location of an area of memory that is to be dumped.

Example:

```
05  TAX-TABLE.
    10  TAX-ENTRY            PICTURE 9(7)V99
                             OCCURS 50 TIMES
05  TAX-TABLE-END            PICTURE XX
                             VALUE "ZZ".

    CALL "TIPLOG" USING      FCS-FLUSH
                             TAX-TABLE
                             TAX-TABLE-END
```

Additional Considerations:

- The considerations outlined above for the FCS-PUT function also apply to this function.
- Start-1 and End-1 are mandatory parameters. Start-2 through End 4 are optional. However, if they are specified, they must be supplied in pairs.
- If the area being dumped is larger than 48 bytes in length and contains all character (displayable) data, then it is dumped in character format to conserve space in the log file. If the area being dumped is either less than 49 characters in length or contains any non-displayable data, then it is dumped in hexadecimal format.

TIPDUMP - Force Program Dump

This TIP function is used to force a program dump at a specific point in the processing. Execution of the program is halted and a memory dump is taken to the log file. After the memory dump is taken, control will return to the previous program on the execution stack. This is similar to a TIPRTN function. However, with the TIPDUMP function, the previous program will receive an error status in its PIB (PIB-PROG-ABEND). The TIPDUMP function indicates abnormal termination of the transaction and will cause rollback of any updates performed by the transaction up to this point.

Syntax:

```
CALL "TIPDUMP"
```

There are no parameters.

All LINKAGE-SECTION areas, PIB, CDA, MCS and WORK are printed in Hexadecimal and the program terminates.

The dump is contained in the user's home directory in the file **log.XXXXXXXX** where XXXXXXXX is the transaction name.

Additional Considerations

- The TIPDUMP function will not create any information in the log file if the program is defined with the "Log Never" option in SMPROG.
- This call writes the information to the currently opened log file. If no log file is open, one will be created.

TIPSNAP - Snap Dump Memory

This subroutine enables a program to produce "snap" dumps of various sections of memory. The specified locations of memory are displayed in a report that is output to a file named "**log.XXXXXXXX**" where "XXXXXXXX" is replaced by the name of the transaction that invoked TIPSNAP.

Syntax:

```
CALL "TIPSNAP" USING    Start-1 End-1
                      [ Start-2 End-2 ]
                      [ Start-3 End-3 ]
                      [ Start-4 End-4 ]
```

Start-n and End-n identify the starting and ending locations of an area to be dumped in hexadecimal format to the log file.

Up to four pairs of parameters may be passed; each pair represents the starting and ending location of an area of memory that is to be dumped.

If the area being dumped is larger than 32 bytes in length and contains all character data, then it is dumped in character format to conserve space in the log file. If the area being dumped is either less than 33 characters in length or contains any non-character data, then it is dumped in hexadecimal format.

Example:

```
CALL "TIPSNAP" USING     WORK-AREA END-WORK
                         MCS END-MCS
```

Additional Considerations:

- Start-1 and End-1 are mandatory parameters. Start-2 through End 4 are optional. However, if they are specified, they must be supplied in pairs.
- If the call is made using:

  ```
  CALL "TIPSNAP" USING MCS WORK-AREA
  ```

  The call will still occur but you may not get the contents of the snap. This is because TIP startup code uses UNIX MALLOC and each area is allocated separately. It could be that the MCS and WORK-AREA may not be contiguous. If this happens, try using:

  ```
  CALL "TIPSNAP" USING MCS END-MCS
  ```

  where END-MCS is a field in the MCS area

- This call is useful when debugging programs but should be removed when placing a program in production.
- This call writes the information to the currently opened log file. If no log file is currently open, one will be created automatically by the TIPSNAP call. When this happens, TIPSNAP will automatically close the log file after the memory dump has been performed. If on the other hand, the log file is currently open, then TIPSNAP creates its report in this log file and leaves it open after the memory dump has been performed.
- This call differs from the TIPLOG FCS-FLUSH call as follows:
- TIPLOG CALLs are ignored if a log file is not currently open
- TIPSNAP will open the log file if it is not currently open
- This call will be ignored if the program is defined with the “Log Never” option in SMPROG.
- Micro Focus COBOL compiler directive "REF" should allows a programmer to correlate an address found in the TIPSNAP dump back to an address within the application program. This can speed up debugging time by allowing the programmer to find exact locations in the dump much faster than trying to progress it manually.

Inglenet does not release the "make.mf" file with this option turned on since it does make the listing much larger than usual.

Sample Log Files

The following are examples of log files created when a transaction called TSTLOG was run. The first example is of a minimum (-d) log while the second is an example of a detailed (-a) log.

Normal Log File (minimal format):

```
14:23:30 ********************************************
```

```
14:23:30 * Entering application TSTLOG Stack level 1 *
* & PIB = 800B12C4 STATUS = PIB-GOOD *
* & CDA = 08081010 size 256 *
* & MCS = 08080444 thru 08080FFC, size 3000 *
* & WRK = 08078730 thru 08080430, size 32000 *
**********************************************
14:23:30 ROLL TSTLOG OPENING TIPQUE:ARCQUE
14:23:30 QUE OPEN 2 ARCQUE
14:23:30 FCS-OPEN 2 TIP$QUE Rec# 1 PIB-GOOD
14:23:30 QUE OPEN 2 ARCQUE Rec# 62 PIB-GOOD
14:23:30 ROLL 62 Records currently in queue.
14:23:30 QUE PUT 3 ARCQUE put 60 bytes
14:23:30 FCS-GETUPW 4 TIP$QUE Rec# 1 PIB-GOOD
14:23:30 FCS-GETUPW 4 TIP$QUE Rec# 64 PIB-NOTFOUND
14:23:30 FCS-ADD 4 TIP$QUE Rec# 64 PIB-GOOD
14:23:30 FCS-GETUPW 4 TIP$QUE Rec# 64 PIB-GOOD
14:23:30 FCS-GETUPW 4 TIP$QUE Rec# 63 PIB-GOOD
14:23:30 FCS-PUT 4 TIP$QUE Rec# 63 PIB-GOOD
14:23:30 FCS-PUT 4 TIP$QUE Rec# 64 PIB-GOOD
14:23:30 FCS-PUT 4 TIP$QUE Rec# 1 PIB-GOOD
14:23:30 FCS-TREN 2 ARCQUE COMMIT PIB-GOOD
14:23:30 FCS-TREN 2 ARCQUE COMMIT PIB-GOOD
14:23:30 QUE CLOSE 2 ARCQUE
14:23:30 FCS-GETUPW 4 TIP$QUE Rec# 1 PIB-GOOD
14:23:30 FCS-PUT 4 TIP$QUE Rec# 1 PIB-GOOD
14:23:30 FCS-CLOSE 2 TIP$QUE Rec# 1 PIB-GOOD
This is a test TIPLOG call, line number=00000001.
This is a test TIPLOG call, line number=00000002.
This is a test TIPLOG call, line number=00000003.
This is a test TIPLOG call, line number=00000004.
This is a test TIPLOG call, line number=00000005.
....+....1....+....2....+....3....+....4....+....5....+....6
1 : ' SNAP 00000005 '
61 : ' 62 Records currently in queue. '
121 : ' '
000000 : 00 3C 00 00 41 4C 4C 49 4E 53 4F 4E 53 2F 31 30 '.<..ALLINSONS/10'
000010 : 41 52 43 20 39 35 30 32 32 33 31 34 32 33 33 30 'ARC 950223142330'
000020 : 30 30 4E 00 54 49 50 4C 4F 47 2C 54 49 50 4C 4F '00N.TIPLOG,TIPLO'
000030 : 47 2C 54 49 50 4C 4F 47 2C 54 49 50 4C 4F 47 2C 'G,TIPLOG,TIPLOG,'
000040 : 54 49 50 4C 4F 47 2C 54 49 50 4C 4F 47 2C 54 49 'TIPLOG,TIPLOG,TI'
000050 : 50 4C 4F 47 2C 54 49 50 4C 4F 47 2C 54 49 50 4C 'PLOG,TIPLOG,TIPL'
000060 : 4F 47 2C 54 49 50 4C 4F 47 2C 54 49 50 4C 4F 47 'OG,TIPLOG,TIPLOG'
000070 : 2C 54 49 50 4C 4F 47 2C 54 49 50 4C 4F 47 2C 54 ',TIPLOG,TIPLOG,T'
000080 : 49 50 4C 4F 47 2C 54 49 'IPLOG,TI'
14:23:30 TIPRTN
```

Detailed Log File (-a option = “all” format)

```
14:27:02 **********************************************
14:27:02 * Entering application TSTLOG Stack level 1
* & PIB = 800B12C4 STATUS = PIB-GOOD
* & CDA = 08081010 size 256
* & MCS = 08080444 thru 08080FFC, size 3000
* & WRK = 08078730 thru 08080430, size 32000
---- [CDA AREA] len[256] ----
000000 : 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 ' '
000010 : 53 4E 41 50 20 20 20 20 30 30 30 30 30 30 30 35 'SNAP 00000005'
000020 : 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 ' '
000030 : 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 ' '
000040 : 20 20 20 20 20 20 20 20 2C 2C 53 4E 41 50 2C 35 ' ,,SNAP,5'
000050 : 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 ' '
000060 : thru 00008F same as last
000090 : 20 20 20 20 20 20 20 20 00 00 00 00 00 00 00 00 ' ........'
0000A0 : 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 '............'
0000B0 : thru 0000EF same as last
0000F0 : 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 '............'
14:27:02 No files present in Active file table
**********************************************
14:27:02 ROLL TSTLOG OPENING TIPQUE:ARCQUE
14:27:02 QUE OPEN 2 ARCQUE
14:27:02 FCS-OPEN 2 TIP$QUE Rec# 1 PIB-GOOD
```

```
14:27:02 QUE OPEN 2 ARCQUE Rec# 63 PIB-GOOD
14:27:02 ROLL 63 Records currently in queue.
14:27:02 QUE PUT 3 ARCQUE put 60 bytes
14:27:02 FCS-GETUPW 4 TIP$QUE Rec# 1 PIB-GOOD
---- [Data record] len[2048] ----
000000 : 00 00 00 01 00 00 00 00 00 00 00 00 41 4C 4C 49 '............ALLI'
000010 : 4E 53 4F 4E 09 50 22 3F 01 42 32 9F 00 00 00 00 'NSON.P"?.B2.....'
000020 : 00 00 00 41 00 00 00 00 00 00 00 00 00 00 00 00 '...A............'
000030 : 41 52 43 51 55 45 20 20 00 00 00 02 00 00 00 40 'ARCQUE .......@'
000040 : 00 00 00 3F 00 00 00 00 00 00 00 00 00 00 00 00 '...?............'
000050 : 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 '................'
000060 : thru 0007EF same as last
0007F0 : 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 '................'
14:27:02 FCS-GETUPW 4 TIP$QUE Rec# 65 PIB-NOTFOUND
Rec#='000000000'
---- [Data record] len[2048] ----
000000 : 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 '................'
000010 : thru 0007EF same as last
0007F0 : 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 '................'
14:27:02 FCS-ADD 4 TIP$QUE Rec# 65 PIB-GOOD
---- [Data record] len[2048] ----
000000 : 00 00 00 41 00 00 00 00 00 00 00 00 00 00 00 00 '...A............'
000010 : 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 '................'
000020 : thru 0007EF same as last
0007F0 : 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 '................'
14:27:02 FCS-GETUPW 4 TIP$QUE Rec# 65 PIB-GOOD
---- [Data record] len[2048] ----
000000 : 00 00 00 41 00 00 00 00 00 00 00 00 00 00 00 00 '...A............'
000010 : 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 '................'
000020 : thru 0007EF same as last
0007F0 : 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 '................'
14:27:02 FCS-GETUPW 4 TIP$QUE Rec# 64 PIB-GOOD
---- [Data record] len[2048] ----
000000 : 00 00 00 40 00 00 00 00 00 00 00 00 00 00 00 00 '...@............'
000010 : 00 00 00 3C 00 00 00 3A 00 00 41 4C 4C 49 4E 53 '...<...:..ALLINS'
000020 : 4F 4E 53 2F 31 30 41 52 43 20 39 35 30 32 32 33 'ONS/10ARC 950223'
000030 : 31 34 32 33 33 30 30 30 4E 00 54 49 50 4C 4F 47 '14233000N.TIPLOG'
000040 : 2C 54 49 50 4C 4F 47 2C 54 49 50 4C 4F 47 2C 54 ',TIPLOG,TIPLOG,T'
000050 : 49 50 00 00 00 00 00 00 00 00 00 00 00 00 00 00 'IP..............'
000060 : 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 '................'
000070 : thru 0007EF same as last
0007F0 : 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 '................'
14:27:02 FCS-PUT 4 TIP$QUE Rec# 64 PIB-GOOD
---- [Data record] len[2048] ----
000000 : 00 00 00 40 00 00 00 00 00 00 00 00 00 00 00 41 '...@...........A'
000010 : 00 00 00 3C 00 00 00 3A 00 00 41 4C 4C 49 4E 53 '...<...:..ALLINS'
000020 : 4F 4E 53 2F 31 30 41 52 43 20 39 35 30 32 32 33 'ONS/10ARC 950223'
000030 : 31 34 32 33 33 30 30 30 4E 00 54 49 50 4C 4F 47 '14233000N.TIPLOG'
000040 : 2C 54 49 50 4C 4F 47 2C 54 49 50 4C 4F 47 2C 54 ',TIPLOG,TIPLOG,T'
000050 : 49 50 00 00 00 00 00 00 00 00 00 00 00 00 00 00 'IP..............'
000060 : 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 '................'
000070 : thru 0007EF same as last
0007F0 : 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 '................'
14:27:02 FCS-PUT 4 TIP$QUE Rec# 65 PIB-GOOD
---- [Data record] len[2048] ----
000000 : 00 00 00 41 00 00 00 00 00 00 00 00 00 00 00 00 '...A............'
000010 : 00 00 00 3C 00 00 00 3A 00 00 41 4C 4C 49 4E 53 '...<...:..ALLINS'
000020 : 4F 4E 53 2F 31 30 41 52 43 20 39 35 30 32 32 33 'ONS/10ARC 950223'
000030 : 31 34 32 37 30 32 30 30 4E 00 54 49 50 4C 4F 47 '14270200N.TIPLOG'
000040 : 2C 54 49 50 4C 4F 47 2C 54 49 50 4C 4F 47 2C 54 ',TIPLOG,TIPLOG,T'
000050 : 49 50 00 00 00 00 00 00 00 00 00 00 00 00 00 00 'IP..............'
000060 : 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 '................'
000070 : thru 0007EF same as last
0007F0 : 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 '................'
14:27:03 FCS-PUT 4 TIP$QUE Rec# 1 PIB-GOOD
---- [Data record] len[2048] ----
000000 : 00 00 00 01 00 00 00 00 00 00 00 00 41 4C 4C 49 '............ALLI'
000010 : 4E 53 4F 4E 09 50 22 3F 01 42 70 2F 00 00 00 00 'NSON.P"?.Bp/....'
000020 : 00 00 00 42 00 00 00 00 00 00 00 00 00 00 00 00 '...B............'
000030 : 41 52 43 51 55 45 20 20 00 00 00 02 00 00 00 41 'ARCQUE .......A'
000040 : 00 00 00 40 00 00 00 00 00 00 00 00 00 00 00 00 '...@............'
```

```
000050 : 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 '................'
000060 : thru 0007EF same as last
0007F0 : 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 '................'
---- [Data] len[60] ----
000000 : 00 3C 00 00 41 4C 4C 49 4E 53 4F 4E 53 2F 31 30 '.<..ALLINSONS/10'
000010 : 41 52 43 20 39 35 30 32 32 33 31 34 32 37 30 32 'ARC 950223142702'
000020 : 30 30 4E 00 54 49 50 4C 4F 47 2C 54 49 50 4C 4F '00N.TIPLOG,TIPLO'
000030 : 47 2C 54 49 50 4C 4F 47 2C 54 49 50 'G,TIPLOG,TIP'
14:27:03 FCS-TREN 2 ARCQUE COMMIT PIB-GOOD
14:27:03 FCS-TREN 2 ARCQUE COMMIT PIB-GOOD
14:27:03 QUE CLOSE 2 ARCQUE
14:27:03 FCS-GETUPW 4 TIP$QUE Rec# 1 PIB-GOOD
---- [Data record] len[2048] ----
000000 : 00 00 00 01 00 00 00 00 00 00 00 00 41 4C 4C 49 '............ALLI'
000010 : 4E 53 4F 4E 09 50 22 3F 01 42 70 2F 00 00 00 00 'NSON.P"?.Bp/....'
000020 : 00 00 00 42 00 00 00 00 00 00 00 00 00 00 00 00 '...B............'
000030 : 41 52 43 51 55 45 20 20 00 00 00 02 00 00 00 41 'ARCQUE .......A'
000040 : 00 00 00 40 00 00 00 00 00 00 00 00 00 00 00 00 '...@............'
000050 : 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 '................'
000060 : thru 0007EF same as last
0007F0 : 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 '................'
14:27:03 FCS-PUT 4 TIP$QUE Rec# 1 PIB-GOOD
---- [Data record] len[2048] ----
000000 : 00 00 00 01 00 00 00 00 00 00 00 00 41 4C 4C 49 '............ALLI'
000010 : 4E 53 4F 4E 09 50 22 3F 01 42 70 2F 00 00 00 00 'NSON.P"?.Bp/....'
000020 : 00 00 00 42 00 00 00 00 00 00 00 00 00 00 00 00 '...B............'
000030 : 41 52 43 51 55 45 20 20 00 00 00 02 00 00 00 41 'ARCQUE .......A'
000040 : 00 00 00 40 00 00 00 00 00 00 00 00 00 00 00 00 '...@............'
000050 : 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 '................'
000060 : thru 0007EF same as last
0007F0 : 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 '................'
14:27:03 FCS-CLOSE 2 TIP$QUE Rec# 1 PIB-GOOD
This is a test TIPLOG call, line number=00000001.
This is a test TIPLOG call, line number=00000002.
This is a test TIPLOG call, line number=00000003.
This is a test TIPLOG call, line number=00000004.
This is a test TIPLOG call, line number=00000005.
....+....1....+....2....+....3....+....4....+....5....+....6
1 : ' SNAP 00000005 '
61 : ' 63 Records currently in queue. '
121 : ' '
000000 : 00 3C 00 00 41 4C 4C 49 4E 53 4F 4E 53 2F 31 30 '.<..ALLINSONS/10'
000010 : 41 52 43 20 39 35 30 32 32 33 31 34 32 37 30 32 'ARC 950223142702'
000020 : 30 30 4E 00 54 49 50 4C 4F 47 2C 54 49 50 4C 4F '00N.TIPLOG,TIPLO'
000030 : 47 2C 54 49 50 4C 4F 47 2C 54 49 50 4C 4F 47 2C 'G,TIPLOG,TIPLOG,'
000040 : 54 49 50 4C 4F 47 2C 54 49 50 4C 4F 47 2C 54 49 'TIPLOG,TIPLOG,TI'
000050 : 50 4C 4F 47 2C 54 49 50 4C 4F 47 2C 54 49 50 4C 'PLOG,TIPLOG,TIPL'
000060 : 4F 47 2C 54 49 50 4C 4F 47 2C 54 49 50 4C 4F 47 'OG,TIPLOG,TIPLOG'
000070 : 2C 54 49 50 4C 4F 47 2C 54 49 50 4C 4F 47 2C 54 ',TIPLOG,TIPLOG,T'
000080 : 49 50 4C 4F 47 2C 54 49 'IPLOG,TI'
14:27:03 TIPRTN
```

Source Code of TSTLOG program used to make log files:

The following listing is of the test program TSTLOG that was used to create the above two logs files. By comparing the log files to the source code, you can relate the debugging information to the program that created it.

```
IDENTIFICATION DIVISION.
PROGRAM-ID.                                    TSTLOG.
AUTHOR.                                        INGLENET CORP.
DATE-WRITTEN.                                  February 1999.
ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
SOURCE-PC. UNIVAC-OS3.
OBJECT-PC. UNIVAC-OS3.
*************************************************************
*
```

```
* TIPLOG TEST PROGRAM:
*
* CDA PARAMETER USAGE

* P1 = OPEN - Create a log/TRID file internally
* P2 = DUMP - End program with a TIPDUMP Call
* P3 = SNAP - Test the TIPSNAP Call
* P4 = number of lines to write (default = 10)
*
************************************************************
/
DATA DIVISION.
WORKING-STORAGE SECTION.
01  FUNCTION-CODES.                          COPY TC-FCS.
01 QUEUEING                                  PICTURE X(80)
        VALUE "JUST WROTE SOME RECORDS TO THE QUEUE.".
01  LOG-MESSAGE.
    05  LOG-LITERAL-1                        PICTURE X(40)
        VALUE "This is a test TIPLOG call, line number=".
    05  LOG-LINE-NUMBER                      PICTURE 9(8).
    05  LOG-LITERAL-2 VALUE "."              PICTURE X(32).
/
LINKAGE SECTION.
01  PIB.                                     COPY TC-PIB.
/
01  DUMMY.
    05  DUMMY-WORK                           PICTURE X(8).
01 WORK-AREA.
    05  WRK-MESSAGE.
        10  WRK-LITERAL-1                    PICTURE X(40).
        10  WRK-COUNT                        PICTURE 9(8).
        10  WRK-LITERAL-2                    PICTURE X(32).
    05  QUEUE-PKT.
        10  QUEUE-NAME                       PICTURE X(8).
        10  QUEUE-STS                        PICTURE X.
    05  STOP-FLAG                            PICTURE X.
        88  START-TEST                           VALUE "1".
        88  STOP-TEST                            VALUE "0".
    05  SNAP-FLAG                            PICTURE X.
        88  SNAP-NO                              VALUE "0".
        88  SNAP-YES                             VALUE "1".
    05  DUMP-FLAG                            PICTURE X.
        88  DUMP-NO                              VALUE "0".
        88  DUMP-YES                             VALUE "1".
    05  OPEN-FLAG                            PICTURE X.
        88  OPEN-NO                              VALUE "0".
        88  OPEN-YES                             VALUE "1".
    05  LOG-COUNT COMP SYNC                  PICTURE 9(5).
    05  WORK-LEN COMP SYNC                   PICTURE 9(4).
    05  WORK-REM COMP SYNC                   PICTURE 9(4).
    05  MAX-RECORD COMP SYNC                 PICTURE 9(8).
    05  TTL-RECORD COMP SYNC                 PICTURE 9(8).
    05  REC-COUNT                            PICTURE ZZZZZZZZ9.
    05  FILLER COMP SYNC                     PICTURE 9(5).
    05  QR-RECORD.
        10  QR-LENGTH COMP-4 SYNC            PICTURE 9(4).
        10  FILLER                           PICTURE X(1).
        10  FILLER                           PICTURE X(1).
        10  QR-CLIENT.
            15  QR-CL-UID                    PICTURE X(8).
            15  QR-CL-TID                    PICTURE X(4).
            15  QR-CL-LOCAP                  PICTURE X(4).
            15  QR-CL-DATE                   PICTURE 9(6).
            15  QR-CL-TIME                   PICTURE 9(8).
            15  QR-CL-PRINT                  PICTURE X.
            15  FILLER                       PICTURE X.
        10  QR-DATA                          PICTURE X(100).
        10  QR-END                           PICTURE X.
/
01  CDA.                                     COPY TC-CDA.
    05  CDA-END                              PICTURE X.
```

```
/
PROCEDURE DIVISION USING                    PIB
                                            CDA
                                            DUMMY
                                            WORK-AREA.
INITALIZATION.
    MOVE 60                                 TO QR-LENGTH
    MOVE 3                                  TO MAX-RECORD
    MOVE 0                                  TO TTL-RECORD
    MOVE 0                                  TO PIB-MIRAM-REL-REC-NUM
    MOVE ALL "TIPLOG,"                      TO QR-DATA
    SET START-TEST                          TO TRUE
    MOVE "ARCQUE"                           TO QUEUE-NAME
SET OPEN-NO                                 TO TRUE
    IF CDA-PARAM (1) = "OPEN"
        SET OPEN-YES TO TRUE
        CALL "TIPLOG" USING                 FCS-OPEN
    END-IF
SET DUMP-NO TO TRUE
    IF CDA-PARAM (2) = "DUMP"
        SET DUMP-YES TO TRUE
    END-IF
SET SNAP-NO TO TRUE
    IF CDA-PARAM (3) = "SNAP"
        SET SNAP-YES TO TRUE
    END-IF
MOVE 10                                     TO LOG-COUNT
    IF CDA-PARAM (4) IS NUMERIC
        MOVE CDA-PARAM (4)                  TO LOG-COUNT
    END-IF
    MOVE "TSTLOG OPENING TIPQUE:ARCQUE"     TO CDA-TEXT
    CALL "ROLL" USING                       CDA-TEXT

    PERFORM OPEN-TIPQUEUE

    IF PIB-GOOD
        IF PIB-MIRAM-REL-REC-NUM > 0
            MOVE "xxxxxxxxx Records currently in queue."
                                            TO CDA-TEXT
            MOVE PIB-MIRAM-REL-REC-NUM      TO REC-COUNT
            MOVE REC-COUNT                  TO CDA-TEXT (1:9)
            CALL "ROLL" USING               CDA-TEXT
        END-IF
        PERFORM WRITE-DATA
        PERFORM ISSUE-TREN
    ELSE
        CALL "TIPFCER" USING                QUEUE-PKT
                                            CDA-TEXT
        CALL "ROLL" USING CDA-TEXT
    END-IF
    MOVE 0                                  TO PIB-MIRAM-REL-REC-NUM
    PERFORM CLOSE-TIPQUEUE
    MOVE LOG-MESSAGE                        TO WRK-MESSAGE
    MOVE 0                                  TO WRK-COUNT
    PERFORM UNTIL WRK-COUNT >= LOG-COUNT
        ADD 1                               TO WRK-COUNT
        CALL "TIPLOG" USING                 FCS-PUT
                                            WRK-MESSAGE
    END-PERFORM
    MOVE "$"                                TO CDA-END
    IF SNAP-YES
        CALL "TIPLOG" USING                 FCS-FLUSH
                                            CDA CDA-END
                                            QR-RECORD QR-END
    END-IF
    IF DUMP-YES
        CALL "TIPDUMP"
    END-IF
    IF OPEN-YES
        CALL "TIPLOG" USING                 FCS-CLOSE
    END-IF
```

```
    CALL "TIPRTN".
/
WRITE-DATA.
    PERFORM WRITE-RECORDS
    IF PIB-GOOD
        SET PIB-COMMIT                  TO TRUE
        PERFORM ISSUE-TREN
    ELSE
        SET PIB-ROLLBACK                TO TRUE
        PERFORM ISSUE-TREN
        SET STOP-TEST TO TRUE
        CALL "TIPFCER" USING            QUEUE-PKT
                                        CDA-TEXT
        CALL "ROLL" USING               CDA-TEXT
    END-IF.
WRITE-RECORDS.
    MOVE PIB-UID                        TO QR-CL-UID
    MOVE PIB-TID                        TO QR-CL-TID
    MOVE PIB-LOCAP                      TO QR-CL-LOCAP
    ACCEPT QR-CL-DATE FROM DATE
    ACCEPT QR-CL-TIME FROM TIME
    MOVE "N"                            TO QR-CL-PRINT
    ADD 1                               TO TTL-RECORD
    PERFORM PUT-TIPQUEUE.
OPEN-TIPQUEUE.
    CALL "TIPQUEUE" USING               FCS-OPEN
                                        QUEUE-PKT.
PUT-TIPQUEUE.
    CALL "TIPQUEUE" USING               FCS-PUT
                                        QUEUE-PKT
                                        QR-RECORD.
CLOSE-TIPQUEUE.
    CALL "TIPQUEUE" USING               FCS-CLOSE
                                        QUEUE-PKT.
ISSUE-TREN.
    CALL "TIPFCS" USING                 FCS-TREN
                                        QUEUE-PKT.
```

Source-level Debugging

Source level debugging is a powerful feature of both the Micro Focus and COBOL-IT compilers. Before attempting to use this feature, you should read the appropriate sections of the compiler documentation regarding source level debugging.

- The Micro Focus source level debugger is called "**animator**".
- The COBOL-IT debugger is part of the COBOL-IT development system which is based on Eclipse

Each compiler has a special procedure that is required to invoke its source level debugging features. These procedures involve special compiling options that must be selected when the program is compiled.

Note: Compiling a program for source level debugging creates a binary image that contains considerably more information. These programs require more overhead to execute. You should only compile your programs with these options when you intend to use the source level debugging features. Once the programs have been thoroughly tested and debugged and are ready for production, they should always be compiled without the debugging options. This will create the most efficient operating environment for your on-line system.

Compiling the program with special options is only part of the story. Once this has been done, the on-line program has to be invoked in a special way to have the appropriate debugger operate correctly.

- For standard (non-TIP) programs, these procedures are outlined in the appropriate COBOL compiler documentation.
- For TIP transaction programs, you must follow the steps outlined below:

Using Two Terminals for Debugging

You must follow a special procedure before using the COBOL compiler source level debuggers.

Note: The problem is that both the debugger and TIP want to have control of the terminal display and keyboard.

To overcome this problem, have the source level debugger interact with the primary (or first) terminal ("**Terminal A**"), and have TIP use a second terminal ("**Terminal B**") to interact with the transaction program.

Step 0: Compile

Compile your transaction program, *trid*, with debugging options:

Micro Focus
Edit make.mf for debugging.

Comment out the non-debugging MFOPT line, and restore (uncomment) the appropriate MFOPT debugging line.

Online:
```
MFOPT = -gUa -I TIPFCS -I TIPMSGO -I TIPMSG
-I TIPH2P -I TIPSUB
```

Bbatch:
```
MFOPT = -gUa -I BATFCS -I INITFTAB
```

Now compile:
```
make -f make.mf trid
```

MBP
Specify the debugging option on the command line.
```
make -f make.mbp trid VISOPT=%debug+
```

After compiling, verify that the support files for debugging have been created:

Compiler	Extensions
Micro Focus	.int, .idy

Step 1: UNIX Sessions

Establish two UNIX sessions on the same UNIX host. (You can use TIP/fe as your terminal emulator.)

Step 2: Get Termids

From **Terminal A**, execute the following command to determine its termid:

```
who am i
```

Now, switch to **Terminal B** and get its termid:

```
who am i
```

In this example, the termid for the first session is “**pts/3**”; and the second is "**pts/6**".

By the way, you might want to find the termid for your sessions without having to switch sessions. For example, if your user id is “ianm”, you could get a list of all your termids as follows:

```
who | grep ianm
```

Step 3: Sleep

On **Terminal B**, execute the Unix **sleep** command.

```
sleep 20000
```

This causes terminal B to sleep for 20,000 seconds (about 5.5 hours).

Later you will switch the MCS interaction to this session. This is to avoid the problems that occur when two programs try to get input from the same terminal.

Step 4: Enable Debugging (from UNIX/LINUX)

You can enable debugging at this point from UNIX/LINUX, or later from TIP.

From Terminal A:

Micro Focus

To enable animation for all Micro Focus COBOL programs compiled with debugging options, set the COBSW environment variable to +A from the UNIX command line:

```
export COBSW=+A
```

Step 5: Change to Debugging Directory

On Terminal A:

Micro Focus

Either:

Go to the directory containing your COBOL source code and debugger support files, or

set the COBPATH environment variable to include the directory containing your COBOL source code. For example:

```
export COBPATH=/prod/src/;test/src;/my/src
```

Step 6: Start TIP

On **Terminal A** (pts/3), enter the TIP command line processor (tipix) using the **-t** option and specifying the termid of **Terminal B** (pts/6):

```
tipix -t pts/6
```

TIP will display its screens on **Terminal B**.

Step 7: Enable Debugging (from TIP)

If you have not already enabled debugging from UNIX, you must enable it now. You can use **smprog** or environment variables:

With smprog:

From **Terminal B**, use the **smprog** utility to set the debug attribute for *this* transaction program. This defines which debugger, if any, TIP will use when running this transaction program.

Debugger	Debug Attribute
Micro Focus Animator	A
COBOL-IT	C
OpenCOBOL	O

This is very useful when you are debugging a program, which is invoked by another program (by means of TIPSUB, TIPXCTL, etc.).

If you do not want to set the debug attribute for all users of a program, you should create a second program definition. For example, if you want to debug program PAY001, add a program definition for PAY001T with the appropriate debug attribute for your compiler. Then create a security entry in your user id group that references the PAY001T definition. That is MYUID/PAY001 references PAY001T, and TIPY/PAY001 references PAY001.

With Environment Variables from TIP Command Line:

From Terminal B:

Micro Focus

To enable animation for all Micro Focus COBOL programs compiled with debugging options, set the COBSW environment variable to +A from the TIP command line:

```
setenv COBSW=+A
```

Step 8: Run

From **Terminal B**, run your transaction:

Micro Focus:

If you used smprog to set the debug attribute, just enter the transaction name at the TIP command line:

```
trid
```

If you set the COBSW environment variable to +A, just enter the transaction name at the TIP command line:

```
trid
```

If the .int file cannot be found, you get "Load error 173".

If the .idy file cannot be found, you get a message to this effect.

The debugger (and your source code) should appear on Terminal A, your transaction executes on Terminal B.

Step 9: Cancel Sleep

When you are finished debugging, switch to **terminal B**.

Execute the FIN command from the TIP command line. TIP displays a message that says how to cancel the **sleep** program. The key sequence to press depends on the interrupt key for your UNIX session.

For example if **stty -a** indicates **intr = ^c** then the message displayed by the TIP shell upon exit will be:

Press Control-C to stop 'sleep'

Follow the instructions.

Step 10: Unset

If you set COBSW, you should unset it to deactivate animation.

```
unset COBSW
```

Using Micro Focus cobanimsrv

If you have compiled a transaction program for debugging then using ‘smprog’ define the transaction definition in TIP/ix to be invoked with Animator and place some unique value in the ‘Debugger Tag’ field.

Then on a Unix/Linux terminal session:

```
setenv COBANIMSRV myval
cobanimsrv
```

Where ‘myval’ is the same value you defined with smprog.

When Tip/ix execute the transaction it will also define the environment variable COBANIMSRV with the defined value to connect the running program to the MF Animator.

Reference Tables

The following Hexadecimal – Decimal Conversion table may be used to convert decimal numbers (base 10) to and from hexadecimal numbers (base 16).

Hexadecimal - Decimal Conversion

6		5		4		3		2		1	
Hex	Dec	Hex	Dec	Hex	Dec	Hex	Dec	Hex	Dec	Hex	Dec
0		0		0		0		0		0	
1	1 048 576		65 536	1	4 096	1	256	1	16	1	1
2	2 097 152	2	131 072	2	8 192	2	512	2	32	2	2
3	3 145 728	3	196 608	3	12 288	3	768	3	48	3	3
4	4 194 304	4	262 144	4	16 384	4	1 024	4	64	4	4
5	5 242 880	5	327 680	5	20 480	5	1 280	5	80	5	5
6	6 291 456	6	393 216	6	24 576	6	1 536	6	96	6	6
7	7 340 032	7	458 752	7	28 672	7	1 792	7	112	7	7
8	8 388 608	8	524 288	8	32 768	8	2 048	8	128	8	8
9	9 437 184	9	589 824	9	36 864	9	2 304	9	144	9	9
A	10 485 760	A	655 360	A	40 960	A	2 560	A	160	A	10
B	11 534 336	B	720 896	B	45 056	B	2 816	B	176	B	11
C	12 582 912	C	786 432	C	49 152	C	3 072	C	192	C	12
D	13 631 488	D	851 968	D	53 248	D	3 328	D	208	D	13
E	14 680	E	917 504	E	57 344	E	3 584	E	224	E	14

6		5		4		3		2		1	
	064										
F	15 728 640	F	983 040	F	61 440	F	3 480	F	240	F	15

Powers of 2

n	2^n	n	2^n	n	2^n
0	1	11	2 048	22	4 194 304
1	2	12	4 096	23	8 388 608
2	4	13	8 192	24	16 777 216
3	8	14	16 384	25	33 554 432
4	16	15	32 768	26	67 108 864
5	32	16	65 536	27	134 217 728
6	64	17	131 072	28	268 435 456
7	128	18	262 144	29	536 870 912
8	256	19	524 288	30	1 073 741 824
9	512	20	1 048 576	31	2 147 483 648
10	1024	21	2 097 152	32	4 294 967 296

Powers of 16

n	16^n	n	16^n
0	1	8	4 294 967 296
1	16	9	68 719 476 736
2	256	10	1 099 511 627 776
3	4 096	11	17 592 186 044 416

n	16^n	n	16^n
4	65 536	12	281 474 976 710 656
5	1 048 576	13	4 503 599 627 370 496
6	16 777 216	14	72 057 594 037 927 936
7	268 435 456	15	1 152 921 504 606 846 976

ASCII Code Chart.

The following table is the character code table for the American Standard Code for Information Interchange (ASCII).

ASCII characters from X'80' to X'FF' are generally undefined, but many vendors use this range to provide international characters and other implementation specific items.

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0x	nul	soh	stx	etx	eot	enq	ack	bel	bs	ht	lf	vt	ff	cr	so	si
1x	dle	dc1	dc2	dc3	dc4	nak	syn	etb	can	em	sub	esc	fs	gs	rs	us
2x	sp	!	"	#	$	%	&	'	(	)	*	+	,	_	.	/
3x	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
4x	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
5x	P	Q	R	S	T	U	V	W	X	Y	Z	[	\	]	^	_
6x	`	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
7x	p	q	r	s	t	u	v	w	x	y	z	{	|	}	~	del

Standard Windows Character Set

The following table is the standard Windows character set:

ISO 8859-1 (Latin-1 or ANSI) Code Chart (Windows variant)

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0x	nul	soh	stx	etx	eot	enq	ack	bel	bs	ht	lf	vt	ff	cr	so	si
1x	dle	dc1	dc2	dc3	dc4	nak	syn	etb	can	em	sub	esc	fs	gs	rs	us
2x	sp	!	"	#	$	%	&	'	(	)	*	+	,	-	.	/
3x	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
4x	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
5x	P	Q	R	S	T	U	V	W	X	Y	Z	[	\	]	^	_
6x	`	a	b	c	d	e	f	g	h	I	j	k	l	m	n	o
7x	p	q	r	s	t	u	v	w	x	y	z	{	|	}	~	del
8x	€	•	,	*f*	„	...	†	‡	ˆ	‰	Š	‹	Œ	•	Ž	•
9x	•	‘	’	"	"	•	–	—	˜	™	š	›	œ	•	ž	Ÿ
Ax	hsp	¡	¢	£	¤	¥	¦	§	¨	©	ª	«	¬	-	®	¯
Bx	°	±	²	³	´	µ	¶	·	¸	¹	º	»	¼	½	¾	¿
Cx	À	Á	Â	Ã	Ä	Å	Æ	Ç	È	É	Ê	Ë	Ì	Í	Î	Ï
Dx	Ð	Ñ	Ò	Ó	Ô	Õ	Ö	×	Ø	Ù	Ú	Û	Ü	Ý	Þ	ß
Ex	à	á	â	ã	ä	å	æ	ç	è	é	ê	ë	ì	í	î	ï
Fx	ð	ñ	ò	ó	ô	õ	ö	÷	ø	ù	ú	û	ü	ý	þ	ÿ

Note: The glyphs assigned to the code points from 0x80 through 0x9F are not part of the official ANSI character set. These glyphs are part of the MS Windows version of the ANSI code set.

National Replacement Character (NRC) Mappings

The following chart shows the National Replacement Character (NRC) Mappings:

Language	Code	National Replacement Character (NRC) Mappings												
EBCDIC		#	$	@	[	\	]	^	_	`	{	|	}	~
Decimal		123	91	124	74	224	79	95	109	121	192	106	208	161
Hex		7B	5B	7C	4A	E0	4F	5F	6D	79	C0	6A	D0	A1
ASCII		#	$	@	[	\	]	^	_	`	{	|	}	~
Decimal	ASC	35	36	64	91	92	93	94	95	96	123	124	125	126
Hex		23	24	40	5B	5C	5D	5E	5F	60	7B	7C	7D	7E
British		£	$	@	[	\	]	^	_	`	{	|	}	~
United Kingdom	BRI	163	36	64	91	92	93	94	95	96	123	124	125	126
		A3	24	40	5B	5C	5D	5E	5F	60	7B	7C	7D	7E

Language	Code	National Replacement Character (NRC) Mappings												
Canadian French	CFR	# 35 23	$ 36 24	à 224 E0	â 226 E2	ç 231 E7	ê 234 EA	î 238 EE	_ 95 5F	ô 244 F4	é 233 E9	ù 249 F9	è 232 E8	û 251 FB
Norwegian Danish	NOR	# 35 23	$ 36 24	Ä 196 C4	Æ 198 C6	Ø 216 D8	Å 197 C5	Ü 220 DC	_ 95 5F	ä 228 E4	æ 230 E6	ø 248 F8	å 229 E5	ü 252 FC
Dutch	DUT	£ 163 A3	$ 36 24	¾ 190 BE	[91 5B	½ 189 BD	| 124 7C	^ 94 5E	_ 95 5F	` 96 60	¨ 168 A8	ƒ 131 83	¼ 188 BC	´ 180 B4
Finnish	FIN	# 35 23	$ 36 24	@ 64 40	Ä 196 C4	Ö 214 D6	Å 197 C5	Ü 220 DC	_ 95 5F	é 233 E9	ä 228 E4	ö 246 F6	å 229 E5	ü 252 FC
French Belgian	FRE	£ 163 A3	$ 36 24	à 224 E0	º 186 BA	ç 231 E7	§ 167 A7	^ 94 5E	_ 95 5F	` 96 60	é 233 E9	ù 249 F9	è 232 E8	¨ 168 A8
German	GER	# 35 23	$ 36 24	§ 167 A7	Ä 196 C4	Ö 214 D6	Ü 220 DC	^ 94 5E	_ 95 5F	` 96 60	ä 228 E4	ö 246 F6	ü 252 FC	ß 223 DF
Italian	ITA	£ 163 A3	$ 36 24	§ 167 A7	º 186 BA	ç 231 E7	é 233 E9	^ 94 5E	_ 95 5F	ù 249 F9	à 224 E0	ò 242 F2	è 232 E8	ì 236 EC
Portuguese	POR	# 35 23	$ 36 24	@ 64 40	Ã 195 C3	Ç 199 C7	Õ 213 D5	^ 94 5E	_ 95 5F	` 96 60	ã 227 E3	ç 231 E7	õ 245 F5	~ 126 7E
Spanish	SPA	£ 163 A3	$ 36 24	§ 167 A7	¡ 161 A1	Ñ 209 D1	¿ 191 BF	^ 94 5E	_ 95 5F	` 96 60	º 186 BA	ñ 241 F1	ç 231 E7	~ 126 7E
Swedish	SWE	# 35 23	¤ 164 A4	@ 64 40	Ä 196 C4	Ö 214 D6	Å 197 C5	Ü 220 DC	_ 95 5F	é 233 E9	ä 228 E4	ö 246 F6	å 229 E5	ü 252 FC
Swiss French German	SWI	ù 249 F9	$ 36 24	à 224 E0	é 233 E9	ç 231 E7	ê 234 EA	î 238 EE	è 232 E8	ô 244 F4	ä 228 E4	ö 246 F6	ü 252 FC	û 251 FB

EBCDIC Code Chart

The following table is the character code table for the Extended Binary-Coded Decimal Interchange Code (EBCDIC).

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0x	nul	soh	stx	etx		ht		del				vt	ff	cr	so	si
1x	dle	dc1	dc2	dc3			bs		can	em			fs	gs	rs	us
2x	ds	sos	fs			lf	etb	esc						enq	ack	bel
3x			syn					eot					dc4	nak		sub
4x	sp										[	.	<	(	+	]
5x	&										!	$	*	)	;	^
6x	-	/									|	,	%	_	>	?
7x										`	:	#	@	'	=	"
8x		a	b	c	d	e	f	g	h	i						
9x		j	k	l	m	n	o	p	q	r						
Ax		~	s	t	u	v	w	x	y	z						
Bx																
Cx	{	A	B	C	D	E	F	G	H	I						
Dx	}	J	K	L	M	N	O	P	Q	R						
Ex	\		S	T	U	V	W	X	Y	Z						
Fx	0	1	2	3	4	5	6	7	8	9						

EBCDIC NRC Chart

The following table is the EBCDIC NRC Chart:

EBCDIC		ANSI Code based on NLS option selected –N xx where xx=						
Char	Hex	sp (Spain)	dn (Denmark /Norway)	fr (France)	ge (Germany)	sw (Sweden /Finland)	uk (United Kingdom)	it (Italy)
#	7B	**£** A3	# 23	**£** A3	# 23	# 23	**£** A3	**£** A3

EBCDIC		ANSI Code based on NLS option selected –N xx where xx=						
$	5B	**$** 24	**$** 24	**$** 24	**$** 24	**¤** A4	**$** 24	**$** 24
@	7C	**§** A7	@ 40	**à** E0	**§** A7	**É** C9	@ 40	**§** A7
[	4A	**¡** A1	**Æ** C6	**º** BA	**Ä** C4	**Ä** C4	**[** 5B	**°** B0
****	E0	**Ñ** D1	**Ø** D8	**ç** E7	**Ö** F6	**Ö** F6	**** 5C	# 23
]	4F	**¿** BF	**Å** C5	**§** A7	**Ü** DC	**Å** C5	**]** 5D	**é** E9
^	5F	**^** 5E	**^** 5E	**^** 5E	**^** 5E	**Ü** DC	**^** 5E	**^** 5E
`	79	**`** 60	**`** 60	**`** 60	**`** 60	**é** E9	**`** 60	**ù** F9
{	C0	**°** B0	**æ** E6	**É** E9	**Ä** E4	**Ä** E4	**{** 7B	**à** E0
|	6A	**ñ** F1	**ø** F8	**Ù** F9	**Ö** F6	**Ö** F6	**|** 7C	**ò** F2
}	D0	**ç** E7	**å** E5	**È** E8	**Ü** FC	**Å** E5	**}** 7D	**è** E8
~	A1	**~** 7E	**¯** AD	**¨** A8	**ß** DF	**Ü** FC	**~** 7E	**Ì** EC

Error Codes

This chapter contains some hints on how to track down errors that occur when running batch shell scripts on Unix.

Unix Shell Error

When the Unix shell reports an error the message text is prefixed with "**UX:sh**"

For example:

```
myjob.sh: 12:31:53 myjob: executing myprog
UX:sh (myjob.sh): ERROR: myprog: Not found
```

Either the file, myfile, was not found in the PATH (obvious), or the file was found but the user did not have execute permission for the file (not so obvious).

Micro Focus Cobol

Micro Focus COBOL file handler errors appear in the form **n/nnn**.

You can look up these messages in the *Micro Focus COBOL System Reference* in the appendix titled "File Handler Utility Error Messages"

For example:

```
myjob.sh: 12:47:44 myjob: executing myprog
 I/O error : file 'MYFILE'
 error code: 9/065 (ANS74), pc=0, call=1, seg=0
 65 File locked
```

Look under the subsection "When status1 is Set to **9**", then look for a Status2 value of "**065**".

Note: This particular message can be eliminated by adding the option CALLFH\"ARMFH\" to the Micro Focus COBOL compiler options in your makefile.

Return Status from Unix System Calls

Sometimes a program reports an error number (status) that it receives from a Unix system call (that your application has called).

These can be found in the module **errno.h** which is usually in **/usr/include/sys/errno.h**. If it is not in this directory you can find it by going to the root directory and entering this command:

```
find . -name errno.h -print
```

The file errno.h associates error numbers with symbolic names. There is usually a text description associated with the symbolic name. On some systems all three appear together on a single line. On others, the associations are in two separate parts of errno.h.

D-ISAM Error Codes

The following table is from the D-ISAM file system documentation, a product of Byte Designs Inc. that is included in TIP.

Error	Description
1- 99	Errors less than 100 generally emanate from the Unix system, and can be found in errno.h. See previous section.
100	An attempt was made to (re)write a duplicate where duplicates are prohibited, or an attempt was made to REWRITE(F) where the primary key permitted duplicates.
101	The fd parameter does not reference an opened file.
102	One of the arguments has a value with no defined meaning.

Error	Description
103	The values of key are not valid.
104	All ISAM file descriptors are used, you cannot open any more files
105	The ISAM file is corrupted, it must be repaired with DCHECK.
106	Exclusive access to the file is not possible.
107	Another process has a read-only lock on the requested record.
108	The value of key has already been established as a key.
109	The requested function may not be performed on the primary key, as requested.
110	The beginning or end of the file has already been reached.
111	No record was found to match your request.
112	There is no "current" record set at this time.
113	The file has been exclusively locked by another process, or if trying to establish an exclusive lock, another process is using the file.
114	The name given for the file is too long or contains unacceptable characters.
115	The lock file cannot be created. Presently not used by D-ISAM.
116	malloc() cannot allocate the request. Usually means out of memory, but possibly the allocation list is corrupted.

The error message text depends on the application program (such as **armdata** or **armsort**), but generally is of the form: "Error ### accessing file" (### represents an error number greater than 99.

Information Management System(IMS)

This chapter will explain how TIP emulates an IMS program and it will also state the known differences between IMS and TIP programs. Please keep your Unisys Information Management System (IMS) Programming

Guide ref # UP-9207 manual on hand as a reference for any IMS problems.

TIP and IMS Interaction

In some situations it may be necessary to have a native mode TIP program call an IMS program that is running under emulation or have an IMS emulated program call a TIP native mode program. IMS programs run under control of the TIP IMS Emulator. The IMS Emulator is designed to "emulate" the IMS environment; it does not give IMS programs access to TIP facilities. IMS and TIP programs may transfer control to each other; however, this interaction must take place according to very specific rules. In any case, the contents of the CDA are copied to and from the programs involved.

An IMS program may "succeed" (an IMS term) to a TIP program by utilizing one of the following methods:

To accomplish external succession, the IMS program must:

```
MOVE "??????"   TO SUCCESSOR-ID
MOVE "E"        TO TERMINATION-INDICATOR
```

When the terminal user responds to the screen information that is (usually) output when the IMS program terminates, the specified TIP transaction identified in the SUCCESSOR-ID field is called. To accomplish delayed internal succession, the IMS program must:

```
MOVE "??????"   TO SUCCESSOR-ID
 MOVE "D"       TO TERMINATION-INDICATOR
 MOVE ZERO      TO TEXT-LENGTH OF OMA
```

Note: The latter point is crucial - the TIP program does **not** have an Input Message Area (IMA). Consequently, the IMS program may **not** leave text in the Output Message Area (OMA) to be carried forward to the next program's Input Message Area (IMA).

The IMS PIB field SUCCESSOR-ID is defined as a 6-byte field; the choice of TIP transaction names is, therefore, limited to six characters when a TIP program is called from an IMS program. To get around this restriction define the TIP transaction twice: once with a six-character name for succession purposes and a second time with whatever name the transaction may need for other types of invocation.

A TIP native mode program may call an IMS program by using the "TIPXCTL" or "TIPDXC" subroutines. The TIP program must move the defined name of the IMS program to "PIB-TRID" and then issue the CALL to "TIPXCTL" (or "TIPDXC").

Note: You must define IMS programs, whether they are **actions** or **transactions**. Define an action with the executable module name as the TIP transaction name.

It is usually inappropriate to invoke an IMS transaction via TIPXCTL if the transaction is expecting data in its Input Message Area (IMA) - since control came directly from a TIP program, there will be nothing in the IMA - this will likely cause the IMS program to be fatally confused.

Output for Input Queuing, from IMS Programs

Output for Input queuing is usually invoked with a CALL SEND statement that has the AUX-ID field of the OMA set to an "I".

You use the SMTERM utility to define terminal names, so that a CALL SEND statement can attempt to queue the message to the terminal name in the destination field of the OMA.

When printing from IMS applications, your application must set AUX-ID to “I”. In addition:

- To start the next transaction as a TIP background program, have your application set the OMA AUX-NO to "F".
- To output to a named terminal, specify PARAM IMSFORKW=NO in the "tipix.conf" file.
- To output to a new window, you must be running TIP/fe in smart mode. In addition, you must either:
  8. ¯specify PARAM IMSFORKW=YES in the "tipix.conf" file, or
  9. ¯have your application set the OMA AUX-NO to "W".

Effect	AUX-NO	AUX-ID	TIP/fe smart mode	IMSFORKW
Start in background	F	I		
Queue to named terminal		I		NO
Open new window		I	Yes	YES
	W	I	Yes	

Error Conditions:

- If starting up a new TIP/fe window fails, your application gets IMS STATUS = 4 and Detailed status = 8.

- If redirecting to a specific terminal fails because the terminal is not available, your application gets IMS STATUS = 6 and Detailed status = 4.

IMS Status Codes

When TIP is emulating IMS, if an error occurs and the error code from TIP does not map into a documented (or known) pair of values for the IMS STATUS-CODE and DETAILED-STATUS-CODE then the following occurs:

- The IMS STATUS-CODE (in the PIB) is set to 4
- the IMS DETAILED-STATUS-CODE (also in the PIB) is set to the TIP error code (PIB-STATUS value).

The TIP PIB-STATUS code is always a displayable ASCII character. Displayable ASCII characters have decimal values greater than 31 so there is no conflict (overlap) with the range of values documented for the IMS DETAILED-STATUS-CODE. For example, a value of decimal 90 (hex 5A) is an ASCII 'Z' which means PIB-FULL. For a list of values for PIB-STATUS, see *PIB - Process Information Block* in the *TIP Programming Reference*.

Known Differences between IMS and TIP

In this section we will make mention to all known differences between IMS and TIP programs.

Call Send:

After the first Call Send the OMA-AUX-FUNCTION field is cleared of the "I" function. This results in the second Call Send not being set to Output for Input queued. On TIP/30 this OMA field was not effected by the call send.

Index

'$PRIMARY
TIPPEER' 53
* field characteristic' 119
*BYP
reserved terminal name 41
*MST
reserved terminal name 41
.B field characteristic 119
.P field characteristic 119
.U field characteristic 119
ACCEPT
COBOL verb' 13
access
existing dynamic file' 215
Accessing TIP/ix Journal Files 254
'account code
PIB 13
accumulating print lines' 238
accuracy
of time' 13
Activating the log file 284
Active File Table
see AFT' 159
add
direct
add record' 202
indexed record 174
line to edit buffer' 223
addresses of storage area' 7
AFT
Active File Table' 159
and FCS 203, 210, 212, 217, 224, 233
after images 260
'after images
purpose' 159
'after images' 168, 254
ASCII code chart 303
ASCII Code Chart. 303
assign
dynamic
assign file' 216
assigning
format names' 88
asynchronous
process creation 41
audit
and journal file 254
automatic passing of parameters' 7
AUX0
and TIPPRINT' 238
AUX1
and TIPPRINT' 238
auxiliary device' 238
BACK$nnn
PIB 13
with TIPFORKW' 45
BACK$nnn' 43
background process
BACK$nnn' 13
background program
start in new window' 45

start with TIPFORK' 43
BAT
BATACTIV entry point..... 263
BATCOMIT entry point..... 264
BATFCS entry point 263
BATPIB entry point 263
BATROLBK entry point..... 264
batch
interface 262
Batch Commit and Rollback..... 264
batch interface
rollback
commit..... 264
batch journal file
access..... 260
close..... 260
open 260
read..... 260
Batch Journal File Access..... 260
BATFCS
commit and rollback 264
BATPEER - Peer-to-Peer from Batch 30
BATQUEUE - Queuing from Batch... 30
before images..... 260
purpose'..... 159
before images' 168, 254
BIT 34
bits
convert bits to bytes' 34
'BOF
bottom of form'..... 238
'BREAK
check for operator break' 134
BREAK - Check For Operator Break 134
break message' 134
BU 243
buffer
for TIPPRINT' 240
BUFFER'..... 241
buffer size
minimum
max' 243
'buffering
TIPPRINT'..... 249
bytes
convert bytes to bits' 31
CALL TIPFCER - Interpret FCS Error
..... 171
Call TIPFCS - Common Parameters. 164
calling
TIP 73
TIPFCS'..... 164
'calling
TIPQUEUE'..... 64
Calling TIP/ix Utilities..... 73
cancel
direct
cancel update' 207
indexed
cancel update 191
'carriage control
copy book'..... 249
CDA
introduction and example'..... 22
passed to transaction'..... 7
program to program data transfer' .. 8

'CDA
CDA= and PIB 13
CDA - Continuity Data Area 22
change elective groups
with TIPGRPST' 48
'client 68
climbing the stack' 8
close
direct
close file' 203
dynamic
close file' 217
edit buffer' 224
indexed file 175
library
close element (no update)' ... 234
library element' 233
sequential
close file' 210
TIPPRINT interface' 240
'close
TIPPEER conversation' 57
TIPQUEUE' 65
Close the Conversation CLOSE 57
Close the Log File 286
Closing a Queue - FCS-CLOSE 65
CMTCHAR=' 124
COBOL
ACCEPT' 13
MBP Visual COBOL 85 278
Micro Focus 278
COBOL Makefiles 280
COBPATH
environment variable 296
coding suggestion' 8
combining transmissions' 107
common carrier lines' 84
common storage' 25
communications codes' 84
construction of prompt' 137
Contents i
Context Sensitive Help 123
context sensitive help' 123
continuation prompts' 133
Continuity Data Area
see CDA' 22
see CDA' 7
control codes
DCIO' 83
convert
bits to bytes' 34
bytes to bits' 31
hex to decimal 301
copy book
TC 13, 31, 34, 143, 165, 166, 168, 232, 243, 249, 254
'copy book
file 266
TC 22, 24, 37, 47, 88, 146, 249
create
asynchronous process 41
dynamic
create files' 217
creating
screen formats' 84
CURSOR 105, 123

Cursor Positioning 123
cursor positioning' 123
D 308
D value
and MCS 111
'D630
printer option
Diablo' 238
DAM
Direct Access Method' 202
data
area layout' 84
transfer from program to program' . 8
validation' 105
data fields
uppercase' 84
date
PIB 13
TIPDATE subroutine' 34
DATE 34
DCIO
DCIO 143, 147, 149, 150
direct communications input output' 83
debugging
source level 296
'debugging
on 283
statements' 285
Debugging on-line programs 283
default
data' 111
terminal destination' 88
deferral
of transaction end' 25
deferred error text
defining' 107
defining
deferred error text' 107
help text' 124
screen formats' 84
delay program execution' 75
delayed
internal succession' 310
transfer control' 35
delete
direct
delete record' 204
edit buffers
delete line' 225
indexed record 175
logical record' 172
permanent' 172
physical record' 172
delimiters
parameter' 135
delivering
error message text' 105
'developing
client 68
Developing Client-Server Applications 68
DI 249
DICE codes
and T 149
DICE codes' 135

direct access
and TIPFCS' 202
direct communications I 143
Direct Communications I/O 143
direct files
add record' 202
and TIPFCS' 202
cancel update' 207
close file' 203
delete record' 204
flush file' 205
open file' 208
read record' 205
read with lock' 206
update record' 209
D-ISAM Error Codes 308
display
execution stack' 8
menu bar' 104
TIPASK subroutine' 92
TIPASKYN subroutine' 94
title' 119
'display
output a line
roll screen' 140
display type
fields' 88
dummy
linkage items' 7
dump
force program dump' 35
'dump
snap dump memory' 70
Dump Memory to the Log File 287
dynamic files
access existing file' 215
assign file' 216
close file' 217
create file' 217
definition' 159
introduction' 214
open file' 220
read records' 218
scratch file' 222
write record(s)' 221
dynamic linking 278
dynamic re 119
E character
error fields' 84
EBCDIC Code Chart 306
edit buffers
closing' 224
delete a line' 225
flush buffer' 226
introduction' 223
open buffer' 227
read line' 226
replace a line' 229
scratch buffer' 230
elective groups
changing with TIPGRPST' 48
retrieval of' 47
Embedded Debugging Statements 285
end
end sequential
set random 176

online program with TIPRTN' .. 69
EOJ command
and PIB 13
'EPSON
printer option'..... 238
erase
screen' 96
'error codes
D 308
Error Codes 307
'error conditions
see status' 35
error fields
in screens'..... 84
error message
delivering' 105
example
FCS 215
logical file name packet'..... 165
of FCS 227
of program stack' 8
program and screen interaction' ... 84
race conditions'..... 161
record locking' 161
using TIPFCER' 171
execution stack
program'..... 8
'execution stack
display' 8
PIB..... 13
explicit
transaction end' 25
external succession'..... 310
failed LOGON attempts
and journal file..... 254
FCC
m and n characters' 119
modifications' 119
sequences'..... 135
FCC Modifications..... 119
FCS 13, 161, 168, 170, 174, 175, 176, 177, 178, 179, 181, 183, 184, 185, 186, 190, 191, 192, 193, 195, 196, 197, 198, 200, 201, 202, 203, 204, 205, 206, 207, 208, 209, 210, 211, 212, 213, 215, 216, 217, 218, 220, 221, 222, 223, 224, 225, 226, 227, 229, 230, 233, 234, 235, 236, 240, 241, 249
and indexed files..... 173
interface packets' 165
interpret errors' 171
overview' 159
summary of functions' 159
'FCS..... 64, 65, 66, 67, 167, 169, 243
FCS Batch Interface..... 262
FCS Interface Packets..... 165
FCS Overview..... 159
FCS_LOCK
indexed
lock record..... 188
FCS-ACCESS - Dynamic Access File 215
FCS-ADD - Direct Add Record..... 202
FCS-ADD - Edit Add/Insert Line.... 223
FCS-ADD - Indexed Add Record.... 174
FCS-ASSIGN - Dynamic Assign File 216

FCS-CLOSE - Close PCXFER Interface 276
FCS-CLOSE - Close TIPPRINT Interface 240
FCS-CLOSE - Direct Close File..... 203
FCS-CLOSE - Dynamic Close File . 217
FCS-CLOSE - Edit Close Buffer..... 224
FCS-CLOSE - Indexed Close File... 175
FCS-CLOSE - Library Close Element 233
FCS-CLOSE - Sequential Close File210
FCS-CREATE - Dynamic Create File 217
FCS-DELETE - Direct Delete Record 204
FCS-DELETE - Edit Delete Line 225
FCS-DELETE - Indexed Delete Record 175
FCS-ESETL - Indexed End Sequential Mode 176
FCS-FLUSH - Direct Flush File..... 205
FCS-FLUSH - Edit Flush Buffer..... 226
FCS-FLUSH - Flush PCXFER Buffer 275
FCS-FLUSH - Flush TIPPRINT Buffer 241
FCS-FLUSH - Indexed Flush File... 176
FCS-GET - Direct Read Record 205
FCS-GET - Dynamic Read Record(s) 218
FCS-GET - Edit Read Line..... 226
FCS-GET - Indexed Read by Key ... 177
FCS-GET - Indexed Read Sequential Key..... 178
FCS-GET - Input Record from PC ... 272
FCS-GET - Library Read Next Line 233
FCS-GET - Sequential Read Record 211
FCS-GET-INDEX - Indexed Read for Key..... 179
FCS-GET-KEYED - Indexed Read by Key..... 181
FCS-GETRN - Indexed Read by Record Number 185
FCS-GET-SEQ-LOCK - Indexed WORKAROUND 182
FCS-GET-SEQ-NEXT - Indexed Read Next Record 183
FCS-GET-SEQ-PREV - Indexed Read Previous Record..... 184
FCS-GETUP - Direct Read With Lock 206
FCS-GETUP - Indexed Read With Lock 186
FCS-HOLD - Hold Resource..... 167
FCS-JOURNAL - Write User Journal Record..... 168
FCS-LOCK - Indexed Lock Record 188
FCS-NEXT - Indexed Get Next Record 190
FCS-NOUP - Direct Cancel Update 207
FCS-NOUP - Indexed Cancel Update 191
FCS-NOUP - Library Close Element (No update) 234
FCS-OPEN - Direct Open File 208
FCS-OPEN - Dynamic Open File.... 220
FCS-OPEN - Edit Open Buffer..... 227
FCS-OPEN - Indexed Open File..... 192
FCS-OPEN - Library Open Element 235
FCS-OPEN - Open PCXFER Interface 271
FCS-OPEN - Open TIPPRINT Interface 243

FCS-OPEN - Sequential Open File.. 212
FCS-PREV - Indexed Get Previous Record 193
FCS-PUT - Direct Update Record ... 209
FCS-PUT - Dynamic Write Record(s) 221
FCS-PUT - Edit Replace Line 229
FCS-PUT - Indexed Rewrite Record 195
FCS-PUT - Library Write Line..... 236
FCS-PUT - Output Print Line 249
FCS-PUT - Output Record to PC..... 274
FCS-PUT - Sequential Write A Record 213
FCS-RELEASE - Release Resource. 169
FCS-SCRATCH - Dynamic Scratch File 222
FCS-SCRATCH - Edit Scratch Buffer 230
FCS-SETL - Indexed Set Sequential Mode 196
FCS-SETL-BOF - Indexed Set Sequential Mode 197
FCS-SETL-EOF - Indexed Set Sequential Mode 198
FCS-SETL-EQ - Indexed Set Sequential Mode 198
FCS-SETL-GT - Indexed Set Sequential Mode 200
FCS-SKIP - Indexed Skip Sequentially 201
FCS-TREN - Mark Transaction End 170
FDES
FDES 166, 217, 232
'FDES
file descriptor packet'..... 166
field attributes
dynamic re 119
modification'..... 119
fields
display type' 88
file 164, 192, 202, 203, 204, 205, 207, 208, 209, 210, 211, 212, 213, 216, 233, 235, 241, 249
file descriptor packet'..... 165, 166
logical file name packet'..... 165
'file 266
'File Control System see FCS' 159
File Control System (FCS) 159
File Descriptor (FDES) Packet 166
file organizations
supported'..... 159
file system
function codes' 165
File System Function Codes 165
File Transfer Interface Copy Books.. 266
filename'..... 237
files
dynamic'..... 214
permanent
scratching' 222
temporary
scratching' 222
fixed format
message prefix' 143
fixed order
parameter passing' 7
flag bits'..... 31
flag services

TIPFLAG subroutine' 37
fld 107
flush
edit buffers' 226
indexed file 176
print buffer' 241
TIPPRINT buffer' 241
'flush
direct file' 205
force
full screen transmit' 118
program dump' 288
format
of calls to TIPFCS' 164
format handler' 84
FORTRAN
skip codes' 249
full screen
force transmit' 118
output' 238
function 237
TIPFCS parameter' 164
TIPFLAG functions' 37
Function Calls 154
function codes
file system' 165
function key
input' 134
Function Key Input 134
GDA
as serial resource' 25
GDA= keyword' 25
Global Data Area' 25
passed to transaction' 7
GDA - Global Data Area 25
get
data from screen format' 107
direct
read record' 205
read with lock' 206
edit buffers
read line' 226
indexed
get next record 190
get previous record 193
read by key 181
read by record number 185
read for key 179
read next record 183
read previous record 184
read record by key 177
read sequential key 178
read with lock 186
library
read next line' 233
one line from terminal' 142
sequential
read record' 211
TIPASK subroutine' 92
TIPASKYN subroutine' 94
'get
input from terminal' 147
record from queue' 67
TIPPEER record' 59
Get a Record from the Queue - FCS-GET 67

GETUP
LOCK' 27
Global Data Area
see GDA' 7
Global Data Area (GDA)
description' 25
'group
PIB 13
HEADCHAR=' 124
help
context sensitive' 123
defining help text' 124
HELP 97
Help Text Definition 124
HELP=' 124
hexadecimal
hex to decimal conversion 301
Hexadecimal - Decimal Conversion 301
HIGH 105
hold
for transaction
HOLD=TR' 163
for update
HOLD=UP' 162
simple
HOLD=YES' 162
'hold
resource
with FCS 167
HOLD=TR' 27
HOLD=TR - Record Locking for Transaction 163
HOLD=UP - Record Locking for Update 162
HOLD=YES - Simple Record Locking 162
'HP
printer option
Hewlett 238
identifying
unacceptable data fields' 105
IMA
input message area' 310
IMS
emulator' 310
program CALL to TIP 310
status codes 312
IMS Status Codes 312
index 164
indexed file
get next record 190
add record 174
and TIPFCS 173
cancel update 191
close 175
delete record 175
flush 176
get previous record 193
lock record 188
open file 192
read by key 177, 181
read by record number 185
read for key 179
read next record 183
read previous record 184

read sequential key 178
read with lock 186
rewrite record 195
sequential (greater than key) 200
sequential at beginning of file 197
sequential at end of file 198
sequential at key 198
set sequential mode 196
skip records sequentially 201
info 243
Information Management System(IMS) 309
Informational text' 105
INOUT files' 210
input
function key' 134
'input
test for terminal input' 150
INPUT files' 210
Input Message Area (IMA)' 310
insert
line in edit buffer' 223
interface level' 159
interface packet
FCS' 165
MCS' 84, 88
interface packet' 86
interpret
FCS error' 171
introduction
to PCS' 7
Introduction 151
ISAM' 159
journal
prefix 254
'journal
record' 254
Journal and QBL File Record Format 254
journal file 260
access batch 260
and failed LOGON attempts 254
close batch 260
open batch 260
read batch 260
record format 254
'journal file
processing' 254
journaled online files' 159
jrn 168, 254, 260
'JRN 168
'Julian date' 13
key 164
LANGUAGE=' 88
'last screen format' 13
LFD' 159
LFN
Logical File Name' 159
LI249
'libbat.a
library' 262
librarian services
for screen formats' 84
library
close element (no update)' 234
close element' 233
element

read next line' 233
write a line' 236
file descriptor' 232
introduction' 231
open element' 235
Library File Descriptor 232
line 83, 223
add to edit buffer' 223
line oriented
PROMPTX8 subroutine' 138
PARAM subroutine' 135
PROMPT subroutine' 137
PROMPTX subroutine' 138
ROLLPT subroutine' 141
terminal I 133
TEXT subroutine' 142
TEXT80 subroutine' 142
Line Oriented Terminal I/O 133
LINES=' 124
linkage items
dummy' 7
LINKAGE SECTION' 165
list
pick from a list' 97
'local
queue' 61
Local and Remote Queues 61
'locap
PIB 13
LOCAP
TIPSUB to another' 71
lock
indexed
lock record 188
LOCK
GETUP' 27
lock indicator
PIB 27
'lock indicator
PIB 13
logical
delete logical record' 172
file name packet' 165
'logical
TIPPEER logical name packet' 53
Logical File Name
LFN' 159
Logical File Name Packet 165
Logical Record Delete 172
'LP
Unix spooler' 238
'LPP
lines per page' 238
main storage
areas' 7
makefile 280
mark
transaction end' 170
MASK
with TIPFLAG' 37
MBP
Visual COBOL 85 278
MBP Visual COBOL 85 278
MCS 13, 88, 105, 111, 147
and TIPMSGE' 105
interface packet' 84, 88

MCS Area' 24
Message Control System' 83
override mechanism' 119
passed to transaction' 7
summary of subroutines' 86
'MCS 88
interfaces provided' 83
MCS - MCS Area 24
MCS Interface Packet 88
MCS Screen Formats 84
MCS Subroutines 86
memory
snap dump of' 289
menu
display bar' 104
message
suppression' 243
'message
output to terminal
T 149
Message Control System
see MCS' 7
Message Control System (MCS) 83
introduction' 83
MCS Area' 24
Message Formats 143
message prefix
fixed format' 143
Micro Focus
COBOL 278
Micro Focus Cobol 307
Micro Focus COBOL Makefile
(make.mf) 281
modifying
field attributes' 119
moving
HIGH 105
MSG WAIT
and function key input' 134
MSGAR
and screen formats' 84
MSGFMT
and screen formats' 84
MSGSHOW
and screen formats' 84
native mode program
general structure' 7
native mode program' 133
NOW PRINTING message' 243
NOW PRINTING message suppression' 243
number
of stack levels' 8
online
program structure' 7
online program
ending with TIPRTN' 69
Online Program Structure 7
open
direct
open file' 208
dynamic
open file' 220
edit buffer' 227
indexed
open file 192

library
open element' 235
sequential
open file' 212
'open
TIPPEER conversation' 56
TIPPRINT' 243
TIPQUEUE' 64
Open the Log File 285
Open the Queue - FCS-OPEN 64
OPEN=NO' 210, 212
operator error' 118
optimization
of output messages' 84
order of parameters
in PROCEDURE DIVISION USING statement' 7
output
data to screen format' 111
message optimization' 84
print line' 249
'output
message to terminal
T 149
OUTPUT files' 210
Output for Input Queuing from IMS Programs 311
overflow status' 243
overlay
current screen' 114
'override
row with PIB 13
packet
interface' 86
file descriptor packet (FDES)' 166
logical file name' 165
PARAM 134
line oriented subroutine' 135
PARAM - Parameterize Data 135
parameter
delimiters' 135
parameterize
an input message' 135
parameterize reply' 134
parameters
automatic passing of' 7
partial screen
transmission of' 118
partitioning an application system' 8
passing
of parameters' 7
PCS
introduction' 7
PCXFER - PC File Transfer 265
'peer 30, 52
'peer conversation
close TIPPEER' 57
for server' 59
open TIPPEER' 56
receive TIPPEER record' 59
send TIPPEER record' 58
table of' 54
perform a program
with TIPSUB' 71
Performance 265
permanent

deletion' 172
files
scratching' 222
files' 214
physical
record delete' 172
Physical Record Deletion 172
PIB ... 13, 27, 35, 41, 43, 45, 75, 81, 111, 162, 202, 203, 204, 205, 206, 207, 208, 209, 210, 211, 212, 215, 216, 217, 220, 222, 223, 226, 229, 243
passed to transaction' 7
Process Information Block' 13
'PIB 13
fields used by TIPPEER' 56
PIB - Process Information Block 13
PIB Fields Used 56
PIB-LOCK-INDICATOR Action 27
pick
from a list' 97
POC signal
definition of' 134
pop
current screen' 117
POS=' 124
Power On Confidence (POC) signal' 134
powers
of 16 302
of 2 302
Powers of 16 302
Powers of 2 302
Prepare to use batch Interface Routine 262
Primary Peer Conversation for the TIPPEER Server 59
print
accumulating lines' 238
current screen' 116
destinations' 237
screen' 111
TIPPRINT destinations' 238
TIPPRINT facility' 237
PRINT 243, 249
PRINT files' 210
'printer
supported printers' 238
PROCEDURE DIVISION
USING statement
order of five parameters' 7
Process Information Block
see PIB' 13
program
and screen interaction
example of' 84
control after CALL' 87
execution stack' 8
execution stack level' 217
native mode' 133
online program structure' 7
program to program data transfer' 8
stack
example of' 8
structure of native mode program' 7
Program Control after CALL 87
Program Control System

introduction' 7
Program Control System (PCS) 7
Program Execution Stack 8
prompt
not parameterized' 138
parameterized' 137
the user for text' 138
PROMPT
and lock indicator' 13
line oriented subroutine' 137
PROMPT - Prompt Terminal for Reply 137
prompts
issuing' 83
PROMPTX
line oriented subroutine' 133, 138
PROMPTX - Prompt for Text 138
PROMPTX8
line oriented subroutine' 138
PROMPTX8 - Prompt for Text 139
Provided Interfaces 83
'PS
printer option
postscript' 238
put
library
write line' 236
sequential
write a record' 213
'put
message to terminal
T 149
TIPPEER record' 58
QBL file 260
'queue
close' 65
get record' 67
local and remote' 61
open' 64
write to queue' 66
'queueing
TIPQUEUE description' 30, 60
quick before image' 163
R value
and MCS 105
race conditions
and FCS 202
race conditions' 161
random
set random mode 176
random mode
place file in
FCS 161
'read
see get' 107
'receive
TIPPEER record' 59
Receive Record GET 59
record 164
add to edit buffer' 223
and FCS 204, 205, 209, 211, 213, 223, 225, 226
delete logical record' 172
how to delete' 172
record' 237
record format

of journal files 254
record lock' 25, 27
record locking
example' 161
for transaction' 163
HOLD=TR' 163
HOLD=UP' 162
HOLD=YES' 162
multiple' 162
purpose' 159
simple' 162
summary' 163
Record Locking 161
Record Locking Summary 163
'record passing
TIPPEER' 54
Record Passing 54
Record-Oriented Program-to-Program Communications 9
Reference Tables 301
rel 202, 204
relative record number' 202
'release
resource
with FCS 169
'remote
queue' 61
Request Conversation OPEN 56
rereading
screen contents' 118
'resource
hold with FCS 167
release with FCS 169
RESULT
with TIPFLAG' 37
retrieve elective groups
TIPGRPS subroutine' 47
Return Status from Unix System Calls 308
rewrite
indexed
rewrite record 195
ROLL
and TIPPRINT' 238
'ROLL
output a line
roll screen' 140
ROLL - Output Line Roll Screen 140
rollback
record locking for transaction' 163
ROLLBACK
and PIB 27
and transaction end' 25
ROLLPT
set terminal roll point' 141
ROLLPT - Set Terminal Roll Point 141
RPG
PIB 13
'run
DOS or Windows program with TIPWINAP' 80
Sample Log Files 290
sample program
listing of tstwin' 126
'schedule
TIPQUEUE' 63

scratch
dynamic
scratch file' 222
edit buffer' 230
edit buffers
scratch buffer' 230
screen
and program interaction
example of' 84
erasing' 96
error fields' 84
force transmit' 118
format name' 88
overlay current screen' 114
pop current screen' 117
print current screen' 116
rereading contents' 118
send error text to' 105
'screen
output a line
roll screen' 140
screen format
output data to' 111
'screen format
last used' 13
read from' 107
screen formats
librarian services' 84
overview' 84
screen formats' 83
Security 265
SEL 97
send
error text to screen' 105
'send
TIPPEER record' 58
Send a Record PUT 58
sequential file
close file' 210
open file' 212
read record' 211
write a record' 213
'sequential file
and TIPFCS' 210
sequential mode
indexed
at end of file 198
indexed
greater than key 200
set with FCS 196
start at key 198
set a file in' 161
'sequential mode
indexed
at beginning of file' 197
serial resource
GDA as' 25
'service
TIPQUEUE schedule' 63
set
terminal roll point' 141
Setting a File in Sequential Mode 161
setuid
UNIX permissions 279
single line
terminal output' 238

'site name' 13
skip
indexed
skip records sequentially 201
skip codes' 249
snap dump
memory' 289
of memory with TIPSNAP' 70
source 296
Source-level Debugging..... 296
space
field characteristic' 119
stack
climbing the' 8
example of' 8
levels
number of' 8
program execution' 8
'stack
PIB 13
start
background program in new window' 45
background program with TIPFORK' 43
program at a terminal 41
status
for FCS . 202, 203, 204, 205, 206, 207, 208, 209, 210, 211, 215, 216, 217, 218, 220, 221, 222, 229
for TIPDXC' 35, 41
for TIPFORK' 43
for TIPFORKW' 45
PIB..... 41
'status
and FCS 174
FCS..... 175, 176, 177, 179
for FCS 66
for TIPPEER'..... 56
for TIPUSRID' 79
for fcs..... 65, 67, 168, 175, 176, 178, 181
for TIPGRPST' 48
for TIPMSG'..... 49
for TIPQUEUE' 64
for TIPRTN' 69
for TIPSUB'..... 71
PIB..... 13
status codes
IMS 312
storage
main areas' 7
storage area addresses' 7
structure
of native mode program' 7
STYLE=' 124
subroutines
for MCS' 86
line..... 83
SUCCESSOR..... 310
suggestion
coding'..... 8
summary
of MCS subroutines' 86
of record locking'..... 163
of TIPFCS'..... 164
'supported

printers' 238
Supported COBOL Compilers 277
'T 147, 149, 150
TAPE files' 210
TC 13, 24, 31, 34, 37, 43, 143, 165, 166, 168, 232, 243, 249, 254
'TC 22, 37, 47, 88, 146, 249
Techniques for Deleting Records 172
templates
screen formats' 83
temporary
files' 214
files
scratching' 222
terminal
destination' 88
line oriented I 133
set roll point' 141
'terminal
PIB 13
test for input
T 150
terminal id
PIB 13
terminals
supporting FCC' 119
'terminals
using two for debugging' 296
TERMINATION 310
'test
for input
T 150
test for input' 150
testing
screen formats' 84
text
and TIPMSGEO' 107
TEXT 138, 142
get line from terminal' 142
TEXT - Get One Line From Terminal 142
TEXT80
get line from terminal' 142
TEXT80 - Get One Line From Terminal 142
TFD
and screen formats' 84
program' 88
T-GET - Get Input 147
time 75
PIB 13
'time
to wait for input' 13
TIMEOUT=' 13
timer
TIPTIMER subroutine' 75
TIP 8, 25, 37, 73, 159, 223, 237, 310
'TIP
summary of MCS subroutines' 86
TIP and IMS Interaction 310
TIP Print Facility (TIPPRINT) 237
TIPASK
display line
get answer' 92
TIPASK - Display One Line and Return Answer 92

TIPASKYN
display line
get answer' 94
TIPASKYN - Display One Line and Return Answer 94
'tipbatpi.o
interface subroutine' 262
tipbatpi.o Interface Subroutine 263
'tipbatsv
batch interface program' 262
TIPBITS
convert bytes to bits ' 31
TIPBITS - Convert Bytes to Bits 31
tipbstpi.o
entry points 263
TIPBYTES
convert bits to bytes' 34
TIPBYTES - Convert Bits to Bytes 34
TIPDATE
subroutine to get date' 34
TIPDATE - Return Date 34
'TIPDUMP
force program dump' 35
TIPDUMP' 288
TIPDUMP - Force Program Dump 35
TIPDXC
and IMS' 310
and program stack' 8
delayed transfer control' 35
TIPDXC - Delayed Transfer Control .. 35
TIPERASE
erase screen' 96
TIPERASE - Erase Screen 96
TIPFCER
interpret FCS error' 171
TIPFCS
and direct files' 202
and dynamic files' 214
and indexed files 173
calling format' 164
edit buffers' 223
library files' 231
overview' 159
'TIPFCS
and sequential files' 210
TIPFCS for Direct Files 202
TIPFCS for Dynamic Files 214
TIPFCS for Edit Buffers 223
TIPFCS for Indexed Files 173
TIPFCS for Library Files 231
TIPFCS for Sequential Files 210
TIPFLAG
flag services subroutine' 37
TIPFLAG - Flag Services 37
TIPFORK
and user id' 43
start background program' 43
start program at a terminal 41
TIPFORK - Start Background Program 43
TIPFORK - Start Program at a Terminal 41
TIPFORKW
start background program in new window' 45
user id

additional considerations'..... 45
TIPFORKW - Start Program in New Window..... 45
TIPGRPS
retrieve elective groups'..... 47
TIPGRPS - Retrieve Elective Groups. 47
TIPGRPST
change elective groups'..... 48
TIPGRPST - Change Elective Groups 48
TIPJRNCL
journal close..... 260
TIPJRNGT
journal read..... 260
TIPJRNOP
journal open..... 260
TIPLIST
HELP..... 97
pick from a list'..... 97
SEL..... 97
TIPLIST - Pick From a List..... 97
TIPLOG'..... 285
TIPLOG - Updating the Log File..... 285
TIPMENU
display menu bar'..... 104
TIPMENU - Display Menu Bar..... 104
'TIPMSG
retrieve error message'..... 49
TIPMSG - Retrieving Error Messages 49
TIPMSGE
send error text'..... 105
uses of'..... 105
TIPMSGE - Send Error Text To Screen..... 105
TIPMSGEO
define deferred error text'..... 107
TIPMSGEO - Define Deferred Error Text..... 107
TIPMSGI
and lock indicator'..... 13
read data from screen format'..... 107
use of'..... 84
TIPMSGI - Read Data from Screen Format..... 107
TIPMSGO
MCS..... 111
output data to screen format'..... 111
use of'..... 84
TIPMSGO'..... 88
TIPMSGO - Output Data to Screen Format..... 111
TIPMSGOV
overlay current screen'..... 114
TIPMSGOV - Overlay Current Screen..... 114
TIPMSGPR
print current screen'..... 116
TIPMSGPR - Print Current Screen... 116
TIPMSGRS
pop current screen'..... 117
TIPMSGRS - Pop the Current Screen..... 117
TIPMSGRV
force full screen transmit'..... 118
TIPMSGRV - Force Full Screen Transmit..... 118
TIPPAGE Paging API..... 152
'TIPPEER

close conversation' 57
get a record' 59
logical name packet' 53
open conversation' 56
peer 30, 52
PIB fields used' 56
primary peer conversation for server' 59
record passing' 54
send a record' 58
table of typical conversation' 54
transaction processing' 60
TIPPEER - Peer-to-Peer Processing ... 52
TIPPEER Logical Name Packet 53
TIPPRINT
AUX0' 238
AUX1' 238
print subroutine' 237
ROLL' 238
'TIPPRINT
open' 243
TIPPRINT Buffering' 249
TIPPRINT Print Destinations 238
TIPPRINTAUX
environment variable' 238
'TIPQUEUE
calling' 64
close' 65
description' 30, 60
get record' 67
local and remote queues' 61
open queue' 64
service time schedule' 63
write to queue' 66
TIPQUEUE - Record Queuing 60
TIPQUEUE Interface (API) 64
TIPQUEUE Service Time Schedule ... 63
TIPRTN
and lock indicator' 13
end online program' 69
summary' 7
TIPRTN - End Online Program 69
TIPSNAP
snap dump memory' 289
TIPSNAP - Snap Dump Memory 70
TIPSUB
and lock indicator' 13
perform a program' 71
to another LOCAP' 71
TIPSUB - Perform Program 71
TIPSUBP - Call a Subprogram 74
TIPTERM
terminal functions' 146
'TIPTERM
get input
T 147
output message
T 149
test for input
T 150
TIPTERM' 143
TIPTERM Functions 146
TIPTIMER
wait for n seconds' 75
TIPTIMER - Timer Services 75
TIPTITLE

display title' 119
TIPTITLE - Display Title 119
tipusr
where is user 78
TIPUSR - Where is User 78
TIPUSRID
user information subroutine' 79
TIPUSRID - User Information 79
'TIPWINAP
run a DOS or Windows program 80
TIPWINAP - Run a DOS or Windows Program 80
TIPXCTL
and IMS' 310
and program stack' 8
transfer control' 81
TIPXCTL - Transfer Control 81
title
display title' 119
'TOF
top of form' 238
T-PUT - Output Message 149
transaction
end' 25
end
deferral of' 25
explicit end' 25
id
PIB 13
initiation' 25
record locking for' 163
termination' 25
'transaction
mark end' 170
processing with TIPPEER' 60
Transaction End 25
Transaction Processing Using TIPPEER 60
transfer
data from program to program' 8
transfer control
with TIPXCTL' 81
transmitting
partial screen' 118
truncated input
message' 143
tstwin
and MCS windowing features' 126
sample program listing' 126
TSTWIN - Sample TIP/ix Program .. 126
T-TEST - Test For Input 150
Types of Executables 279
U character
uppercase data fields' 84
unacceptable
data fields
identifying' 105
unique screen formats' 84
UNIX permissions
setuid 279
Unix Shell Error 307
update
direct
update record' 209
indexed
rewrite record 195

updating
screen formats' 84
uppercase data fields' 84
'UPSI
PIB 13
user
security level' 13
USER 79
user id
and TIPFORK' 43
PIB 13
with TIPFORKW' 45
user information
TIPUSRID subroutine' 79
USING statement
order of five parameters' 7
utilities
calling' 73
V DI 249
validation of data' 105
wait 75
wait for n seconds' 75
WHOSON
display execution stack' 8
windowing features
sample program' 126
WORK 7, 24
'work area
PIB 13
Work Area' 7
Work-Area 24
'write
to queue' 66
user journal record' 254
Write a Record to a Queue - FCS-PUT 66
Write a Text Message to the Log File 286
'WRK=
and PIB 13
XI$BATCH
equivalent 262