UNISYS

System 80 OS/3 Transaction Platform System (TPS)

Programming Guide

Volume II

IMPORTANT NOTE!

This documentation describes the *Transaction Platform System* (TPS) software product which is a fully functional subset of Allinson-Ross Corporation *Transaction Interface Processor /30* (TIP/30).

All references to **TIP/30** in this documentation can be understood to refer to **TPS**.

TIP/30 is a trademark of Allinson-Ross Corporation, Mississauga, Ontario, Canada. **TPS** is a trademark of Unisys Corporation.

OS/3 Release 13

August 1990

Printed in U S America

Priced Item

7002 3999-100

About This Document

Purpose

The *TIP/30 Programming Reference Manual Volume II*:

- explains TIP/30 programming facilities and
- describes, in detail, how to install, generate and maintain TIP/30.

Scope

This document provides detailed descriptions of the TIP/30 programming facilities and provides instructions on how to install, generate and maintain TIP/30.

Audience

The primary audiences for this document are:

- data processing programming staff
- system administrators
- system operators.

Prerequisites

Anyone using this document should be familiar with Unisys computer hardware and the OS/3 software system.

In addition, knowledge of the COBOL programming language is an asset.

How to Use This Document

You should read this entire document to familiarize yourself with its contents. The sections on the TIP/30 programming facilities explain how to use these facilities in your applications. Additional sections describe how to install, generate and maintain TIP/30.

Organization

This document contains ten sections and an appendix:

"Program Control System (PCS)" on page 1-1

This section describes the facilities of the Program Control System (PCS). All TIP/30 facilities that provide program control are included in this classification.

"Message Control System (MCS)" on page 2-1

This section describes the facilities provided by TIP/30 to enable an online program to perform input and output to a terminal.

"File Control System (FCS)" on page 3-1

This section describes the facilities of the TIP/30 File Control System (FCS). FCS is the TIP/30 component that provides the interface between transaction programs and data files.

"TIP/30 System Generation" on page 4-1

This section describes TIP/30 generation parameters and procedures.

"TIP/30 System Files" on page 5-1

This section describes each of the files required for the operation of the TIP/30 system.

"TIP/30 Job Control Options" on page 6-1

This section describes the job control used to execute TIP/30 and the run-time job control options that may be specified for TIP/30.

"Offline (Batch) Recovery" on page 7-1

This section describes offline file recovery procedures.

"TIP/30 Batch Jobs" on page 8-1

This section describes the batch job control streams supplied to support the TIP/30 system.

"Operations Guide" on page 9-1

This section contains information needed by operations personnel to operate the TIP/30 system.

"TIP/30 Installation Guide" on page 10-1

This chapter is the installation guide for TIP/30.

"Programming Reference" on page A-1

This appendix contains programming reference tables.

Contents

Section 1
Program Control System (PCS)

This chapter describes the facilities of the Program Control System (PCS). All TIP/30 facilities that provide program control are included in this classification.

PCS, as a component of TIP/30, 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/30 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/30 PCS routines.

The linked interface routines are rarely changed, thus ensuring upward compatibility from one release of TIP/30 to another.

1.1. PCS Facilities

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.

TIPABRT Establish exception routine to handle abnormal termination conditions.

TIPABRT allows a program to trap certain types of abnormal termination events (this subroutine is available only for Assembler language transaction programs).

TIPBITS Convert a series of 32 bytes to 32 bits.

TIPBITS is a utility subroutine provided to permit COBOL language programs to manipulate bit values.

TIPBYTES Convert a series of 32 bits to 32 bytes.

TIPBYTES is a utility subroutine provided to permit COBOL language programs to manipulate bit values.

TIPDATE Return date in readable format
(example: TUESDAY OCTOBER 18 1988).

TIPDATE is a utility subroutine provided to return the date in expanded format (including day of the week).

TIPDUMP Cause deliberate program check.

TIPDUMP is a utility subroutine provided to cause a deliberate program check condition and, therefore, force a program dump for debugging purposes.

TIPDXC Transfer control to another transaction program *after* the arrival of an input message from the terminal.

TIPDXC allows 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).

TIPFLAG is a utility subroutine provided to permit transaction programs to manipulate internal TIP/30 flag bits and use these flags as semaphores to implement queueing schemes.

TIPFORK Start a transaction program running as an asynchronous process.

With TIPFORK, a program can initiate another program as an asynchronous task, and thus create an independently executing process.

The independent process may run at another terminal in the network or as a "background process" (without a connected terminal).

TIPGRPS Retrieve elective group membership.

The TIPGRPS subroutine is used to retrieve the names of the application groups to which the user has membership.

TIPGRPST Set elective group membership.

The TIPGRPST subroutine is used to change or set the names of the application groups to which the user belongs.

TIPJUMP The TIPJUMP subroutine permits an online program to transfer control directly to another program — passing *all* of the calling program's work areas to the next program.

This call essentially allows a program to continue executing using a different load module.

TIPRTN Terminate transaction program and return control to calling program.

All TIP/30 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 purposes.

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 Invoke a resident subroutine.

TIPSUBP allows a program to "CALL" a resident subroutine and receive control when that subroutine is finished.

TIPTIMER Delay program execution for a specified number of seconds.

TIPUSR Retrieve terminal name where a specified user is using TIP/30 system.

TIPUSRID Retrieve information about TIP/30 user.

TIPXCTL "GOTO" another program.

Using TIPXCTL, a program can "GO TO" another program without any return of control.

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/30 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/30 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.

1.2. Online Program Structure

TIP/30 provides an environment for transaction programs. Part of the environment is a number of areas of main storage that are established automatically for use by the program. Some areas are used to communicate information to the TIP/30 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/30 calls a transaction program exactly as if the program was a subroutine of TIP/30. 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/30 native mode must be aware of the parameters that are automatically passed by TIP/30. All transaction programs are called either by TIP/30 (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/30 native mode program. For convenience, the examples use COBOL syntax.

TIP/30 passes five parameters to a transaction program, in the following fixed order:

1. PIB	Process Information Block
2. CDA	Continuity Data Area
3. MCS	Message Control System work area
4. WORK-AREA	Work area
5. GDA	Global Data Area (TIP/30 generation option)

Each of these areas represents main storage, established by TIP/30, that the transaction program may use.

Example:

```
DATA DIVISION.
LINKAGE SECTION.

01  PIB.                          COPY TC-PIB OF TIP.
01  MCS.                          COPY TC-MCS OF TIP.
01  WORK-AREA.
    . . .
01  CDA.                          COPY TC-CDA OF TIP.
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 very 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.

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.

OS/3 programming languages do 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.

1.3. Program Execution Priority

Priority levels that are established in the TIP/30 generation parameters control the execution priority of an online program. The TIP/30 generation keywords that control these execution priorities are:

- BACKPRI=
- SCHDPRI=
- PRIORITY=
- USERPRI=.

The execution priority is also related to the priority of other jobs in the OS/3 system, and not just TIP/30 activities; therefore, you should specify transaction priorities with care.

The PRIORITY= keyword in the TIP/30 Catalogue entry for the PROGram may be used to designate a specific priority level for an individual transaction program.

Note: *Use of these facilities may result in TIP/30 transactions running at OS/3 switch list priorities (numerically) higher than 4 and, therefore, may impact other tasks in the system such as: output writer, run processor, interactive services, etc. If changes are made to the TIP/30 transaction priority levels, the values assigned to related OS/3 generation options (SYMBPRI=, ISINTPRI=, PRIORITY=) must be examined.*

The following discussion will help you to better understand transaction scheduling. First, the OS/3 execution priorities that TIP/30 uses:

Assume that TIP/30 has been executed using a priority of "X" on the EXEC statement in the job control. This value is often "1", but we will denote it as "X" for the purpose of this discussion.

The default situation is as follows:

- The TIP/30 MAIN task runs at priority level **X**; this level may not be altered
- The TIP/30 COMMunication task runs at priority level **X+1**; this level may not be altered
- Foreground transaction programs (those executing at a terminal) run at priority level **X+2**
- The TIP/30 scheduler task and *background* transaction programs run at priority level **X+3**.

There are several generation keywords (or TIP/30 job control options) that allow the specification of both the number of user transaction priority levels and the levels that are used for various types of transactions:

PRIority=p The number of required priority levels.

Default: p=2.

Maximum: 10

The range of priority levels established by the value specified for this keyword range from X+2 to X+2+p-1 inclusive.

Note: *The PRIority= keyword establishes the number of priority levels that exist* **beyond** *the levels that are reserved for the MAIN and COMM tasks (running at levels X and X+1 respectively).*

The following keywords specify a numeric value that determines the various priorities relative to X+1:

UserPRI=u

Priority level for foreground transactions. Foreground transactions execute at priority level X+1+u.

Default: u=1.

BackPRI=b

Priority level for background transactions. Background transactions execute at priority level X+1+b.

Default: b=2.

SchdPRI=s

Priority level for the TIP/30 scheduler task. The scheduler executes at priority level X+1+s.

Default: s=2.

You should **not** run the TIP/30 scheduler at USERPRI because the scheduler should not preempt user transactions. If you do, scheduling new work takes priority over transactions that are already running.

Additional Considerations:

Running a program using IDA (Interactive Debug Aid) forces that program's scheduling priority to the lowest (worst) priority: (X+1+(value of Priority= keyword)).

1.4. Program Execution Stack

TIP/30 transaction programs operate in a stack oriented environment. The standard system prompt is displayed by the TIP/30 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/30 prompt is regarded as stack level zero.

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

On the other hand, if the initial program transfers control to another program with an implied return of control, TIP/30 does the following:

- Suspends execution of the calling program
- Saves the calling program's "activation record" (PIB, CDA, MCS, and WORK-AREA) in the TIP$SWAP file.
- 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).

This process of "climbing" the stack may proceed up to 64 levels. When any program issues a call to the TIPRTN subroutine, TIP/30 does the following:

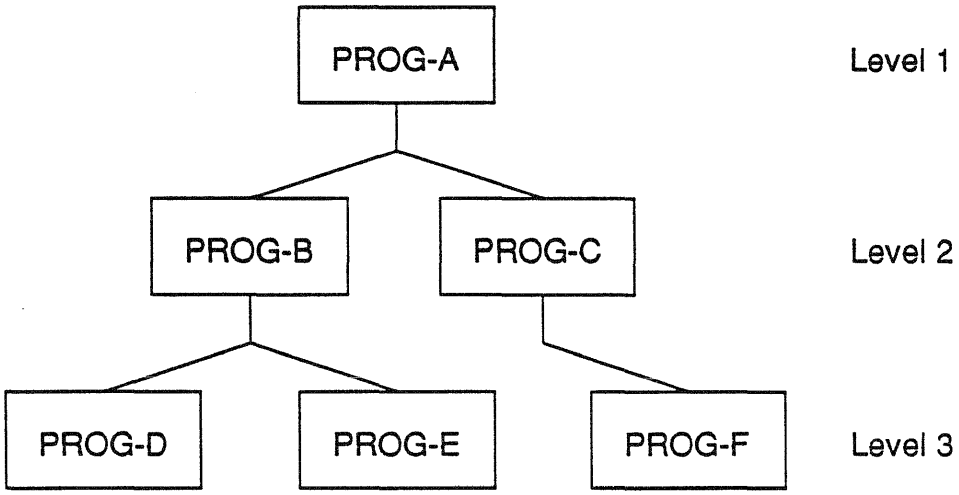
- Loads (from the TIP$SWAP file) the saved "activation record" of the program that preceded the terminating program on the execution stack
- Copies the contents of the CDA of the terminating program to the CDA of the previous program on the stack (for a length of the shorter of the two CDA areas)
- De-allocates the PIB, MCS, and WORK-AREA of the terminating program
- Resumes execution of the program that invoked the terminating program.

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/30 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) terminate 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!

The ability to stack or nest program execution is illustrated by the following hierarchy of programs:

Example Program stack:



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/30 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 (TIPJUMP, TIPXCTL, TIPDXC) or PERFORM (TIPSUB).

Issuing a call to TIPSUB involves TIP/30 system overhead since the current activation record is saved in the TIP$SWAP file. This overhead is somewhat more than that required for TIPXCTL or TIPDXC.

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/30 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.

Using a **subprogram** (CALL TIPSUBP) to perform I/O is a different situation altogether and is a much better idea.

1.5. 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/30 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/30 subroutine.

The layout of the PIB is contained in the COBOL copy element "TC-PIB" supplied in the TIP library:

```
*
*          TIP/30  -  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-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 S9(6) COMP-3.
     05  PIB-TIME                          PICTURE S9(6) COMP-3.
     05  PIB-JULIAN-DATE.
         10  PIB-YEAR                      PICTURE 9(2) 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(4) 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(4).
05  PIB-WAIT-TIME                    PICTURE S9(4) COMP-4.
05  PIB-DETAIL-STATUS                PICTURE 9(4) COMP-4.
    88  PIB-DUPS-AHEAD               VALUE  1.
05  PIB-LOCK-INDICATOR               PICTURE X(1).
    88  PIB-ROLLBACK                 VALUE 'O'.
    88  PIB-RELEASE                  VALUE 'R'.
    88  PIB-HOLD                     VALUE 'H'.
05  PIB-RPG-UPSI                     PICTURE X(1).
05  PIB-ALT-MCS-ROW                  PICTURE 9(2) COMP-4.
05  PIB-CDA-I                        PICTURE 9(6) COMP-4.
05  PIB-WRK-I                        PICTURE 9(6) COMP-4.
05  PIB-LEVEL                        PICTURE 9(3) COMP-4.
05  PIB-TERM-TYPE                    PICTURE 9(4) COMP-4.
    88  PIB-U200                     VALUE 0.
    88  PIB-UTS-400                  VALUE 1.
    88  PIB-UTS-20                   VALUE 2.
    88  PIB-UTS-30                   VALUE 3.
    88  PIB-UTS-40                   VALUE 4.
    88  PIB-UTS-10                   VALUE 5.
    88  PIB-UTS-60                   VALUE 6.
    88  PIB-PC                       VALUE 7.
    88  PIB-IBM-3270                 VALUE 8.
    88  PIB-BI-SYNC                  VALUE 9.
    88  PIB-Q310                     VALUE 10.
    88  PIB-TELETYPE                 VALUE 11.
    88  PIB-OFIS-LINK                VALUE 12.
    88  PIB-OFIS-PC                  VALUE 13.
    88  PIB-MAPPER-5                 VALUE 14.
    88  PIB-MAPPER-6                 VALUE 15.
    88  PIB-PC-UTS-30                VALUE 16.
    88  PIB-PC-UTS-60                VALUE 17.
    88  PIB-DOPS                     VALUE 18.
    88  PIB-VIPS                     VALUE 19.
    88  PIB-UNIX                     VALUE 20.
    88  PIB-UTS-20-WKST              VALUE 21.
    88  PIB-SVT-112X                 VALUE 22.
    88  PIB-SPECIAL-A                VALUE 23.
    88  PIB-SPECIAL-B                VALUE 24.
```

```
05  PIB-MIRAM-REL-REC-NUM          PICTURE 9(7) COMP-4.
05  PIB-CDA-SIZE                  PICTURE 9(7) COMP-4.
05  PIB-MCS-SIZE                  PICTURE 9(7) COMP-4.
05  PIB-WRK-SIZE                  PICTURE 9(7) COMP-4.
05  PIB-CDA-LENGTH                PICTURE 9(7) COMP-4.
05  PIB-LANGUAGE                  PICTURE X(1).
05  PIB-WDEL-INDICATOR            PICTURE X(1).
    88  PIB-WAIT-DELIVERY         VALUE 'Y'.
    88  PIB-WAIT-DELIVERY         VALUE 'N'.
```

Where:

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/30 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/30 subroutine that required modification of this field (example: TIPSUB, TIPSUBP).

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

userid

The userid of the user that is executing the program.

TP

If the user is not logged on TIP/30.

BACK$nnn

If the executing program is running as a background process. "nnn" is 3 digits representing the assigned background process number.

CONSOLE

If the program was executed via the "EXEC" console command.

tttt*BYP

If the program is executing on a "bypass" terminal. "tttt" is the terminal name of the originating terminal.

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

Certain TIP/30 subroutine calls, for example TIPFORK, may require the program to move information to this field.

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

PIB-STATUS

This one byte field contains the status returned as a result of a call to a TIP/30 subroutine. A number of 88 level items are defined in the copy element for your convenience.

It is *strongly recommended* that programs interrogate this status field after a call to a TIP/30 subroutine. Subroutine calls that work one day may fail miserably the next due to unforeseen external influences — for example, TIP/30 Catalogue entries may be "improved" or otherwise modified, thereby causing much grief.

Note: *The TIP/30 Message Control System (MCS) also uses an additional status field in the MCS area (MCS-STATUS). The documentation of the various calls to MCS describe 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/30 is concerned. Any other value may be an error — although it may be only a warning.

A transaction program may interrogate this field on initial entry to the program. If the status is not PIB-GOOD, the error indicates that TIP/30 was unable to access one or more of the files that were identified in the program's catalogue record.[1]

PIB-SYSTEM

This one byte field is set to the value "PIB-EOJ-PENDING" if and only if TIP/30 has been given the shutdown command "EOJ".

This mechanism allows TIP/30 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.

If the TIP/30 system has been configured for more than 2 elective groups (by the specification of the NUMGRPS= TIPGEN keyword), 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 TIPGRPS.

1. The files required by an online program may be implicitly assigned by the use of the FILES= keyword in the catalogue entry for the program.

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/30 operates internally, this field may not be accurate. The best resolution is approximately 1 second (this field is updated by TIP/30 as a side effect of calling some of the TIP/30 routines; between calls to TIP/30 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 specified in the TIP/30 generation parameters or job control options.

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).

Refer to the description of the TIP/30 Catalogue Manager program (CAT) for a detailed discussion of security levels and their use.

In the TC-PIB copy element, 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/30.

For further information, see the discussion of the keyword ACCT= for the "USER" command in the TIP/30 CAT utility program (catalogue manager utility).

PIB-LAST-MCS-NAME

This field contains the name of the last TIP/30 screen format used at this terminal.

If the last message output to the terminal was not issued via the TIP/30 Message Control System (MCS) this field contains low-values.

PIB-LOCAP

This field contains the name of the LOCAP (local application in a Global ICAM) where the program is running. If a dedicated ICAM is in use, the

contents of this field are not defined.

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/30 generation parameter TIMEOUT=.

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/30 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.

PIB-LOCK-INDICATOR

A program sets this field to indicate to the system the type of record lock handling desired.

TIP/30 examines this field whenever the program calls TIPRTN, TIPSUB, TIPDXC, TIPFORK, TIPJUMP, TIPXCTL or calls TIPFCS with a function code of FCS-TREN or solicits terminal input (by calling TIPMSGI, PROMPT, etc.).

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 TIP$B4 file.

PIB-ROLLBACK (O)

All updates that were made to files that were generated as "hold for transaction (HOLD=TR)" are rolled back and a TREN (transaction end) record is written to the TIP$B4 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.

For 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/30. The recommended technique is to move the appropriate value to this field before calling a TIP/30 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/30 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.

To permit this override action, mark the screen format using the MSGAR command "ALTRON" — alternate row on.

This field is cleared to zero when the transaction begins; thereafter, TIP/30 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 — if the screen format is marked with "ALTRON".

The contents of this field have no effect if a screen format is used that was NOT marked "ALTRON".

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 catalogue 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 catalogue record.

PIB-LEVEL

This field contains the current program execution stack level. Refer also to the description of the program stack in the previous section of this document.

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/30 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/30 File Control System reads a record from a MIRAM file, this binary fullword 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/30 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.

Note: *This field is always interpreted by the TIP/30 system in multiples of 256 bytes. A value which is not already a multiple of 256 is adjusted to the next higher multiple of 256.*

PIB-MCS-SIZE

This field is set by the TIP/30 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-WRK-SIZE

This field is set by the TIP/30 system to the size of the program's WORK area. This value represents the number of bytes in the WORK 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 lesser 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.

Note: *This field is always interpreted by the TIP/30 system in multiples of 256 bytes. A value which is not already a multiple of 256 is adjusted to the next higher multiple of 256.*

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/30 Catalogue USER record for the userid.

See the description of the LANGuage= keyword of the USER command for the CAT utility transaction.

PIB-WDEL-INDICATOR

Set this field to the value "Y" to request TIP/30 to unconditionally wait for ICAM delivery notification on the next terminal output message. TIP/30 does not necessarily wait for delivery notification after every terminal output message. Set this field to "Y" if your program needs to be sure that ICAM acknowledges delivery of the message.

TIP/30 resets this field to "N" after each output message.

1.6. CDA — Continuity Data Area

The Continuity Data Area (CDA) is an area of storage that TIP/30 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/30 catalogue 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 — even if no CDA size is specified in the TIP/30 catalogue entry for the program.

TIP/30 always allocates the CDA in multiples of 256 bytes. If the catalogue CDA size is not a multiple of 256, the TIP/30 system adjusts the size to the next highest multiple of 256.

If a transaction program is called from the TIP/30 command line and the transaction is defined with CML=YES in the TIP/30 catalogue, the TIP/30 Command Line Processor (TCP) will place data from the command line into the program's CDA.

The COBOL copy element TC-CDA in the TIP library defines the format for this particular use of the CDA:

```
*
*        TIP/30  -  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).
```

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 upper case by the TIP/30 command line processor (TCP).

For more information about parameterization, refer to "2.5.3. PARAM — Parameterize Data" on page 2-34.

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 upper case by the TIP/30 command line processor (TCP).

CDA-TEXT

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

Alphabetic characters in the CDA-TEXT area are forced to upper case by the TIP/30 command line processor (TCP).

Additional Considerations:

If the program was **not** called from the TIP/30 command line, the layout and contents of the CDA are entirely at the discretion of the calling program.

1.7. MCS — MCS Area

The Message Control System Area (MCS) is an optional area that TIP/30 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/30 catalogue entry for the transaction.

The MCS area is initially set to low values (X'00') by TIP/30.

The COBOL copy element TC-MCS in the TIP library defines the layout of the MCS packet prefix that is required to interface with the Message Control System.

```
*
*       TIP/30  -  MESSAGE  CONTROL  SYSTEM  PACKET
*
     02  MCS-NAME                        PICTURE X(8).
     02  MCS-TERM                        PICTURE X(4).
     02  MCS-FUNCTION                    PICTURE X.
         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 SYNC.
     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-F-REBUILD               VALUE '1' '5' 'N'.
           88  MCS-F-NEXT                  VALUE '2' '6'.
           88  MCS-F-UPDATE                VALUE '4' '8'.
       02  MCS-FILLER                    PICTURE X.
           88  MCS-UNDERLINE               VALUE '_'.
           88  MCS-ASTERISK                VALUE '*'.
       02  MCS-COUNT                     PICTURE S9(4) COMP-4 SYNC.
/
       02  MCS-DATA.
*
*      USER SUPPLIED RECORD LAYOUT FOR MCS SCREEN FOLLOWS HERE
*
```

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

1.8. Work-Area

The WORK-AREA is an optional area that TIP/30 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/30 catalogue 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). It is this value that is specified in the TIP/30 Catalogue entry for the program.

TIP/30 programs use the work area as an area containing fields that are modified during execution. The modification of any field in the COBOL WORKING-STORAGE section is not allowed by the COBOL compiler when the program is compiled with the IMSCOD=YES or IMSCOD=REN option.

The work area is the proper place for the various record areas for files that are manipulated online.

TIP/30 sets the work area to low values (all X'00') before the transaction program is entered.

1.9. GDA — Global Data Area

The Global Data Area (GDA) is an optional area that may be configured when TIP/30 is generated. If the GDA is generated in the TIP/30 system, it is an area of fixed (specified) size that is accessible by all TIP/30 programs.

The first fullword of the GDA is set to the length[2] (in bytes) of the GDA by TIP/30 initialization. The remainder of the GDA is cleared to low values (X'00').

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.

Note: *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*
- *subroutine TIPFCS (functions FCS-HOLD and FCS-RELEASE)*

A very convenient way to initialize the contents of the GDA is to write a transaction program for that purpose and specify that transaction name as the system STARTUP transaction. Refer to the description of the TIP/30 generation parameter STARTUP= for more information about this technique. The documentation of the *utility transaction* STARTUP also contains valuable information about this topic.

TIP/30 installations which make use of the Global Data Area should consider creating a local COPY element that user-written programs can use to define the layout of the GDA.

WARNING

The TIP library is completely rebuilt when a new release of TIP/30 is installed — do not place such COPY elements in the TIP library! Place user copy elements in the same library that is used for user-written online programs.

Additional Considerations:

Programs that **modify** the contents of the GDA must be defined in the TIP/30 Catalogue with the specification DEBUG=NO, otherwise they will abort with "Protection Exception" if they attempt to move data to any portion of the GDA.

2. The size of the GDA is specified by the TIP/30 generation parameter GDA=.
The length is for information purposes only — user programs may overwrite this information if desired.

1.10. Reentrant COBOL

The COBOL compiler provides the capability to generate a reentrant online program. Prior to OS/3 Release 10, the COBOL compiler created object code that was only "sharable" in the sense that each user had to have a unique VOLatile area, which TIP/30 and IMS systems have to internally manage for the user program.

The Release 10 (and later) version of the compiler can generate an object program that eliminates the VOL area by introducing a somewhat larger fixed size area that the compiler assumes is located at the end of the program's work area (regardless of the type of program: IMS or TIP/30). You must add the size of this reentrant work area to the size of the work area needed by the program.

To create a reentrant transaction program, include the following run time PARAM statement in the compile job control:

```
// PARAM IMSCOD=REN
```

Note: *This facility is relevant ONLY for online programs!*

If you use this feature you will notice that the compiler now generates a message indicating the number of bytes that must be **added** to the apparent size of the **WORK-AREA** to support reentrant programs:

```
RE-ENTRANCY CONTROL = xxxxxx WORK AREA BYTES
( NOT INCLUDING PROGRAM DEFINED DATA AREAS )
```

This message appears on the title page of the compile (in the box that outlines the selected compiler options).

The following table summarizes the resultant usage of an online COBOL program that uses a *linked subroutine* that is also written in COBOL:

Main	Subroutine	Result
Shared	Shared	Reload (TIP) Reuse (IMS)
Shared	Reent	Shared
Reent	Shared	Reload (TIP) Reuse (IMS)
Reent	Reent	Reent

Note: *In the above table "Shared" refers to the result of specifying IMSCOD=YES and "Reent" refers to the result of specifying IMSCOD=REN.*

Online programs that are compiled with IMSCOD=REN *must* specify a WORK-AREA that is larger than the size indicated in the DATA DIVISION MAP by the amount stated in the compilation summary. The TIP/30 catalogue entry VOL= need not be stated in this case. The appropriate work area size must be reserved in the TIP/30 Catalogue entry for the program *even if the program does not have a work area defined in the LINKAGE SECTION.*

COBOL programs that are compiled with IMSCOD=REN that call linked COBOL subroutines may be used in a reentrant manner only if all of the linked COBOL subroutines were also compiled with the IMSCOD=REN option.

In any case, the size of the work area for the transaction must also include the sum of all of the "reentrant control areas" — for the main program and for all linked COBOL subroutines.

1.11. Transaction End

In TIP/30 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/30 to consider that the transaction has terminated.

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

1. TIP/30 or OS/3 or the hardware aborts.
2. The transaction program ABORTS and does not contain specific coding to trap such errors.
3. The transaction program issues a call to TIPFCS (the TIP/30 File Control System) with a function code of FCS-TREN (a form of transaction commit).
4. The transaction program issues a call to TIPRTN (end of program).
5. The transaction program issues a call to TIPSUB, TIPXCTL, TIPDXC, TIPJUMP or TIPFORK (various transfers of control).
6. The transaction program solicits terminal input (via TIPMSGI, PROMPT, TIPTERM, etc.) without previously specifying that record locks are to be maintained across terminal input.

In cases (1) and (2), the system always rolls back any updates that were performed on files specifying HOLD=TR and releases all locks that were maintained for the program.

In cases (3) through (6), 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").

In general, transaction end causes the release of record locks and the writing of a "TREN" (mark transaction end) record to the TIP$B4 file, if records were updated in a file that is defined as HOLD=TR.

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

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

1.11.1. PIB-LOCK-INDICATOR Action

The following table summarizes the action of the TIP/30 system when it examines the field PIB-LOCK-INDICATOR.

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
R (release)	No	Released	Kept	No
H (hold)	No	Kept	Kept	No

Where:

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. The FCS-HOLD function is considered a "GETUP" lock for the purpose of this discussion.

UPDATE LOCK

A record lock that is currently imposed because the record has been updated by the program and the record is still held because the file is defined as HOLD=TR.

ROLLBACK UPDATES

Reversal of a modification of a record by the online transaction ROLLBACK (supplied by Allinson-Ross Corporation).

1.12. TIPABRT — Program ABORT Trap

When a user program aborts (with a program check such as a data exception), TIP/30 automatically invokes the Post Mortem Dump Analysis program (PMDA) on behalf of the program in error. The PMDA program produces a dump for analysis and invokes (abnormal) transaction end processing.

If the program needs to gain control when an error occurs (to take some counter action), the program must first issue call TIPABRT to establish an abnormal termination entry point (island code) for the program.

When a program establishes such an abnormal termination handler, PMDA will not be called and a program check when abort coding is activated is not considered an end of transaction point.

Note: *Only* **assembler language** *TIP/30 native mode programs may use this facility.*

Syntax:

```
CALL      TIPABRT,(savearea [,entry-addr])
```

Where:

savearea The label of an area (9 doublewords) where TIP/30 stores the PSW and registers 0 through 15 (inclusive, in that order) if an abnormal condition occurs.

entry-addr

The point to transfer control after an abnormal condition occurs. If this parameter is omitted, the entry address is assumed to be immediately following the savearea (savearea+72).

TIP/30 saves the PSW and registers 0 through 15 in the location specified by the first parameter and branches (via register 15) to either the address given as the second parameter or to SAVEAREA+72.

You may use R15 as a temporary cover register for the recovery code.

R1 is loaded with the address of the original parameter list passed to the transaction (PIB, CDA, MCS, WORK-AREA, GDA).

The programmer must keep in mind that the contents of registers other than 1 and 15 are *not defined* since the original program check may have occurred at any point during execution of the program.

Once the abort routine has started, any subsequent abnormal conditions result in loading PMDA unless the user program calls TIPABRT again to reestablish an abnormal termination entry point.

Example:

```
         CALL  TIPABRT,(ABTERM)  . TRAP PROGRAM CHECKS
                ...
                ...
                ...
ABTERM   DC    D'0'             . PSW AT TIME OF ERROR
ABREGS   DC    16F'0'           . REG 0-15 AT TIME OF ERROR
         USING *,R15
*        .....  ABNORMAL TERMINATION ROUTINE ENTRY POINT
```

Additional Considerations:

The save area must be at least fullword aligned.

1.13. 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 fullword (defined in COBOL as 9(9) COMP SYNC) with the corresponding bits in the fullword set to a zero or one (X'F0' or X'F1').

The bits in the fullword are numbered from 31 to 0 from **LEFT** to **RIGHT**.

Syntax:

```
CALL  'TIPBITS'   USING  BIT-SWITCHES
                         BYTE-SWITCHES
```

Where:

BIT-SWITCHES

The receiving field defined as a binary fullword — 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 graphic zero or one (X'F0' or X'F1').

Example:

```
MOVE '11001100110011001100110011001100' TO BYTE-SWITCHES.
CALL 'TIPBITS'  USING  BIT-SWITCHES
                       BYTE-SWITCHES
```

The field "BIT-SWITCHES" would contain:

Binary	'11001100110011001100110011001100'
Hex	'CCCCCCCC'

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

1.14. TIPBYTES — Convert Bits to Bytes

This subroutine is supplied as a utility for COBOL language programmers that need to manipulate bits. TIPBYTES converts a fullword (defined in COBOL as 9(9) COMP SYNC) 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 fullword.

The bits in a fullword are numbered from 31 to 0 from **LEFT to RIGHT.**

Syntax:

```
CALL  'TIPBYTES'  USING  BIT-SWITCHES
                         BYTE-SWITCHES
```

Where:

BIT-SWITCHES

The fullword field (defined as PIC 9(9) COMP SYNC) that contains the bits that 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 element named TC-BITS in the TIP library defines the two parameters in the above syntax description. See the description of the TIPFLAG subroutine.

1.15. TIPDATE — Return Date

This routine returns the date in a readable format. The actual characters returned depend on the TIP/30 generation keyword LANGUAGE=. English is the default language.

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
```

Additional Considerations:

TIP/30 always keeps the date current. If the OS/3 system is generated with DAYCHANGE=YES, the date changes at midnight and TIP/30's date also changes at this time.

1.16. 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'
```

Where:

There are no parameters.

Additional Considerations:

TIP/30 causes the dump by executing an illegal machine instruction (namely: X'00DEAD') on behalf of the transaction program. The Post Mortem Dump will indicate "OPERATION EXCEPTION"; this error however, is not an error that is exclusively caused by calls to TIPDUMP!

1.17. 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 lesser 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
  CALL  'TIPDXC'
```

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-NOT-FOUND	The program identified by the value in the field PIB-TRID is not defined in the TIP/30 catalogue, the load module could not be found, or there was insufficient memory to load the program.
PIB-SECURITY	The user running the calling program does not have sufficient security to run the requested program or the requested program is locked at this time of day.

Example:

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

WARNING

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 by ICAM. The ICAM interface that is used by TIP/30 does not permit an input message from a terminal if a previous input was not followed by at least one output message.

1.18. TIPFLAG — Flag Services

TIP/30 flag services provides user programs with the ability to manipulate up to 32 binary switches. These switches (flags) are stored as bits of a fullword within TIP/30 and may be accessed by any TIP/30 transaction program or by console operator commands (see 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.

This facility is similar to that provided by the hardware instruction TS (Test and Set). The TS instruction is commonly used to control a semaphore in a multi-tasked environment.

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.

Note: *The bits in a fullword are numbered from 31 to 0 from* LEFT to RIGHT.

Syntax:

```
CALL  'TIPFLAG'   USING   FUNCTION
                          MASK
                        [ RESULT ]
```

Where:

FUNCTION

A character code (X'F0' through X'F9') representing the function to be performed by TIPFLAG:

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.

In the above descriptions, "set" means the value 1; "clear" means the value 0.

MASK A binary fullword that identifies the flags to be acted on by this call to TIPFLAG. Each bit represents a flag. The bits of the fullword 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.

The result field is only used by function codes 8 and 9.

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 queueing for flag 13 in this case).

```
WORKING-STORAGE SECTION.
     ...
     COPY TC-FLAG OF TIP.
     ...
01   WORKAREA.
     ...
     COPY TC-BITS OF TIP.
     ...
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 until flag 13 was 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 queueing 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 fullword and thus obtain the proper bit pattern in the mask. In practice, COBOL makes it very awkward to move 10 digits to a binary fullword 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.
```

Additional Considerations:

The COBOL copy element **TC-FLAG** in the TIP library 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 element 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'.
```

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

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

```
*****************************************************************
*             DEFINE THE 32 SWITCHES USED BY TIPFLAG            *
*****************************************************************
*
     05  BIT-SWITCHES                    PICTURE 9(9)
         COMPUTATIONAL-4 SYNCHRONIZED.
*
     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'.
```

1.19. TIPFORK — Start Program at a Terminal

This call is used to start a program running on another terminal in the network as an independent, asynchronous process. The program that is started at another terminal runs independently of the initiating program.

The target terminal must be connected to the TIP/30 system[3] and must not be in use.

Before issuing this call, the calling program must:

- move the transaction-id of the program to be started to PIB-TRID
- move the name of the intended terminal to PIB-TID.

The calling program's CDA data is copied to the CDA of the next program for a length which is the lesser 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-TID
  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 Must be set to the name of the terminal where the program is to be started.

The reserved terminal names *BYP and *MST may be moved to the field PIB-TID to start a new process running on the bypass terminal or master terminal respectively. The bypass and master terminal is determined from information in the TIP/30 generation CLUSTER statements.

If TIPFORK is called with the field PIB-TID set to the caller's terminal name (if for example, the program forgot to move a value to the field!) or PIB-TID contains spaces or low values, the TIPFORK subroutine assumes that the program to be started is to be run in background (see "1.20. TIPFORK — Start Background Program" on page 1-46).

3. In a GLOBAL ICAM network, it must be defined as a static session or already have issued an appropriate $$SON or $$OPEN command.

Example:

```
*
*     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-NOT-FOUND	The program is not catalogued, or the load module could not be loaded. The terminal name does not exist, cannot be resolved, or the specified terminal is not session connected at this time.
PIB-LOCKED	There is already a program running on the specified terminal — this may be because a user is logged on the terminal, or the terminal is running an IMS transaction. If this error is returned, the program issuing the call to TIPFORK can issue a call to the TIP/30 subroutine TIPTIMER to delay a few seconds and then retry the call to TIPFORK.
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 TIPFORK will not receive control until the child process has started running unless an error is reported.

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

1.20. 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 number of potential Background programs is controlled by the TIP/30 generation parameter BACK=.

The calling program's CDA data is copied to the CDA of the next program for a length which is the lesser 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/30 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. This implies that a background program MAY use input redirection (See separate discussion of the topic: "1.35. Redirection and the Command Line" on page 1-76).

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.

Note: The field PIB-TID must not be modified before the call to this variation of TIPFORK. The TIPFORK subroutine uses the contents of the field PIB-TID to determine what type of TIPFORK is intended — if PIB-TID contains the terminal name of the program that is the caller (that is, the field was not altered to some other terminal name), the TIPFORK request is interpreted as a desire to start a background program.

Example:

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

Error Conditions:

PIB-NOT-FOUND	The program identified in the PIB-TRID is not defined in the TIP/30 Catalogue, or the load module could not be loaded. There are no available background process tables (see description of the TIP/30 generation parameter BACK=). The specified transaction may be defined in the TIP/30 Catalogue with the specification BACK=NO (meaning this transaction is not allowed to be run in background).
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 TIPFORK will not receive control until the child process has started running unless an error is reported.

1.21. 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 element (TC-GRPS in the TIP library):

```
*-----------------------------------------------------------------*
*                                                                 *
*  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 99 COMP SYNC.
        10  GRPS-ACTUAL           PICTURE 99 COMP SYNC.
        10  GRPS-NAMES.
            15  GRPS-NAME         PICTURE X(8) OCCURS 16 TIMES.
```

GRPS-MAX

A binary halfword 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 halfword 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.

Additional Considerations:

The TIP/30 system generation parameter NumGRPS= constrains the number of group names actually returned. No more than the configured maximum number of elective groups is returned.

1.22. 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 following copy element (TC-GRPS in the TIP library):

```
*-------------------------------------------------------------*
*                                                             *
*  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 99 COMP SYNC.
         10  GRPS-ACTUAL           PICTURE 99 COMP SYNC.
         10  GRPS-NAMES.
             15  GRPS-NAME         PICTURE X(8) OCCURS 16 TIMES.
```

GRPS-MAX

A binary halfword that is set by the calling program to a value between 1 and 16 (inclusive). The value placed in this field indicates the number of group names that are passed to the TIPGRPST subroutine by the calling program.

GRPS-ACTUAL

This field is not used by the TIPGRPST subroutine.

GRPS-NAME

An array of group names (only GRPS-MAX of these will be examined). This array corresponds (in one-to-one order) with the desired elective groups in the order of search.

Error Conditions:

PIB-NOT-FOUND	One or more of the suggested group names is not in the user's groupset(s).

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/30 catalogue.

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!

1.23. TIPJUMP — Direct Transfer Control

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-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.
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:

```
MOVE  'PHASE2' TO PIB-TRID.
CALL  'TIPJUMP'.
IF NOT PIB-GOOD
   PERFORM ERR-ON-JUMP.
```

1.24. TIPRTN — End Online Program

This call terminates an online TIP/30 program.

If the terminating program was running in background, TIP/30 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 which called the terminating program.

If the terminating program was executed from the TIP/30 command line, control returns to the TIP/30 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 lesser 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).

1.25. 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 directed to the system printer file (PRNTR).

Syntax:

```
CALL  'TIPSNAP'   USING   BGN-1  END-1
                        [ BGN-2  END-2 ]
                        [ BGN-3  END-3 ]
                        [ BGN-4  END-4 ]
```

Where:

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. Excessive use of TIPSNAP degrades system performance!

The PRNTR file is not automatically breakpointed by TIP/30 when a call is issued to TIPSNAP. To breakpoint the PRNTR for TIP/30, issue a breakpoint command for the TIP/30 job — represented by "xxxxxxx" in the examples below. This command may be entered on the system console, or the TIP/30 utility transaction BR may be used:

```
BR ACT,PRT,JOB=xxxxxxxx
```

If you are running on an OS/3 system that has spooling set to non-burst mode, it is also necessary to start a burst mode output writer to print the breakpointed print file:
PR BX,JOB=xxxxxxx

1.26. 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 64 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 which 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'
```

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.

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 lesser 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:

```
MOVE  SPACES  TO CDA.
MOVE  'PAYUP' TO PIB-TRID.

CALL  'TIPSUB'.
IF NOT PIB-GOOD
    PERFORM ERROR-ON-SUB.
```

Error Conditions:

PIB-NOT-FOUND	The program is not catalogued, 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.
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).

1.26.1. Calling TIP/30 Utilities

This section describes the procedures that must be followed when a user-written transaction program calls a utility transaction supplied with the TIP/30 system.

The transaction programs that are supplied with the TIP/30 system (such as: RV, FSE, TQL, TLIB, etc.) are written on the assumption that the programs are executed directly from the TIP/30 command line.[4]

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/30 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 element TC-CDA in the TIP library. Although the examples illustrate the use of TIPSUB to call the TIP/30 utilities, other methods of transferring control (TIPXCTL, TIPFORK, etc.) may be used if appropriate.

Example ①

This example illustrates calling the RV transaction to initiate a batch job. The batch job used for illustration would normally be started as follows:

```
TIP?▶RV TJ$COB74:TJ,,F=SYSGEN,E=MYPROG
```

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

```
MOVE SPACES          TO CDA.
MOVE 'RV'            TO PIB-TRID.
MOVE 'TJ$COB74:TJ,,F=SYSGEN,E=MYPROG'
                     TO CDA-TEXT.
CALL 'PARAM'  USING CDA-PARAMETERS
                    CDA-TEXT.
CALL 'TIPSUB'.
IF NOT PIB-GOOD ...
```

Note:

- The command line information — exactly as it would be typed — is literally moved to the CDA field CDA-TEXT
- The supplied subroutine PARAM[5] is then called to "parameterize" the data in CDA-TEXT field into the 8x8 byte buckets at CDA-PARAMETERS (the PARAM subroutine is exactly the same mechanism the TIP/30 Command Line Processor uses to parse the command line parameters)
- The program then calls TIPSUB to "perform" the RV transaction
- The RV transaction gets control (with the contents of the CDA correctly formatted) and performs its functions and returns control to the calling program.

4. There are some minor exceptions to this general statement.
5. This subroutine is documented in the MCS section of this manual.

Example ②

This example illustrates calling the MSG transaction to send a message to a terminal (the assumption is made that this terminal is one that is likely to be logged on TIP/30 — perhaps a terminal that is next to the operator console that is used primarily to run OCN).

To invoke MSG from the command line would normally require syntax such as:

```
TIP?▶MSG/TC11 Payroll update complete...
```

The user-written program calls the supplied utility MSG in this manner:

```
MOVE SPACES                          TO CDA.
MOVE 'MSG'                           TO PIB-TRID.
MOVE 'TC11'                          TO CDA-OPTIONS.
MOVE 'Payroll update complete...' TO CDA-TEXT.
CALL 'PARAM'  USING CDA-PARAMETERS
                    CDA-TEXT.
CALL 'TIPSUB'.
IF NOT PIB-GOOD ...
```

Note:

- In this example, data is explicitly moved to the CDA-OPTIONS field because the MSG utility transaction expects the destination terminal name to appear there
- The MSG utility expects the text of the message to be on the command line — it is, therefore, moved to CDA-TEXT
- The PARAM subroutine is called to be certain that data that appears in the CDA-TEXT field also appears (in parameterized form) in the CDA-PARAMETERS field.

Example ③

This example illustrates calling the FCLOSE transaction to issue file CLOSE requests. If FCLOSE was issued at the TIP/30 Command Line, it would be entered as follows:

```
TIP?▶FCLOSE *PAY,*AP,AUDITPAY
```

The user-written program calls the supplied utility FCLOSE in this manner:

```
MOVE SPACES                TO CDA.
MOVE 'FCLOSE'              TO PIB-TRID.
MOVE '*PAY,*AP,AUDITPAY' TO CDA-TEXT.
CALL 'PARAM'  USING CDA-PARAMETERS
                    CDA-TEXT.
CALL 'TIPSUB'.
IF NOT PIB-GOOD ...
```

Of course, the FOPEN utility transaction can be called in a similar manner to open files.

1.27. TIPSUBP — Call Resident Subroutine

This subroutine calls subprograms that have been separately compiled and linked. Only parameters identified by the calling program are passed to the subprogram.

Application systems often have subroutines that are used by a large number of the programs in the system. In the batch environment, these subroutines are compiled and included (linked) with each program which intends to call the subroutine. This scheme has two disadvantages (at least):

- if the subroutine is large, the size of each calling program is significantly increased by the size of the subroutine(s)
- if the subroutine requires compilation, all of the calling programs have to be re-linked (at most sites this means re-compilation too).

The TIP/30 environment allows such subroutines to be separately compiled and linked. These separate load modules can then be made resident and access to the routine is accomplished by calling TIPSUBP. TIP/30 handles the details of establishing proper calling linkage between the program and the subprogram.

Load modules that are called by using TIPSUBP must be made resident by using the RESIDENT= keyword in the TIP/30 job control option stream. Refer to the documentation of TIP/30 Generation, Maintenance and Installation.

There is negligible overhead associated with calling TIPSUBP. Basically all that is involved is a brief look up of the routine's address in TIP/30 internal tables and a direct branch to the code. If the called subroutine is not reentrant, callers are queued by TIP/30 to gain access to the subroutine.

Contrast this with the overhead involved in a TIPSUB call — it may be very beneficial to replace TIPSUB calls to a relatively small module with TIPSUBP calls to an equivalent separate subprogram. Care must be taken to carefully consider the parameters that must be passed to allow the subprogram to perform the desired functions.

Subprograms are defined in the TIP/30 Catalogue with a program name equal to the load module name. Since a subprogram is essentially an extension of the executable code for the calling program, the subprogram does NOT have any of the external work areas that a transaction program normally has: PIB, CDA, MCS-AREA, WORK-AREA, etc.

The TIP/30 Catalogue entry for a subprogram is required only to define the ability for the subprogram to be shared by multiple threads (USAGE=REENT or REUSE). If a catalogue entry for a subprogram is not found, TIP/30 assumes that the subprogram is reentrant.

The required keywords for the TIP/30 Catalogue Manager program (CAT) are:

SUBPROG=YES

To identify this entry as a subprogram entry.

USAGE= To declare the usage of the subprogram. Choices are: REENT (for a reentrant subprogram) or REUSE (for a serially reusable subprogram).

Example use of CAT to define a Subprogram:

```
TIP?▶CAT
CAT(1)?▶PROG TIP$Y$/CHKDGT00  LDM=CHKDGT SUBPROG=YES USAGE=REENT.
CAT(1)?▶END
```

The calling program must move the "load module" name to the field PIB-TRID to identify the desired subroutine. TIP/30 automatically assumes that the load module name has two trailing zeroes even if only six characters are supplied.

Syntax:

```
MOVE  '????????'  TO PIB-TRID
CALL  'TIPSUBP'   USING  param-1
                         ...
                         param-n
```

The parameters depend on the requirements of the subroutine. TIP/30 does not pass any parameters other than those identified on the CALL statement.

In particular, if the subprogram wishes to examine fields in the PIB, the program issuing the call to TIPSUBP must pass the PIB as a parameter.

Example:

```
MOVE  'CHKDGT'    TO     PIB-TRID
CALL  'TIPSUBP'   USING  PART-NUM
                         CHEK-DIGIT
                         PIB
IF  NOT  PIB-GOOD
    PERFORM ERROR-CALLING-CHKDGT.
```

In the example above, if the subprogram CHKDGT was written in COBOL, it would have the following general structure:

```
    ...
LINKAGE SECTION.
01  PART-NUMBER.
    ...
01  CHECK-DIGIT.
    ...
01  PIB. COPY TC-PIB OF TIP.
PROCEDURE DIVISION    USING  PART-NUMBER
                             CHECK-DIGIT
                             PIB.
    ...
    EXIT PROGRAM.
```

Error Conditions:

PIB-NOT-FOUND	The subprogram is not found as a resident load module. This may be because the value placed in PIB-TRID is a spelling error, the module name may not have been specified in the list of resident programs identified by the RESIDENT= keyword in the TIP/30 job control options, or there may have been a problem during TIP/30 initialization which prevented the proper loading of the subprogram.

Additional Considerations:

If the subprogram is not written as a reentrant routine then it must be defined in the TIP/30 catalogue as USAGE=REUSE (the default is assumed to be REENT).

The CATalogue manager utility transaction does not allow TYPE=TIP programs to be defined as USAGE=REUSE. In this case, you must specify SUBPROG=YES before specifying USAGE= to coerce the CAT program to accept USAGE=REUSE.

If the calling program is a TIP/30 native mode program, the subprogram may perform file I/O operations only by calling TIPFCS. If the calling program is an emulated IMS program the subprogram may perform file I/O operations only by issuing IMS style file I/O calls (CALL 'GET', etc.).

COBOL language subprograms return control using the "EXIT PROGRAM" COBOL statement. Assembler language subprograms return control using the "RETURN" macro.

The following table summarizes the resultant usage of an online program that uses a subprogram, that is written in COBOL:

Main	Subprogram	Result
Shared	Shared	Reload (TIP) Reuse (IMS)
Shared	Reent	Shared
Reent	Shared	Reload (TIP) Reuse (IMS)
Reent	Reent	Reent

Note: *In this table "Shared" refers to the result of specifying IMSCOD=YES for a COBOL program. The term "Reent" refers to the result of specifying IMSCOD=REN.*

WARNING

COBOL subprograms that are compiled with IMSCOD=REN option always assume that the reentrant control area that they require is located at the end of the transaction program's work area. This is true regardless of the type of transaction program that calls the subprogram. All programs that call TIPSUBP to invoke a COBOL subprogram compiled with IMSCOD=REN must add the appropriate number of bytes to their work area.

Note: *The TIP/30 system considers a subprogram as an extension of the instruction code of the calling program. As a result, subprograms may perform most of the TIP/30 calls that a "mainline" program may call with the exception that subprograms cannot perform any transfer of control (other than to return to the caller).*

Serial subprograms (those identified in the TIP/30 catalogue as USAGE=REUSE) may not solicit terminal input (since access to the serial subprogram is, by definition, a serial resource).

1.28. 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 fullword (PIC S9(9) COMP SYNC) that specifies the number of seconds that the issuing program wishes to wait. This parameter is required.

TIP/30 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/30 system is ready to run.

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

- CPU intensive activities
- prolonged periods of sequential file reading.

If the WAIT-TIME field is set to exactly **-1**, the TIPTIMER subroutine will use the value specified by the TIP/30 generation parameter TIMEOFF= (time to automatic logoff).

Note: *TIP/30 cannot provide TIMER services with accuracy better than one second. The program is delayed at least the number of seconds that is specified.*

A requested wait time of 1, 2 or 3 seconds by a **background** process is interpreted as 4 seconds (since background programs typically execute for long periods of time, the TIP/30 system discourages frequent calls to the TIPTIMER subroutine).

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

1. Obtain the current time of day from the operating system (the COBOL verb "ACCEPT" is handy for this).
2. Compute the number of seconds between the current time of day and the desired wake up time (taking into account possible day changes).
3. 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.

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/30 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/30 places the first available 12 bytes (converted to upper case) of the input message — if an input message is available (example: PIB-MSG-AVAIL status was returned).

The contents of this field are not defined if TIPTIMER returns a status of "PIB-TIMED-OUT".

Example:

```
05  TIMER-WAIT              PIC S9(9) COMP SYNC.
05  TIMER-STATUS            PIC X.
05  TIMER-PREVIEW           PIC X(12).
...
MOVE  +60           TO  TIMER-WAIT.
CALL  'TIPTIMER' USING  TIMER-WAIT
                        TIMER-STATUS
                        TIMER-PREVIEW.
```

In this example, TIP/30 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) and TIMER-PREVIEW contains the first 12 text characters (translated into upper case) of the input message.

Additional Considerations:

When the EOJ system shutdown command is issued, TIP/30 examines the queue of processes that are currently waiting for TIPTIMER completion. The time remaining is reduced (if necessary) so that it does not exceed the number of seconds implied by the TIP/30 generation parameter TIMEOFF=.

In effect, an EOJ truncates the outstanding wait time to a maximum of the TIMEOFF= generation value. This ensures that a program that is waking itself up once every hour (for example) will have an opportunity to, at least, wake up and observe that the system is trying to shutdown.

It is the responsibility of the program to examine the PIB-EOJ-PENDING flag in the PIB before going back to sleep, otherwise the program may delay system shutdown indefinitely.

Calling TIPTIMER does **not** cause the TIP/30 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/30 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/30 aborts the program with the reason code: "Resources locked, waiting TIPTIMER".

1.29. TIPUSR — Where is User

This subroutine is called to return the name of the terminal where a specified TIP/30 user is located. The subroutine searches for the specified TIP/30 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              PIC X(8).
    10  USER-TERM              PIC 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.

1.30. TIPUSRID — User Information

TIP/30 programs use this call to retrieve information about a specified TIP/30 userid. Information on the elective groups that the user belongs to and the comment from the user's TIP/30 catalogue record are returned.

Syntax:

```
CALL  'TIPUSRID'  USING  USER-ID-DATA
                         USER-ID
```

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 (if any) and the optional comment information that is in the user's TIP/30 catalogue entry.[6]

USER-ID An 8 character field containing the userid to be used in the search for information.

Example:

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

Error Conditions:

PIB-NOT-FOUND The specified userid is not found in the TIP/30 Catalogue. The result area is cleared to spaces when this error condition occurs.

6. The TIP/30 CAT transaction program permits the system administrator to enter this information via the CMT= keyword of the USER command.

Additional Considerations:

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

1.31. 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 lesser 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'
```

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'
IF NOT PIB-GOOD
   PERFORM ERR-ON-XCTL.
```

Error Conditions:

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 TIPXCTL.
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/30 system.

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

```
MOVE SPACES   TO CDA.
MOVE 'LOGOFF' TO PIB-TRID.
CALL 'TIPXCTL'.
IF NOT PIB-GOOD
   CALL 'TIPERASE'
   MOVE 'UNABLE TO LOGOFF'    TO ERROR-TEXT
   CALL 'ROLL'              USING ERROR-TEXT
   CALL 'TIPRTN'.
```

The code illustrated above is careful to check whether or not the transfer of control to the LOGOFF transaction was completed (if the TIPXCTL failed for any reason, the program will not just "fall off the end").

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.

Note: *The LOGOFF program is also catalogued as a transaction named "SOFF". If this variant of LOGOFF is called, the program can force a TIP/30 logoff and the auto transmit of a $$SOFF command to terminate a dynamic ICAM session.*

1.32. TIP/30 and IMS Interaction

In some situations it may be necessary to have a native mode TIP/30 program call an IMS program that is running under emulation or have an IMS emulated program call a TIP/30 native mode program.

IMS programs run under control of the TIP/30 IMS Emulator. The IMS Emulator is designed to "emulate" the IMS environment; it does **not** give IMS programs access to TIP/30 facilities.

IMS and TIP/30 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.

1.32.1. IMS to TIP/30

An IMS program may "succeed" (an IMS term) to a TIP/30 program by utilizing one of the following methods:

External succession

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/30 transaction identified in the SUCCESSOR-ID field is called.

Delayed Internal succession

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/30 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).*

Additional Considerations:

The IMS PIB field SUCCESSOR-ID is defined as a 6-byte field; the choice of TIP/30 transaction names is, therefore, limited to six characters when a TIP/30 program is called from an IMS program.

To get around this restriction define the TIP/30 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.

1.32.2. TIP/30 to IMS

A TIP/30 native mode program may call an IMS program by using the "TIPXCTL" or "TIPDXC" subroutines. The TIP/30 program must move the catalogued 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**, *in the TIP/30 catalogue. Catalogue an action with the load module name as the TIP/30 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/30 program, there will be nothing in the IMA — this will likely cause the IMS program to be fatally confused.

1.33. TIP/30 Command Line

The TIP/30 command line provides the facility to run a transaction as a background process, at a specified terminal.

To run a transaction in BACKGROUND, prefix the transaction name on the command line with a period:

```
TIP?▶.STATUS
```

In this example the STATUS program is started in background — the STATUS program recognizes that it has been run in background and reacts by printing a report. Of course, the transaction selected for background processing MUST be suitable for execution in background.

If the system does not have an available background process table (refer to TIP/30 generation parameter BACK=), or, if the transaction was explicitly prohibited from running in background,[7] the following error message is displayed:

```
Invalid transaction code
```

To run a transaction at another terminal (for example, to run a printing program on a bypass terminal), precede the transaction code with a period and immediately follow the transaction code with the name of the desired terminal enclosed in parentheses:

```
TIP?▶.TLIB(T109) PR JCS/TIP30,,AUX1
```

In this example TLIB runs on terminal "T109". The command line parameters to TLIB ("PR JCS/TIP30,,AUX1") appear as they normally would. In this case, "AUX1", therefore, means T109's AUX1 (since the TLIB program will be executing on T109).

7. If the transaction catalogue entry specifies BACK=NO.

To run a transaction on another terminal, the target terminal must be:

- Connected to the TIP/30 LOCAP if a Global ICAM is used — bypass terminals are often permanently connected to the TIP/30 LOCAP via a SESSION statement in ICAM since it may not be possible to issue a $$SON command on the bypass terminal;
- NOT marked "down" by ICAM;
- NOT logged on to TIP/30 or running any IMS emulated transaction.

If the target terminal does not meet the above restrictions, the following error messages are possible:

```
Invalid transaction code

Terminal busy or down
```

1.34. Redirected Input (.IN File)

TIP/30 provides a mechanism whereby line-oriented terminal input may be redirected to read data from a library element or an edit buffer.

This facility is analogous to the batch or command file capability provided in the Personal Computer world by MS-DOS ".BAT" files.

Each time a program requests line-oriented terminal input (via PROMPT, PARAM, etc.) the system reads the next available line of the specified element or edit buffer instead of soliciting input directly from the terminal.

In this way, an element or edit buffer can contain input that is processed line by line *as if it came directly from the terminal.*

The advantage of this scheme is the ability to create "command streams" to perform a series of operations. The command stream can proceed without keyboard intervention.

The library element or edit buffer that is involved in this process is often called a .IN (pronounced "dot-in") file.

Note: *TIP/30 programs that run in BACKGROUND are not permitted to solicit input* **from the terminal.** *However, it is perfectly legal, and often useful, for a background program to call line-oriented subroutines for input from a .IN file. The input comes from the .IN file (not from the physical terminal), therefore, TIP/30 does not abort the background program.*

1.35. Redirection and the Command Line

One way to take advantage of input redirection is via the TIP/30 command line. The TIP/30 command processor (TCP) recognizes the following syntax on the command line:

```
TIP?▶<lib/elt
```

OR

```
TIP?▶<bufrname
```

Use a a less-than character followed immediately by either a library/element specification or an edit buffer name to specify input redirection.

If you specify input redirection in conjunction with a transaction code (with or without accompanying parameters) precede the less-than character by a space.

If only one parameter appears following the "<", the parameter is assumed to be the name of an edit buffer (in the user's first elective group). If two parameters (separated by a slash or a comma) appear following the "<" they are interpreted as a library/element specification.

TCP ignores any other data following the input redirection specification; place input redirection after any transaction code and parameters specified for the transaction.

Example:

```
TIP?▶CAT <run/changes
```

```
TIP?▶<commands
```

The input library element or edit buffer is opened with a logical file name of ".IN". A file with that name in the user's Active File Table (AFT) is the signal for TIP/30 internal input routines to get input from the file rather than the terminal.

A TIP/30 native mode program is permitted to explicitly open a file with a logical file name of ".IN" (in the file name packet). Once such a file is open, subsequent requests for line-oriented input will use that file.

The TIP/30 system allows .IN files to be nested to a depth of three. Each time a file is opened with the logical file name ".IN", it is placed logically ahead of all other files in the AFT. This is the only occasion where duplicate logical file names are permitted in the AFT!

When a .IN file is open, all requests for line-oriented input (subroutines BREAK, PARAM, PROMPT, PROMPTX, PROMTPX8, TEXT, TEXT80) cause TIP/30 to read the next available line from the .IN file.

The program which requests the input is not aware that the input is obtained in this manner.

For example, assume that a library element "RUN/FOO" contains the following five lines:

```
WHOSON
SYS J
TLIB
PRINT JCS,TIP30
E
```

TIP/30 executes this element when the following command is entered on the TIP/30 command line:

```
TIP?►<run/foo
```

The result is:

- Execute WHOSON
- Execute SYS with parameter 1 set to "J"
- Execute TLIB
- TLIB "reads" and performs the "print" command
- TLIB "reads" and performs the "e" command (causing TLIB to terminate).

There is a minor problem with the library/element as shown above: when all five lines are processed, TCP "stalls" because it receives "end of file" on the .IN file. TCP closes the .IN file, but the terminal user must press the [MSG WAIT] key, a function key or the [XMIT] key to "wake up" TCP so that it issues the standard TIP/30 prompt.

To avoid this problem, a special convention has been established for .IN files that are read by TCP; if TCP reads a line of input from a .IN file that begins with a dollar sign character, TCP processes the line[8] and then closes the .IN file and resumes normal operation.

8. The dollar sign is ignored while processing the command line.

Thus, the dollar sign convention can be used to signal the last operational line of a .IN file that is used by TCP.

A common technique is to invoke the NOTE transaction as the last line of a .IN file:

```
TLIB
PR SRC/PAY010
PR SRC/PAY020
PR SRC/PAY030
END
$NOTE all done!
```

This example illustrates the main reason for the existence of the NOTE transaction. Use the NOTE transaction throughout command files to monitor the execution progress of the command file.

A variation of the use of redirection on the command line is to specify both a transaction name to execute and a .IN file specification:

```
TIP?▶TLIB <foo
```

Where:

Edit buffer "foo" contains a number of TLIB commands:

```
DELETE SRC/PAY020
COPY TST/PAYX20,,SRC/PAY020
PRINT SRC/PAY020
E
```

Note: The "$" convention is not required here because TLIB, **not** *TCP, is reading the .IN file.*

A common pitfall, avoided in the example above, is the potential appearance of an unexpected prompt. If the DELETE command was missing from the above example, TLIB might discover during the COPY command that element SRC/PAY020 already exists and dutifully display on the terminal:

```
Element currently exists - overwrite? ▶Yes  ▶No
```

TLIB however, is unaware that a .IN file is in use, so it gobbles up the next line as the answer to the unexpected prompt and of course gets quite off track!

WARNING

Carefully consider how to use such command files to achieve the desired results.

1.36. Input Redirection at LOGON

Another potential use of input redirection is at the time a TIP/30 user logs on the TIP/30 system. The TIP/30 LOGON program arbitrarily attempts to open a file called ".IN" after the user has logged on.

Opening the .IN file follows the standard order of search:

- userid
- elective groups (GRP1, GRP2, etc)
- TIPY.

To establish a redirected input file to be used at LOGON, create an edit buffer with the buffer name of ".IN" (in the appropriate group or groups) to "auto start" the user or groups of users.

Example:

```
FSE ,,,EDP/.IN
```

Where:

FSE creates an edit buffer named ".IN" (which is applicable to all users in the group "EDP"). Such an edit buffer which might contain, for example:

```
NOTE *
NOTE *      Todays News:
NOTE *      ============
NOTE *      ... ... ... ...
NOTE *      ... ... ... ...
NOTE *
NOTE,W     Press MSG WAIT to continue
$MAIL,?
```

WARNING

Once the edit buffer is created it is immediately used by the LOGON program for subsequent logons. Similarly, while a user is logging on, the edit buffer may be "in use" for a potentially long period of time and may be difficult to modify or correct.

Use extreme caution before unleashing .IN files — it is easy to create a situation that gets out of control.

Create all such edit buffers in a private group (your userid is a good choice) and test them before final implementation.

Note: *In situations where a command file or .IN file has "run amok", one possible course of action is to wait for a terminal prompt (or attempt to cause a prompt by pressing the* [MSG WAIT] *key) and then make use of the fact that TIP/30 monitors all input messages for the "magic" string* **@@DIE**

If the first 5 characters of any input message exactly match[9] *the string* **@@DIE** *TIP/30 will abort the program that is running.*

An important side effect (the key point in this context) is that the recognition of @@DIE causes TIP/30 to close all .IN files that are open for that terminal!

9. The characters "DIE" must be upper case letters!

1.37. TIP/30 RPG II Support

The TIP/30 Message Control System (MCS) supports:

- input-only fields
- output-only fields
- input/output fields.

MCS also handles RPG indicators to control screen formatting.

The screen format definition program (TFD) allows the use of RPG indicators to control the fields on the screen. If the corresponding indicator is on, the field is displayed using the error field colour; if the corresponding indicator is off, the field is not displayed.

The MSGAR utility transaction (message archiver) provides commands which produce RPG input or output specifications from a screen format and a command that controls whether the program may override the starting row for the screen format:

MSGAR RPGIN xxx

Create a source module containing the RPG input specifications ("I" statements) for the specified screen format

MSGAR RPGOUT xxx

Create a source module containing the RPG output specifications ("O" statements) for the specified screen format

The MSGAR ALTRON command sets a flag that indicates that the program controls the starting row of the screen format. An RPG program uses RLABL SLNO to activate this feature and EXIT RPGSLN to load the current starting row.

1.37.1. RPG II Exit Routines

A TIP/30 RPG II program may EXIT to any of the following routines:

RPGCLR Sets the MCS data area to spaces.

RPGCUR Define MCS Cursor modifications. This exit is used in conjunction with exit RPGFCC. The TIP/30 RPG program must issue an EXIT for this label before any screen output occurs.

RPGDMP CALLs TIPDUMP.

RPGERA CALLs TIPERASE.

RPGFCC Define MCS Field (FCC) modifications. This exit is used in conjunction with exit RPGCUR. The TIP/30 RPG program must issue an EXIT for this label before any screen output occurs.

The RPG II program must define two areas called TIPFCC and TIPCUR.

```
ISVDS          DS
I                                1    8 TIPFCC
I                                1    2 FLD01F
I                                3    4 FLD02F
I                                5    6 FLD03F
I                                7    8 FLD04F
I                                9   12 TIPCUR
I                                9    9 FLD01C
I                               10   10 FLD02C
I                               11   11 FLD03C
I                               12   12 FLD04C
```

For further information about using FCC and Cursor modification, see "2.4.7. FCC Modifications" on page 2-26 and "2.4.8. Cursor Positioning" on page 2-29.

RPGFRK CALLs TIPFORK.

The field PIB-TRID must contain the name of the program being called.

RPGJMP CALLs TIPJUMP.

The field PIB-TRID must contain the name of the program being called.

RPGMSE Set the next screen output to be a CALL to TIPMSGE, using the data in the field TIPERR. The next screen output calls TIPMSGE (using TIPERR) instead of the usual call to TIPMSGO.

RLABL TIPERR is required. The length of TIPERR is determined by the screen formats used in the program.

RPGMEO Set the next screen output to be a CALL to TIPMSGEO, using the data in the field TIPERR. The next screen output calls TIPMSGEO (using TIPERR) followed by a call to TIPMSGO.

RLABL TIPERR is required. The length of TIPERR is determined by the screen formats used in the program.

RPGPFL Flushes a TIPPRINT buffer.

RLABL TIPPRT is required and is defined as follows:

```
ISVDS          DS
I                                                1 102 TIPPRT
I                                                1   8 IPLFD
```

The name of the print file whose buffer is flushed (from the RPG II "F" specification).

The buffer is only flushed if output is going to an auxiliary printer or to an MS-DOS file AND the program has 1 or more indexed files in sequential mode (SETLL).

If a flush is needed, any files in sequential mode are put into random mode (ESETL). You must appropriately SETLL the files following this exit.

RPGPIN Modifies a TIPPRINT information packet or changes the output destination of a print file.

Do not use "1P" headings with this EXIT (see the sample print program (TT$RPP in the TIP library) for an example and description of proper technique).

RLABL TIPPRT is required and is defined as follows:

```
ISVDS          DS
I                                      1 102 TIPPRT
I                                      1   8 IPLFD
I                                      9  16 IPLFN
I                                     17 102 IPINFO
I                                     19  20 IPPGLN
I                                     21  24 IPTERM
I                                     25  25 IPTOF
I                                     26  26 IPLF
I                                     27  27 IPNOWP
I                                     28  28 IPUPPR
```

IPLFD The name of the print file from the RPG II "F" specification.

IPLFN The new destination of the print output. Any valid TIPPRINT destination may be specified.

IPINFO The TIPPRINT information packet (some of the more popular fields of that packet have been defined following this field).

RPGSLN Copies SLNO (2 byte packed field) to PIB-ALT-ROW.

RLABL SLNO is required.

Note: *Perform this EXIT only once during the program's initial input cycle — SLNO is zeroed during the EXIT.*

RPGSUB CALLs TIPSUB.

The field PIB-TRID must contain the name of the program to call.

RPGTMR CALLs TIPTIMER.

The PIB-WAIT-TIME field must contain the number of seconds to wait.

RPGXCT CALLs TIPXCTL.

The field PIB-TRID must contain the name of the program to call.

1.37.2. Cataloguing TIP/30 RPG Programs

The following TIP/30 catalogue keywords are significant for RPG II programs. These keywords are also documented in the description of the TIP/30 Catalogue Manager (CAT) utility transaction.

USAGE=RELOAD

Required for TIP/30 RPG II programs.

DEBUG=YES

Highly recommended.

S34=YES Specify this keyword as shown if the RPG II program was written to expect data on the screen to be transmitted when a function key is pressed.

The TIP/30 RPG interface routines simulate this capability by reading the screen when a function key is pressed.[10]

CDA=n As required by the program.

MCS=n As required for the data of largest screen used by the program. Add 32 bytes to the value the program requires.

FILES= As required by program.

WORK=n Allocate at least 1024 bytes.

Allocate an additional 3000 bytes for each printer file defined in the RPG II program.

Allocate additional work area space equal to the size of the largest record updated by the program.

10. This represents some overhead, but may be necessary to properly support programs from other vendors.

Section 2
Message Control System (MCS)

This chapter describes the facilities provided by TIP/30 to enable an online program to perform input and output operations at a terminal.

Three levels of interface are provided:

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.

Note: *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.*

2.1. Message Control System (MCS)

The TIP/30 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:

1. TFD/TFU — Utility transaction to define and update screen formats.
2. MSGSHOW — Utility transaction to test screen formats.
3. MSGAR — Utility transaction that provides librarian services for screen formats.

The fourth component of MCS is the Message Formatter (MSGFMT), an internal part of TIP/30 that provides an interface between the formats and the data supplied by the program. MSGFMT is the TIP/30 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.

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.

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.

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

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/30 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.

The following diagrams illustrate the relationship between the program and the TIP/30 screen format. Further information about how a screen format is defined may be found in the description of the utility transaction TFD (TIP/30 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









EEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEE
```

The first line is heading information; the strings of "U" characters define upper case 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 define 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:[11]

```
05   S-NAME        PIC X(25).
05   S-ADDR-1      PIC X(25).
05   S-ADDR-2      PIC X(25).
05   S-ADDR-3      PIC X(25).
05   S-BALANCE     PIC 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/30 MCS subroutine "TIPMSGO" to output the screen format and the data supplied by the program.

Conversely, a call to the TIP/30 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.

11. Error fields are not considered data fields in this context and are handled separately.

2.1.1. MCS Reserved Terminal Names

Output messages are normally sent to the terminal that is executing the transaction program. Logical terminal clusters may be defined in the TIP/30 generation parameters to define a group of terminals that have similar characteristics (see the description of the CLUSTER generation statement in *TIP/30 Generation, Maintenance and Installation — ARP-600-05*). This capability can be quite useful since many new terminal systems are designed as clusters.

If the program sends a output message to the reserved terminal name "***MST**", TIP/30 directs the output to the master terminal of the cluster to which the sending terminal belongs.

Similarly, specifying the reserved terminal name "***BYP**" directs the message to the terminal defined as the **BYPASS** terminal for the cluster. If no cluster definition or bypass/master terminal is given, the message is sent to the originating terminal.

Example:

Assume that the following terminal cluster is defined in the TIP/30 generation parameters:

```
CLUSTER T101 SLAVES=(T102,T103) BYPASS=T104.
```

This CLUSTER statement defines a group of terminals (T101, T102, T103) to have a common "bypass" terminal, namely: T104. Terminal T104 need not be physically related to any of the other terminals — in fact, it may be located anywhere in the network! Terminal T101 is defined as the master terminal of the cluster.

Programs executing on terminals T102 and T103 may use the reserved terminal name *MST to refer to terminal T101. Similarly, terminals T101, T102 and T103 may use the reserved terminal name *BYP to refer to terminal T104.

The advantage of using the reserved names and defining clusters is to permit programs to refer to another related terminal in a way which does not require terminal names to be directly coded in the program. This technique allows such programs to execute correctly anywhere in the network (provided the CLUSTER statements are correctly specified).

Commercial application packages take advantage of this scheme to offer ease of use and flexibility to users of the package.

2.1.2. UNISCOPE Terminal Control Page

The automatic setting of the control page XMIT setting may be controlled at the program, cluster or system level. When TIP/30 schedules an online program the control page is set to a particular value according to the following order of precedence:

[1] Specification of XMIT= in the PROG statement in the TIP/30 catalogue entry for the program

[2] Specification of XMIT= in a CLUSTER statement containing the terminal

[3] The XMIT= specification in the TIPGEN section of the TIP/30 generation parameters.

The TIP/30 Command line processor sets the control page to VAR if the last program requested the control page set to CHAN (due to differences in the way various terminal types implement "CHAN", this mode is not suitable for most work and is not recommended).

The TIP/30 LOGON program sets the control page to the XMIT= value specified in the TIPGEN section of the TIP/30 generation parameters.

Note: *TIP/30 does not always set the control page every time a program is run; it sets the control page only if the desired setting is different from the* last *setting made by TIP/30.*

For example, if a terminal operator locally sets the control page to "ALL" and then runs a program that is catalogued as XMIT=VAR, TIP/30 is not aware that the control page was altered manually; it therefore decides against setting the control page.

2.1.3. Down Line Loaded Screen Formats

An advantage of terminal clustering is that the controlling (master) terminal may be programmable (UTS400 terminals). This feature may be used to improve terminal I/O throughput. DLL is an online utility program supplied with TIP/30 that is used to download screen formats into UTS400 master terminals.

When a set of formats are down loaded, MCS keeps track of their names. TIP/30 maintains a table of down loaded format names for each defined cluster.

Whenever an online program requests that a screen format be sent to a terminal, TIP/30 checks the format table to determine whether the format has already been down loaded.

If the requested screen format is down loaded, TIP/30 sends a short message to the terminal to retrieve the format from the terminal's internal memory, followed by the data.

This technique greatly reduces transmit time. The user may change the set of displays at any time using the utility transaction program DLL and all subsequent terminal I/O adjusts accordingly.

Currently, only UTS-400 terminals with user programmable memory may use this facility.

2.2. MCS Interface Overview

An online TIP/30 native mode program uses TIP/30 screen formats by issuing subroutine calls to the TIP/30 Message Control System to transfer data to and from the terminal.

The subroutines are summarized as follows:

TIPERASE	Erase screen.
TIPMSGE	Send "error" message to terminal.
TIPMSGEO	Define a deferred error message.
TIPMSGI	Input data from terminal.
TIPMSGO	Output data to terminal.
TIPMSGRV	Force read terminal screen.

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.

Note: *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.

The following code fragment illustrates the general structure of a TIP/30 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.
        --evaluate input data--
        IF ANY-ERRORS-DETECTED
           MOVE ERROR TEXT TO ERROR-MESSAGE-TEXT
           CALL 'TIPMSGE' USING ...
           GO TO GET-INPUT.
        --update information on file etc--
        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.

The TIP library includes the COBOL source for a working TIP/30 demonstration program (element name TT-SP1). This program implements a simple on-screen calculator. The program uses a TIP/30 screen format as the interface with the terminal operator. The transaction code to execute this sample program is TSPCALC.

2.3. MCS Interface Packet

The COBOL copy element 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/30 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.

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.

```
       01  MCS-AREA.  COPY  TC-MCS OF TIP.
*
*        TIP/30  -  MESSAGE  CONTROL  SYSTEM  PACKET
*
     02  MCS-NAME                        PICTURE X(8).
     02  MCS-TERM                        PICTURE X(4).
     02  MCS-FUNCTION                    PICTURE X.
         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 SYNC.
     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-F-REBUILD               VALUE '1' '5' 'N'.
           88  MCS-F-NEXT                  VALUE '2' '6'.
           88  MCS-F-UPDATE                VALUE '4' '8'.
       02  MCS-FILLER                    PICTURE X.
           88  MCS-UNDERLINE               VALUE '_'.
           88  MCS-ASTERISK                VALUE '*'.
       02  MCS-COUNT                     PICTURE S9(4) COMP-4 SYNC.
/
       02  MCS-DATA.
*
*      USER SUPPLIED RECORD LAYOUT FOR MCS SCREEN FOLLOWS HERE
*
```

Where:

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/TFU 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 Catalogue record in the TIP/30 Catalogue), 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).

Default: 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/30 (in global ICAM network), the terminal name is ignored.

The reserved terminal names ***MST** and ***BYP** are specified to indicate (respectively) that the intended terminal is the MASTER or BYPASS terminal (as defined in the CLUSTER definition for the issuing terminal).[12]

12. See the TIP/30 "CLUSTER" generation statement for additional information about defining terminal clusters.

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.

To maintain downward compatibility with older TIP/30 releases, this field may be set to the value "H" to indicate that a subsequent TIPMSGI should **NOT** release any current record update locks.

The programmer is advised to maintain record locks using the field "PIB-LOCK-INDICATOR" in the TIP/30 Program Information Block (PIB). See the description of this field in the PCS section of this manual.

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".

If auxiliary device printing is requested via TIPMSGO, this field is set to the ICAM delivery status. See the section on "Auxiliary device I/O" for a list of the possible status codes.

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.

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 data was detected.

For example, if the terminal operator enters a partial value in a long field and presses XMT 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-COUNT is set to zero after a CALL to TIPMSGE.

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.

2.4. MCS Subroutine CALLS

2.4.1. TIPMSGO — Output Data to Screen Format

MCS provides the TIPMSGO subroutine to display a TIP/30 screen format (with or without) accompanying data. Since this subroutine call is normally the first interaction between the program and TIP/30 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 catalogue 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 format on a terminal other than the terminal running the program.

Note: *Only screen OUTPUT may be redirected in this manner — terminal input must always occur at the terminal that is running the program.*

MCS-FUNCTION

Before issuing a call to TIPMSGO, the program may specify one of a number of function codes in this field:

Note: *When "D" is specified in MCS-FUNCTION, data fields which 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.*

Table 2-1. MCS-FUNCTION Values for TIPMSGO

Value	Description
space	Transmit the entire screen format (both headings and data).
D	Transmit data only (not the heading information).
M	Send the output screen format as an unsolicited message (sends data and heading information).
P	Output screen format with a "print transparent" 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 transparent" 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" for some fields[13] MCS places the default data in the fields when the field is located beyond the end of the data supplied in MCS-DATA — according to the value of MCS-COUNT.

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.

Syntax:

```
CALL   'TIPMSGO'   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 "2.4.7. FCC Modifications" on page 2-26 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 "2.4.8. Cursor Positioning" on page 2-29 for details.

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.

13. See description of default data in the documentation for the TFD utility transaction program.

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 FORMAT: XXXXXXXX
```

Where XXXXXXXX is the data that was found in the field MCS-NAME.

2.4.2. 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/30 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.

Before issuing a call to TIPMSGI, the program must ensure that the MCS interface packet contains appropriate values in a number of the 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 terminal 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.

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/30 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.

Syntax:

```
CALL   'TIPMSGI'   USING   MCS
```

Where:

MCS The MCS interface packet as previously described.

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, including MSG WAIT is pressed.

Error Conditions:

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/30 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.

A program may not have access to a serial resource when a call to TIPMSGI is issued. A serial resource is a resource which, by nature, is available to only one online transaction program at a time. Examples of a serial resource are:

- a file that has been placed in sequential mode (by a call to TIPFCS function FCS-SETL and its variants)
- Imparts to the DMS data base.

Only one online program is allowed to have a particular file in sequential mode at any instant (unless the file is defined in the TIP/30 generation parameters as MULTISEQ=YES — setting a MULTISEQ=YES file in sequential mode is not considered a serial resource).

TIP/30 cannot allow a transaction program to acquire such a serial resource and then wait (perhaps forever) for terminal input. This scenario might cause horrendous delays in the system and may even cause deadlock conditions (deadlock occurs when two independent processes are waiting on each other — each process has exclusive use of the resource the other wants to access).

Programs which have a serial resource acquired when calling TIPMSGI are aborted by TIP/30 with a reason code of:

```
Resources locked; waiting input
```

Additional Considerations:

Well-behaved programs should periodically check (by examining the PIB field PIB-SYSTEM-FLAG) for a pending TIP/30 system shutdown. In general, a good place to check for this condition is after a call to TIPMSGI in the main processing loop of the program. By doing so, the program at least ensures that no new work is started after system shutdown has been requested. Another good technique is to place an upper limit on the wait time for all input messages and check for PIB-EOJ-PENDING whenever a request for input causes a timeout (the terminal operator may have abandoned the terminal).

2.4.3. 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:

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

To deliver error message text, the program passes a parameter which 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". On terminals that do not have the capability to blink data fields, for example the U200, a start-blink character (X'1C') is sent to the screen position immediately preceding the data field.

Blinking fields in error should be used with discretion on terminals like the U200 because the X'1C' character that is output by MCS may destroy data in the screen format.

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).

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 "2.4.7. FCC Modifications" on page 2-26 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 "2.4.8. Cursor Positioning" on page 2-29 for details.

Example:

```
05  ERROR-TEXT                          PIC 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
```

Additional Considerations:

TIP/30 sets MCS-COUNT to zero after a call to the TIPMSGE subroutine.

OS/3 programming languages have no provision for the omission of parameters (other than trailing parameters) on the CALL statement. It is not possible to avoid specifying FCC-MODS if the CURSOR-MODS parameter is specified.

2.4.4. 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 *address* of the TEXT area and uses this address 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.

2.4.5. 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/30 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) COMP) 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.

Additional Considerations:

Use of this function virtually doubles the traffic on a line; use this function with discretion.

It is often preferable to send the terminal an error message (for example, **Cursor Incorrectly Placed!**) and to then instruct the operator to reposition the cursor and press XMIT again.

2.4.6. 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.

Syntax:

```
CALL   'TIPERASE'
```

Where:

There are no parameters.

Additional Considerations:

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

Example:

```
...
CALL 'TIPMSGI' USING MCS.
IF MCS-FKEY4
   CALL 'TIPERASE'
   CALL 'TIPRTN'.
```

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

2.4.7. 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:

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.

Include the supplied COBOL copy element (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
*
*
***** FOLLOWING VALUES MAY BE USED TO JUST CHANGE PROTECTION
*
     05  FCC-PROTECT                      PICTURE XX VALUE '.P'.
     05  FCC-UNPROTECT                    PICTURE XX VALUE '.U'.
     05  FCC-BLINK                        PICTURE XX VALUE '.B'.
```

Example:

Assume that the screen format has 3 fields: a name, an address and a credit limit:

```
05  SCREEN-NAME          PIC X(40).
05  SCREEN-ADDR          PIC X(40).
05  SCREEN-CRLIMIT       PIC 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        PIC X(2).
  10  FCC-MOD-ADDR        PIC X(2).
  10  FCC-MOD-CRLIMIT     PIC 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 copy element 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 copy element (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.

2.4.8. 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.

2.5. 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/30 Message Control System (MCS), that was discussed in the previous section.

A native mode TIP/30 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.

Table 2-2. Line Oriented Subroutine Summary

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.
TIPCOP	Print screen on auxiliary printer.
TIPCPAGE	Alter terminal control page XMIT setting.
TIPSCAN	Unstring a character string according to specified delimiters.

2.5.1. Function Key Input

When a function key or [MSG WAIT] is pressed, *absolutely no data is transmitted from the terminal.* ICAM receives a signal that a particular function key has been pressed.

To allow programs to properly process function keys, TIP/30 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/30 into the pseudo function key [F23].

A utility transaction named "POC" may be defined to handle the arrival of a POC signal when the terminal has output a TIP/30 command line prompt. For more information see the description of the POC utility transaction.

2.5.2. 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 output[14]

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/30 "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 "2.5.3. PARAM — Parameterize Data" on page 2-34.

See "2.5.1. Function Key Input" on page 2-31 for a description of how function keys are returned.

14. See "2.5.7. ROLL — Output Line & Roll Screen" on page 2-40.

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 "2.5.1. Function Key Input" on page 2-31).

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.

2.5.3. 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:

- a comma
- a slash
- a single space
- multiple consecutive spaces
- an 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 upper case
- left justified
- space padded on the right to a maximum of 8 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 "2.5.1. Function Key Input" on page 2-31 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):

Table 2-3. Examples of Parameterization

TEXT-AREA	PARAM-AREA
"DR. John Smith III"	DR. JOHN SMITH III
"TSPUPDT 123/x PRINT"	TSPUPDT 00000123 X PRINT
"ABCDEFGHIJKLMNOPQRST"	ABCDEFGH JKLMNOPQ ST

Note: *As illustrated in the final example in the preceding table, if a valid delimiter is not encountered, PARAM assumes the next character is a delimiter and discards it. Programs must not rely on the presence or absence of this "feature".*

2.5.4. 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:

```
VTOC(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 upper case.

See "2.5.1. Function Key Input" on page 2-31 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.

Note: *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:

① `"\YES  \NO  " (exactly 11 characters)`

② `"\NO  \YES  " (exactly 11 characters)`

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 (upper case!) 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.
    ...
    05  REPLY-AREA      PICTURE X(64).
    ...
 PROCEDURE DIVISION.
    ...
    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).

2.5.5. 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 upper case alphabetic translation.

Syntax:

```
CALL  'PROMPTX'  USING   TEXT-AREA
                       [ PROMPT-STR ]
```

Where:

TEXT-AREA

The 64 byte area where the unparameterized terminal input is placed.

See "2.5.1. Function Key Input" on page 2-31 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:

Refer to "2.5.4. PROMPT — Prompt Terminal for Reply" on page 2-36 for additional details.

2.5.6. 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 upper case alphabetic translation.

Syntax:

```
CALL  'PROMPTX8'  USING   TEXT-AREA
                        [ PROMPT-STR ]
```

Where:

TEXT-AREA

The 80 byte area where the unparameterized terminal input is placed.

See "2.5.1. Function Key Input" on page 2-31 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:

Refer to "2.5.4. PROMPT — Prompt Terminal for Reply" on page 2-36 for details.

2.5.7. 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 upper case 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                          PIC X(80)
    VALUE '     Amount         Tax           Total'.
...
LINKAGE SECTION.
...
05  DETL-LINE.
    10  DETL-AMT                      PIC   ZZZ,ZZ9.99.
    10  FILLER                        PIC         X(3).
    10  DETL-TAX                      PIC     Z,ZZ9.99.
    10  FILLER                        PIC         X(3).
    10  DETL-TOTAL                    PIC Z,ZZZ,ZZ9.99.
    10  FILLER                        PIC        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). Refer to 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, the text is output on the system console.

2.5.8. 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  ROLL-POINT   PIC S9(2)  COMP SYNC   VALUE 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.

2.5.9. 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 that 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 upper case.

Syntax:

```
CALL  'TEXT'  USING  TEXT-AREA
```

Where:

TEXT-AREA

The 64 byte area where the terminal input is to be placed.

See "2.5.1. Function Key Input" on page 2-31 for a description of how function keys are returned.

Example:

```
05  TEXT-AREA              PIC X(64).
```

2.5.10. 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 upper case.

Syntax:

```
CALL  'TEXT80'  USING  TEXT-AREA
```

Where:

TEXT-AREA

An 80 byte area where the terminal input is to be placed (without parameterization).

Note: *This field must be defined as 80 bytes, but no more than 72 bytes of terminal data are returned.*

See "2.5.1. Function Key Input" on page 2-31 for a description of how function keys are returned.

Example:

```
05  TEXT-AREA                  PIC X(80).
```

2.5.11. TIPCOP — Print Screen on Aux Printer

To simplify the handling of an auxiliary printer, the program may call TIPCOP to send a PRINT command to a selected terminal. TIPCOP places the cursor at the last column of the row specified in the call. A second optional parameter is the terminal name to be used.

Syntax:

```
CALL  'TIPCOP'  USING  ROW
                       [ TERM-NAME ]
```

Where:

ROW — The row number of the last line to be printed on the auxiliary printer. This field must be a binary halfword. The proper output character(s) that cause the terminal to PRINT on the auxiliary printer will be placed immediately following this row number.

If this value is not a valid row number, the TIPCOP subroutine assumes the last row of the terminal.

TERM-NAME

Optional parameter that supplies the name of the destination terminal. Default: the terminal that is calling 'TIPCOP'.

Example:

```
05  ROW         PIC 9(2)  COMP  SYNC.
05  TERM-NAME   PIC X(4).
...
CALL 'TIPMSGO' USING MCS.
MOVE 24 TO ROW.
CALL 'TIPCOP' USING ROW.
```

Additional Considerations:

This subroutine is a brute force way to accomplish terminal printing. The TIPPRINT subroutine interface provides much more flexibility and is preferable for volume printing.

2.5.12. TIPCPAGE — Set Control Page

An online program may set the control page XMIT (Transmit) Field of a UTS-400[15] terminal by calling this subroutine with the choice of transmit option.

Syntax:

```
CALL  'TIPCPAGE'  USING  CPAGE-OPTION
```

Where:

CPAGE-OPTION

A four character field indicating the desired transmit option:

ALL	Transmit all (both protected and unprotected areas).
VAR	Transmit variable (unprotected data fields only).
CHAN	Transmit only changed data fields.

Example:

```
77  CPAGE-OPTION      PIC X(4)    VALUE 'VAR'.
```

Additional Considerations:

The CHAN option appears attractive, but a number of terminals do not set the "changed" attribute properly under some circumstances (for example, when a field is erased using the "erase EOL" key). Since the terminal is not reacting properly, the program must make allowances for this type of scenario (*easier said than done*).

15. Or any terminal with equivalent control page.

2.5.13. TIPSCAN — Scan String For Parameters

This subroutine facilitates parsing (scanning) fields from a string of characters. It utilizes delimiters that are specified by the calling routine.

The following example places the first field which may be up to eight bytes long and ends with any of the specified delimiters (a comma, a slash or a space), into OSTRING-TXT.

After the call to TIPSCAN, the field "IPTR" is set to the zero relative offset into "ISTRING" of the next character after the delimiter in preparation for another call to TIPSCAN.

Repeated calls to TIPSCAN scan out all such fields. The program must check the value of IPTR after each call to determine when the end of the field "ISTRING" is reached.

Syntax:

```
CALL  'TIPSCAN'  USING  ISTRING
                        IPTR
                        OSTRING
                        DELIM-TBL
```

Example:

```
05  ISTRING                PIC X(??).
05  IPTR                   PIC 9(3) COMP SYNC.
05  OSTRING.
    10  OSTRING-LEN        PIC 9(3) COMP SYNC.
    10  OSTRING-TXT        PIC X(8).
05  DELIM-TBL.
    10  DELIM-COUNT        PIC 9(4) COMP SYNC.
    10  DELIMS             PIC X(3).
    ...
    MOVE  ????        TO  ISTRING.
    MOVE  8           TO  OSTRING-LEN.
    MOVE  3           TO  DELIM-COUNT.
    MOVE  ',/ '       TO  DELIMS.
    CALL  'TIPSCAN'   USING  ISTRING  IPTR
                             OSTRING  DELIM-TBL.
```

2.6. Direct Communications I/O

TIP/30 provides facilities that an online program may use to directly interface with ICAM — 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 ICAM (the operating system communications control code) via calls to a TIP/30 subroutine named "TIPTERM".

The program is responsible for:

- issuing messages that conform to ICAM specifications
- 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.

The documentation in this section requires knowledge of the facilities of OS/3 ICAM and UNISCOPE communications protocol.

2.6.1. Message Formats

All input and output messages must begin with a fixed-format message prefix.

COBOL copy elements TIP/TC-DCINP and TIP/TC-DCOUT define the message prefixes in a COBOL program.

Each copy element defines a standard message prefix as documented in the Communications Manuals (ICAM) supplied by the manufacturer, with one exception — an extra fullword added at the beginning of each prefix is used only by TIP/30; the remainder of the prefix is the normal ICAM message prefix for the TCI ICAM interface.

The layout of the input message area is provided by the COBOL copy element TC-DCINP in the TIP library:

```
            COPY TC-DCINP OF TIP.
*
*---------------------------------------------------------------*
*        COPY ELEMENT FOR DCIO INPUT PACKET                     *
*---------------------------------------------------------------*
*
     05  DCIO-INP-PKT.
         10  FILLER                      PICTURE 9(8) COMP 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) COMP SYNC.
         10  DCIO-INP-COUNT              PICTURE 9(4) COMP 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                         *
*---------------------------------------------------------------*
*
```

Where:

DCIO-INP-STATUS

This field is set to the appropriate status after calling TIPTERM with an input function.

Note: *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, ICAM truncates the input message and generates a warning to TIP/30. TIP/30 writes a message to the console that indicates that "Truncated Input" occurred at the noted terminal.

This situation is generally innocuous because ICAM is merely pointing out that more data arrived than the program was willing to handle and the input data that is given to the program has been truncated. This can happen if the terminal operator presses XMIT from a location that results in extra data being sent to the host.

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 ICAM terminal name of the terminal which generated the input. This is normally the same as the terminal which is running the program.

DCIO-INP-DATA

This hanging group item is the last line of the COPY element. The intention is that the programmer codes (immediately following this) whatever elementary items that are needed to allow the program to examine the input message(s).

The layout of the output message area is provided by the COBOL copy element TC-DCOUT in the TIP library:

```
            COPY TC-DCOUT OF TIP.
*
*---------------------------------------------------------------*
*        COPY ELEMENT FOR DCIO OUTPUT PACKET                    *
*---------------------------------------------------------------*
*
     05  DCIO-OUT-PKT.
         10  FILLER                        PICTURE 9(8) COMP 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) COMP 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) COMP SYNC.
     05  DCIO-OUT-DATA.
*---------------------------------------------------------------*
*        USER OUTPUT DATA LAYOUT FOLLOWS                        *
*---------------------------------------------------------------*
*
```

Where:

DCIO-OUT-STATUS

This field is set to the appropriate status when a call is issued to TIPTERM with an output function.

Note: *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/30 accepted the output and passed the information to ICAM. The result returned in DCIO-OUT-STATUS represents the output delivery status from ICAM.*

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. For the appropriate use of this field refer to the ICAM and terminal-specific documentation.

DCIO-OUT-AUX-DVC

This field may be set to the desired auxiliary device number. For the appropriate use of this field refer to the ICAM and terminal-specific documentation. In general, a binary value is placed here (eg: 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 COPY element. 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.

2.7. 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 element (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 element TC-DCINP) or an output message packet (as defined by the COPY element 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 elements TIP/TC-DCINP and TIP/TC-DCOUT to define the appropriate message areas. These copy elements are shown and described below.

ASSEMBLER programs use the macros TIPIMP and TIPOMP that are supplied in the TIP release library. A description of assembler coding and calling techniques is beyond the scope of this manual.

COBOL programs also should include the following supplied copy element in the WORKING-STORAGE section of the program to define the corresponding function codes for calls to TIPTERM:

COPY TC-DCIO OF TIP.

```
*
** TC-DCIO COPY ELEMENT 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-READ                         PICTURE X VALUE  'R'.
     05  T-SEND                         PICTURE X VALUE  'S'.
     05  T-UN                           PICTURE X VALUE  'U'.
     05  T-MSGO                         PICTURE X VALUE  'O'.
     05  T-MSGI                         PICTURE X VALUE  'I'.
     05  T-MSGE                         PICTURE X VALUE  'E'.
     05  T-PHN                          PICTURE X VALUE  '#'.
     05  T-DISC                         PICTURE X VALUE  'D'.
     05  T-HELD                         PICTURE X VALUE  'H'.
     05  T-TEST                         PICTURE X VALUE  'W'.
```

Note: *Many of the function codes listed in the TC-DCIO COPY element are not discussed in this documentation. The description of these functions are beyond the scope of this manual.*

2.7.1. 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
```

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 element — recall that the last line of that copy element 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 (XMIT) and sends more than 1900 bytes in as an input message.

The result: the input is duly truncated by ICAM (and noted on the console by TIP/30) 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.

Table 2-4. TIPTERM (T-GET) Result Status

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.

Note: 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. This is a feature of UNISCOPE terminals!

2.7.2. 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).

Refer to "2.7.5. Output Delivery Notification" on page 2-61 for the possible result status codes (output delivery notification).

Syntax:

```
MOVE ? TO DCIO-OUT-COUNT.
MOVE ? TO DCIO-OUT-DATA.
MOVE ? TO DCIO-OUT-TERM-ID.
CALL 'TIPTERM' USING   T-PUT
                       DCIO-OUT-PKT.
```

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 ICAM terminal name of the intended destination terminal. If this field is invalid, output is sent to the terminal where the program is running.

The following output message can be used to home the cursor and clear the screen:

Example:

```
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 DICE codes 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 data unprotected and protected).

The programming manuals for the various types of terminal contain this type of detailed information about controlling the terminal.

2.7.3. 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.
```

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 element — recall that the last line of that copy element is a group item.

See the description of this field in the discussion of the T-GET function code.

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.*

Table 2-5. TIPTERM (T-TEST) Result Status

STATUS	DESCRIPTION
DCIO-INP-FKEY	Function key received. The first byte of the data area contains the function key code. See description of T-GET function call.
DCIO-INP-GOOD	Input message received.
DCIO-INP-NOT-AVAIL	No input message available.

2.7.4. T-UN — Send Unsolicited Message

The TIPTERM function T-UN allows a program to send an unsolicited output message to a terminal. An unsolicited message is one that is not expected by the receiving terminal. TIP/30 actually sends two outputs to the destination terminal on the high priority queue. The first part is what is called a "sona alert" (a beep). The second part is the actual text of the message.

The sona alert is sent in such a way that the receipt of that message at the terminal causes the Message Waiting alarm to sound (most terminals implements this as a continuous regular "beeping" sound).

The message text (which follows the sona alert) is released from the queue when the terminal operator presses [MSG WAIT] and is displayed *wherever the cursor is resting unless the message contains explicit cursor positioning codes at the beginning of the text.*

Pressing [KEYB UNLK] (keyboard unlock) or any function key causes the terminal to stop beeping. It does not free the high queue or deliver the text of the message.

The terminal operator should position the cursor to an unused area on the terminal prior to pressing the [MSG WAIT] key. The terminal operator often has no idea what is coming in the text message so it is usually quite prudent to finish what is in progress before pressing the [MSG WAIT] key.

Refer to "2.7.5. Output Delivery Notification" on page 2-61 for the possible result status codes.

Syntax:

```
MOVE   ??   TO DCIO-OUT-DATA.
MOVE   ??   TO DCIO-OUT-COUNT.
MOVE   ??   TO DCIO-OUT-TERM-ID.
CALL   'TIPTERM'   USING   T-UN
                           DCIO-OUT-PKT.
```

DCIO-OUT-DATA

The desired output data (and control codes if any) must be placed in the group item DCIO-OUT-DATA.

DCIO-OUT-COUNT

The number of bytes of data (and control codes if any) must be placed in the field DCIO-OUT-COUNT. Only this number of characters is sent.

DCIO-OUT-TERM-ID

The field DCIO-OUT-TERM-ID may be set to the ICAM terminal name of the intended destination terminal. If this field is invalid, output is sent to the terminal where the program is running.

2.7.5. Output Delivery Notification

For all output messages sent to an auxiliary device, the program is not re-scheduled by TIP/30 until ICAM notifies TIP/30 that the message has been delivered. This notification is known as "Output Delivery Notification". For Direct I/O (CALLs to TIPTERM), the output delivery status is returned in the first byte of the output message packet.

WARNING

In some (rare) circumstances *30 seconds or more* may elapse before ICAM returns Delivery Notification to TIP/30. TIP/30 may wait several minutes before deciding that delivery notification is not forthcoming.

All messages sent to auxiliary devices are sent on the LOW priority communications queue, while messages sent to the terminal are sent on the MEDIUM priority queue. This arrangement permits the program to send important informational messages to the terminal operator even though an auxiliary device encounters an error condition.

The delivery status codes are tabled below:

Table 2-6. OUTPUT Delivery Notification Status

Status	Description
DCIO-OUT-GOOD	Good delivery.
DCIO-OUT-LINE-DOWN	Line Down.
DCIO-OUT-TERM-DOWN	Terminal Down.
DCIO-OUT-INV-DEST	Invalid Destination.
DCIO-OUT-NO-BUF	No available buffers.
DCIO-OUT-IO-ERR	Disk I/O error.
DCIO-OUT-INVALID-LEN	Invalid output message length.
DCIO-OUT-NOT-CONN	Destination terminal not connected.
DCIO-OUT-AUX-DOWN	Auxiliary device down.
DCIO-OUT-NOT-OP	Read/Write inoperative.
DCIO-OUT-PAPER-OUT	Printer out of paper.
DCIO-OUT-EOF	End of cassette.
DCIO-OUT-NO-RESP	No response.

In all cases other than DCIO-OUT-GOOD, the output message that caused the error is deleted and the application program must take some recovery action.

Section 3
File Control System (FCS)

3.1. FCS Introduction

This section describes the facilities of the TIP/30 File Control System (FCS). FCS is the TIP/30 component that provides the interface between transaction programs and data files.

FCS allows transaction programs to access:

- Standard OS/3 Data Management files
- Standard OS/3 SAT libraries
- TIP/30 dynamic files (files used on demand)
- TIP/30 edit buffers.

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.

3.2. FCS Overview

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.

Programs refer to files by referencing a Logical File Name (LFN). The LFN is the name by which the file is known to TIP/30. The LFN need not be the same as the actual physical file name (LFD) as supplied in the Job Control for TIP/30. An entry in the TIP/30 catalogue relates a LFN to the actual physical file. All online files must have an entry in the TIP/30 catalogue.

Each file entry in the TIP/30 catalogue specifies the group ownership of the file and the security level required to access the file. Programs may only access files that are assigned to the program (either by an explicit OPEN request to FCS or an implicit OPEN requested in the program's TIP/30 catalogue entry).

Example:

```
PROG EDP/SAMPLE      SECURITY=PROG LOADM=LOADPROG
                     CDA=152 WORK=1024
                     FILES=PAYMAST,PAYDETL
                     TYPE=TIP USAGE=REENT.
```

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.

FCS fully supports the following file organizations:

ISAM	Indexed Sequential Access Method.
DAM	Direct Access Method.
IRAM	Indexed Random Access Method.
SAM	Sequential Access Method.
MIRAM	Multiple Indexed Random Access Method.

FCS also provides:

- the ability to access OS/3 library elements
- the capability of creating (on demand) TIP/30 Dynamic Files
- the ability to access Edit Buffers (a specific type of dynamic file).

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.

TIP/30 has a generalized resource locking facility that enables a program to enter (or remove) a value in the TIP/30 key-holding table. Once a given value has been entered in the table, any process that attempts to enter the same value for the same file receives a "locked" status.

This ability can be used as a generalized queuing mechanism to control access to any resource, for example, an entire file or a group of records.

An online file may be journaled by specifying a TIP/30 generation option. This facility allows either before images or after images (or both) to be written to the TIP/30 journal file:

- **Before** images are often used to roll back updates that were not completed due to a hardware or software failure.
- **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 are direct access files that are managed by TIP/30 (and are in fact subsets of the TIP/30 file "TIP$RNDM"). Programs may create or erase dynamic files, as necessary.

3.3. FCS and the TIP/30 Catalogue

The files that a transaction program needs to access may be specified in the entry for the program in the TIP/30 catalogue. This technique is highly recommended because this feature of the TIP/30 catalogue allows the program to avoid the need to open or close such files explicitly.

A maximum of 12 files may be specified in the program's catalogue entry. If a program requires more than 12 files, explicit calls must be issued by the program (namely: FCS-OPEN and FCS-CLOSE) to gain and relinquish access to the additional files.

Refer to the TIP/30 Catalogue Manager Program (CAT) for information about defining a transaction program.

3.4. Techniques for Deleting Records

TIP/30 supports two types of record delete schemes:

1. Logical record deletion (often called delete flag)
2. Record Control Byte (RCB) deletion.

Logical record delete is available whether the file is an indexed file or a relative record (or direct) file.

RCB delete is available only for MIRAM files created with the RCB delete specification (// DD RCB=YES)

The desired method of deleting records must be specified in the TIP/30 generation parameters for the file. See the documentation of the DELETE= FILE definition keyword in *TIP/30 Generation, Maintenance and Installation — ARP-600-05*. Online programs issue a call to the TIP/30 File Control System (TIPFCS) to delete a record. TIPFCS performs the type of delete operation stated in the generation parameters for the file.

3.4.1. Logical Record Delete

Logical record deletion is a convention established when a file is defined in the TIP/30 generation. 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/30 follows is:

When TIP/30 reads a record from the file that contains the specified flag value in the specified location, TIP/30 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/30" 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 (eg: X'C4' = C'D').

Note: *Programmers must realize that this scheme is merely a convention that TIP/30 follows — the records appear perfectly normal to OS/3 Data Management. In particular, batch programs must be prepared to recognize such "deleted" records and take appropriate action (such as ignoring them!)*

MIRAM indexed files that have one or more secondary keys where duplicates are not allowed should use RCB deletion; otherwise it is not possible to add a record that has a secondary key that matches a previously "logically deleted" record.

3.4.2. Record Control Byte Deletion

OS/3 Data Management provides a record deletion technique for MIRAM files: RCB (Record control byte). This form of record deletion is not a convention — it is a real and effectively permanent delete. An extra byte (the Record Control Byte) is physically associated with each record in the file.

This control byte is not accessible by any program; Data Management uses it to permanently flag a deleted record. Once a record is deleted using the RCB, no program can read the record.

All references to an RCB deleted record are removed from the indices of the file; the data space occupied by the record is not reclaimed — the data space remains as orphaned space in the data partition.

To reclaim the space occupied by RCB deleted records, the file must be physically unloaded and reloaded; the unload program cannot read deleted records and "drops" them. To choose this method of deleting records, you must initially open the MIRAM file with a job control DD statement (or the equivalent statement in the program's file control blocks):

```
//  DD  RCB=YES
```

MIRAM indexed files that have one or more secondary keys where duplicates are not allowed should use RCB deletion; otherwise it is not possible to add a record that has a secondary key that matches a previously "logically deleted" record.

3.5. Setting a File in Sequential Mode

In the online environment that TIP/30 provides, transaction programs often read indexed files in random mode — that is, by providing a specific key of the desired record.

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/30 File Control System to place the file in sequential mode. This technique is analogous to the batch COBOL verb "START".

In general, TIP/30 can allow only one online program at a time to have a particular file in sequential mode. Each program that requests to set that file in sequential mode is queued behind other programs that have that file in sequential mode.

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 GET function code retrieves the next record in sequence.

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.

Note: *Records cannot be updated while a file is set in sequential mode. Only read operations are permitted for a file in sequential mode.*

3.6. MIRAM and Duplicate Keys

You may define a MIRAM file to allow duplicate values on secondary keys although the duplicate keys may present a programming problem when a program must cycle through a set of records that have identical keys.

Consider the following scenario:

An online program wishes to display information from several records in an indexed file. The records are to be accessed via a (secondary) key that allows duplicates. After every screen of data, the user may press a function key to continue viewing subsequent records.

You would probably implement this program according to the following algorithm:

1. Set the file in sequential mode via the secondary key.
2. Read several records (enough to construct one screen full).
3. Set the file to random mode (out of sequential mode).
4. Output the screen.
5. Request terminal input;
   If the user requests to continue the display, go to step 1.

When a program issues a SETL request using a (secondary) key that allows duplicates, Data Management **always** positions the file at the beginning of the set of duplicates — this is undesirable (except the first time through the cycle!)

TIP/30 requires a program to release all serial resources (which includes a file in sequential mode) before requesting terminal input. However, if the specification MULTISEQ=YES is specified for this file in the TIP/30 generation parameters, TIP/30 will **not** treat setting this file in sequential mode as locking a serial resource and the program need not issue an ESETL function across terminal input.

One approach to this problem is to have the program read and ignore already displayed records. To accomplish this, the program can save the primary key of the last record on the current screen full — the primary key must be unique by definition — and on the next cycle skip records until it encounters the primary key again.

WARNING

What happens if some other user deletes that particular record in the meantime? The program must be prepared to handle that possibility.

Another approach is to construct a table of the PRIMARY keys of the records in the duplicate set (how big a table?) and then use the table to "step" through the keys by reading randomly via the primary key in the table.

Each of these approaches has positive and negative aspects. The first approach may be an "optimistic" approach because it works best if the user doesn't step too far through the set of duplicates.

The latter approach incurs much of the I/O overhead of building the table of keys on the assumption that the user usually steps a long way through the set of duplicates.

Perhaps a combination of the two approaches is best.

An alternative that is available for MIRAM files that are accessed using Consolidated Data Management (CDM) and on OS/3 Release 11 (or later) is to generate the file to allow multiple sequential readers (file generation parameter MULTISEQ=YES). This specification informs TIP/30 there is no need to queue programs trying to set this file in sequential mode. Since no queuing is involved, this situation is not treated as a serial resource and terminal input is permitted while the file is in sequential mode.

3.7. 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.

To illustrate the problem, assume that JOE and TOM are working at different terminals updating FILEX and there is no record locking capability:

1. JOE displays record 500 intending to update it.
2. JOE is interrupted for a moment and TOM reads record 500, changes it at his terminal and rewrites the record in the file.
3. 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/30 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).

3.7.1. HOLD=YES — Simple Record Locking

Specify HOLD=YES in the TIP/30 generation parameters for a file 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 user program receives a PIB-HELD status when it attempts to FCS-GETUP a record from a file with HOLD=YES, TIPFCS releases all other record locks acquired by the program (for files defined as HOLD=YES) and the program must re-acquire all those record locks.

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 slightly less than one second. The program may try the GETUP again or CALL TIPTIMER to wait a little longer.

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.

3.7.2. HOLD=UP — Record Locking for Update

Specifying HOLD=UP in the TIP/30 generation 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).

3.7.3. HOLD=TR — Record Locking for Transaction

Specifying HOLD=TR in the TIP/30 generation parameters for a file causes the TIP/30 File Control System to lock the records for that file for the duration of the transaction.

Refer also to the definition of transaction end in the Program Control System (PCS) section of this documentation.

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/30 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 TIP$B4 file. TIP/30 uses these quick before images to roll back the record if the transaction does not complete normally.

A user program may issue a CALL to FCS-BACK to roll back updates on a file-by-file basis for HOLD=TR files (or more simply use the facilities provided by the field "PIB-LOCK-INDICATOR" as described in the Program Control System section).

3.7.4. Record Locking Summary

The table below compares the record locking schemes that are supported by TIP/30.

Table 3-7. Record Locking Summary

	HOLD=YES	HOLD=UP	HOLD=TR
Roll back Capability?	NO	NO	YES
Hold Multiple Records per File?	NO	YES	YES
When Records Released?	Update, NOUP, or next GETUP	Update or NOUP	TREN
Release on PIB-HELD Status?	YES	NO	NO
Possible Deadlock?	NO	YES	YES

3.8. Call TIPFCS — Common Parameters

Calls to TIPFCS must pass a number of 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 element TC-FCS from the TIP library.

Warning: passing an invalid function code to the TIPFCS subroutine may cause unpredictable results.

file-pkt The Logical file name packet.

Must contain the logical file name of the file (as it is defined in the TIP/30 Catalogue).

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 (ie: PIC 9(6) COMP SYNC).

index-num

For multi-indexed MIRAM, this specifies the desired index number (one through five inclusive).

Define this field as a binary halfword (PIC 9 COMP SYNC). If this parameter is omitted, the primary key for the file 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.

3.9. File System Function Codes

The first parameter on every call to TIP/30 FCS is a one-byte function code that specifies the file system operation to be performed. A copy element is supplied with TIP/30 that COBOL programs can use to define the function codes.

Include this copy element in the WORKING-STORAGE SECTION of the COBOL program (the name selected for the 01 level item is not particularly important):

```
       01 FCS-FUNCTION-CODES.      COPY TC-FCS OF TIP.

*    TC-FCS COPY ELEMENT FOR TIP/30 FILE CONTROL INTERFACE

****************************************************************
*    THE FOLLOWING DATA ITEMS ARE FCS FUNCTION CODES           *
****************************************************************
     05  FCS-ACCESS                    PICTURE X VALUE 'A'.
     05  FCS-ADD                       PICTURE X VALUE '9'.
     05  FCS-ASSIGN                    PICTURE X VALUE '>'.
     05  FCS-BACK                      PICTURE X VALUE 'B'.
     05  FCS-CLOSE                     PICTURE X VALUE 'D'.
     05  FCS-CREATE                    PICTURE X VALUE 'N'.
     05  FCS-DELETE                    PICTURE X VALUE '<'.
     05  FCS-ESETL                     PICTURE X VALUE '6'.
     05  FCS-FLUSH                     PICTURE X VALUE 'F'.
     05  FCS-GET                       PICTURE X VALUE 'G'.
     05  FCS-GETUP                     PICTURE X VALUE '0'.
     05  FCS-GETRN                     PICTURE X VALUE '7'.
     05  FCS-HOLD                      PICTURE X VALUE 'H'.
     05  FCS-JOURNAL                   PICTURE X VALUE 'T'.
     05  FCS-NEXT                      PICTURE X VALUE 'X'.
     05  FCS-NOUP                      PICTURE X VALUE '2'.
     05  FCS-OPEN                      PICTURE X VALUE 'O'.
     05  FCS-PUT                       PICTURE X VALUE 'P'.
     05  FCS-RELEASE                   PICTURE X VALUE 'R'.
     05  FCS-SCRATCH                   PICTURE X VALUE 'Q'.
     05  FCS-SETL                      PICTURE X VALUE '5'.
     05  FCS-SETL-BOF                  PICTURE X VALUE 'S'.
     05  FCS-SETL-EQ                   PICTURE X VALUE 'E'.
     05  FCS-SETL-GT                   PICTURE X VALUE 'Z'.
     05  FCS-SKIP                      PICTURE X VALUE 'I'.
     05  FCS-TREN                      PICTURE X VALUE '*'.

****************************************************************
*  THE FOLLOWING DATA ITEMS ARE FCS DYNAMIC FILE CLASSES       *
****************************************************************
     05  FCS-CLASS-PERM                PICTURE X VALUE 'P'.
     05  FCS-CLASS-TEMP                PICTURE X VALUE 'T'.
     05  FCS-CLASS-QED                 PICTURE X VALUE 'E'.

****************************************************************
*  THE FOLLOWING DATA ITEMS ARE FCS DYNAMIC FILE TYPES         *
```

```
*****************************************************************
     05  FCS-TYPE-NEW                PICTURE X VALUE 'C'.
     05  FCS-TYPE-OLD                PICTURE X VALUE 'E'.

*****************************************************************
*  THE FOLLOWING DATA ITEMS ARE FCS FILE PERMISSIONS            *
*****************************************************************
     05  FCS-PERM-READONLY           PICTURE X VALUE 'R'.
     05  FCS-PERM-WRITEONLY          PICTURE X VALUE 'W'.
     05  FCS-PERM-UPDATE             PICTURE X VALUE 'U'.

*****************************************************************
*  THE FOLLOWING DATA ITEMS ARE FCS LOCK OPTIONS                *
*****************************************************************
     05  FCS-LOCK-YES                PICTURE X VALUE 'Y'.
     05  FCS-LOCK-NO                 PICTURE X VALUE 'N'.
```

3.10. FCS Interface Packets

Two packets are used to control processing of files through FCS:

1. Logical File Name Packet
2. 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.

3.10.1. Logical File Name Packet

This is the primary control packet used for processing files. It consists of two fields:

1. 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/30 Catalogue to determine which physical file is actually used by the program
2. 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/30 Catalogue 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.

3.10.2. File Descriptor 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/30 Dynamic Files, TIP/30 Edit Buffers, Library elements and, in situations where unusual processing is desired (such as opening a file with read-only access).

```
        02 FILE-DESCRIPTOR.   COPY TC-FDES OF TIP.
*****************************************************************
*                 FCS FILE DESCRIPTOR PACKET                    *
*****************************************************************
       05  FDES-USER-ID                PICTURE X(8).
       05  FDES-CATALOG                PICTURE X(8).
       05  FDES-FILE-NAME              PICTURE X(8).
       05  FDES-PASSWORD -             PICTURE X(8).
       05  FDES-FCS-CLASS              PICTURE X.
       05  FDES-FCS-TYPE               PICTURE X.
       05  FDES-FCS-PERM               PICTURE X.
       05  FDES-FCS-LOCK               PICTURE X.
*****************************************************************
*         ADDITIONAL FIELDS FOR LIBRARY ELEMENT ACCESS          *
*****************************************************************
       05  FDES-ELEMENT                PICTURE X(8).
       05  FDES-COMMENTS               PICTURE X(30).
       05  FDES-DATE                   PICTURE X(8).
       05  FDES-TIME                   PICTURE X(5).
*****
```

Where:

FDES-USERID

May contain the USERID or Group name to which the file belongs.

If opening for:

Read A complete search of the TIP/30 catalogue is done.

Output Uses the specified value. If creating a dynamic file, this is set to the callers USERID.

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-FILE-NAME

File name for dynamic files or the catalogued 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 an access password to a TIP/30 Dynamic file or Edit Buffer. The password may be established when the file is created; thereafter, all attempts to open the file must supply the same password that was used to create the file.

If this field is spaces or low-values, no password is assigned for the file.

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/30 catalogue with the supplied name.

E	Edit Buffer.
P	Permanent dynamic file.
S	Data management 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	Input file with PUT allowed.
(space)	Read/write.

FDES-FCS-LOCK

For Edit Buffers and TIP/30 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.

Note: *PIB-LOCKED status is returned if any other process (online program) is using the Edit Buffer or Dynamic File.*

3.11. FCS Miscellaneous Functions

3.11.1. FCS-BACK — ROLL BACK Changes

The FCS-BACK function causes FCS to roll back (undo) changes made to a specified file by the issuing process, since the last transaction end point (a discussion of transaction end is contained in the "Program Control System" chapter of this manual). This function is applicable only to a file that is defined to have record holding for the duration of the transaction (HOLD=TR in the TIP/30 generation parameters).

TIP/30 accomplishes the roll back using the information retained in the TIP/30 quick before look file (TIP$B4 file).

> **WARNING**
>
> This function rolls back all changes (updates, adds, deletes, etc.) for the single file identified by the CALL. If the program wishes to roll back all changes made by the transaction, the field PIB-LOCK-INDICATOR should be used.

Syntax:

```
CALL  'TIPFCS'  USING  FCS-BACK
                       file-pkt
```

FCS-BACK

Function code from the TC-FCS copy book.

file-pkt Logical file name packet.

Error Conditions:

PIB-FUNCTION The file named in the file-pkt is not assigned to the program.

Additional Considerations:

A side effect of the use of this function is that the file is taken out of sequential mode (if it was in that mode).

3.11.2. FCS-HOLD — Hold Resource

A program may use this function to place a user-defined value in the TIP/30 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 roll back considerations apply and the lock is not discarded if the process receives PIB-HELD status on some other FCS-HOLD call.

An important point to understand is that 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 copy book.

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 value of a fullword of one is used.

Error Conditions:

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).

3.11.3. FCS-JOURNAL — Write User Journal Record

Programs may use this function to write a user defined record to the TIP/30 Journal file (TIP$JRN) or the TIP/30 Log file (TIP$LOG). These "user" journal records are often used for accounting or audit purposes. A separate section of this manual describes the contents of the journal file records and the mechanics of reading the journal file in a user written batch program. This section describes how an online program can create and write "user" type records to the TIP/30 journal (or log) file.

Syntax:

```
CALL  'TIPFCS'  USING  FCS-JOURNAL
                       dummy
                       rec
```

Where:

FCS-JOURNAL

Function code from the TC-FCS copy book.

dummy This is a dummy parameter required to maintain symmetry with other file system calls. FCS ignores the contents of this parameter.

journal-rec

Record area containing the journal record data that is to be written to the TIP$JRN or TIP$LOG file.

Additional Considerations:

The copy element TIP/TC-JRN (see the section of this manual entitled "Journal File Processing") 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 copy element it is named JRN-REC-LEN).

The value placed in JRN-REC-LEN is the length of the journal record prefix (56 bytes) plus the number of bytes of data that follows the prefix.

The program need not supply any other information in the prefix area since the TIP/30 file system fills in the information before writing the USER record to the journal file.

3.11.4. FCS-RELEASE — Release Resource

Release an entry in the TIP/30 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 copy book.

file-pkt Logical file name 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 is required.

If this parameter is omitted, a value of a fullword of one is used.

Error Conditions:

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.

3.11.5. FCS-TREN — Mark Transaction End

If a file is defined in the TIP/30 Generation parameters with HOLD=TR ("hold for transaction"), TIP/30 automatically rolls back any updates if a program aborts while updating that file. 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 copy book.

Only one parameter is required; any other parameters are ignored.

Use of FCS-TREN signals transaction end to TIP/30. Another use of this function is to cause TIP/30 to examine a value that the program has placed in the field PIB-LOCK-INDICATOR — for example, a program may place "O" in that field and then call FCS-TREN to force a transaction roll back.

Example ① Roll back updates done so far:

```
MOVE  'O'       TO PIB-LOCK-INDICATOR
CALL  'TIPFCS'  USING FCS-TREN
```

Example ② Mark new roll back point:

```
MOVE  SPACE     TO PIB-LOCK-INDICATOR
CALL  'TIPFCS'  USING FCS-TREN
```

3.12. 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/30 File Control System (FCS). When a program issues a call to TIPFCS, any error status is returned in two places:

1. the PIB (PIB-STATUS)
2. 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 (eg: 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        PIC X(9).
05  FCS-ERROR-TEXT     PIC 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.

3.13. TIPFCS for Indexed Files

This section describes TIP/30 file control system operations you may specify for indexed files. Indexed files include ISAM, MIRAM (single or multi-index) and IRAM. TIP/30 uses MIRAM data management to access IRAM files.

User programs may access MIRAM files via any of the indices defined for the file. When a MIRAM multi-indexed file is defined in the TIP/30 generation parameters, the length, location, and the attributes of each of the keys must be stated and one of the keys must be designated (PKEY=) as the primary key (KEY1 is the default primary key).

Multi-indexed MIRAM files used by TIP/30 must have the primary key defined as (NDUP,NCHG) — no duplicate key values and no changes allowed to this key.

In the TIP/30 catalogue entry for a file, the keyword KREF= may be specified to establish a default index number for the file. If a program accesses a file and does not explicitly state the implied index number that is meant, the KREF= value is used by TIPFCS to determine which index the user program is using (1 through 5).

3.13.1. 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 the file is defined to use Logical Record delete, the designated delete flag byte should be set to some neutral value before issuing an FCS-ADD function. If the delete flag byte contains the designated delete flag value, a logically deleted record is added — a subsequent FCS-GET operation returns PIB-NOT-FOUND.

If an FCS-ADD function is issued for a record that is logically deleted, the TIP/30 File System allows the "add" operation by rewriting the logically deleted record.

Syntax:

```
CALL  'TIPFCS'  USING  FCS-ADD
                       file-pkt
                       record
```

Where:

FCS-ADD	Function code from the TC-FCS copy element.
file-pkt	Logical file name packet.
record	Record area containing new record data.

Error Conditions:

PIB-DUP-KEY	A record with the same key already exists.
PIB-FUNCTION	The file is not assigned to the program.
PIB-IO-ERROR	An I/O error occurred on the disk.
PIB-WRONG-MODE	Write operations are not permitted for the file.

3.13.2. 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 and the file was generated with OPEN=NO, TIPFCS physically CLOSEs the file by issuing a "CLOSE" imperative macro to Data Management.

Syntax:

```
CALL  'TIPFCS'  USING  FCS-CLOSE
                       file-pkt
```

Where:

FCS-CLOSE

Function code from the TC-FCS copy element.

file-pkt

Logical file name packet.

Error Conditions:

PIB-FUNCTION	File is not assigned to the program.

Additional Considerations:

This function is appropriate for files that were opened by issuing an FCS-OPEN function (files explicitly accessed by the program).

3.13.3. FCS-DELETE — Indexed: Delete Record

The FCS-DELETE function call deletes a record from the file. The TIP/30 file system uses the applicable delete scheme as specified in the TIP/30 generation parameters for the file. If DELETE=RCB is specified in the file's generation parameters, TIPFCS deletes the record using RCB, otherwise, the defined delete flag value is placed in the record and the record is updated in the file.

For additional information, see "3.4. Techniques for Deleting Records" on page 3-5.

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 copy element.

file-pkt Logical file name packet.

record Record area.

Error Conditions:

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 was released by TIP/30.

Additional Considerations:

If the file is defined to use "logical record deletion" — for example, X'FF' in a particular byte — the application program need not move the delete value to the appropriate location in the record area since the FCS-DELETE activity performs this automatically.

3.13.4. FCS-ESETL — Indexed: End Sequential Mode

Set a file to random processing mode (terminate sequential processing of the file). TIP/30 allows only one process at time to set a particular file in sequential mode. Other processes that wish to set the file in sequential mode are queued waiting for the file to be ESETL.

A process that has a file in sequential mode may not ask for input from the terminal. If an attempt is made the process is terminated with a "Resource Lock Exception" error. Before requesting terminal input, the program must issue an FCS-ESETL function call for each file that is in SETL mode.

The foregoing discussion applies to files that are not defined with MULTISEQ=YES in the generation parameters for the file. When multiple sequential readers are allowed, TIP/30 does not consider having that file in sequential mode as a serial resource. Therefore, leaving such a file in sequential mode across screen input is permitted.

Syntax:

```
CALL  'TIPFCS'  USING  FCS-ESETL
                       file-pkt
```

Where:

FCS-ESETL

Function code from the TC-FCS copy element.

file-pkt

Logical file name packet.

Error Conditions:

PIB-FUNCTION	The file is not assigned to the program.

3.13.5. FCS-FLUSH — Indexed: Flush File

The FCS-FLUSH function requests the TIP/30 file system to physically CLOSE and reopen a specific file. This function forces the updating of the VTOC end-of-data pointers after records are added to a file.

Use the FCS-FLUSH with discretion since it is a relatively time consuming operation that makes the file inaccessible to everyone for a short period of time.

> **WARNING**
>
> If this function is issued for a file that is generated as MULTISEQ=YES, the function is not performed. Physically closing a file that has multiple sequential readers causes Data Management to lose the sequential position information that was being maintained for all the sequential readers.

This function was occasionally beneficial before the introduction of Consolidated Data Management and, specifically, before the introduction of the MIRAM recovery option. If the indexed file was created with // DD RECV=YES there is no valid reason to issue an FCS-FLUSH function for that file.

Syntax:

```
CALL  'TIPFCS'  USING  FCS-FLUSH
                       file-pkt
```

Where:

FCS-FLUSH

Function code from the TC-FCS copy element.

file-pkt

Logical file name packet.

Error Conditions:

PIB-FUNCTION	The file is not assigned to the program.
PIB-IO-ERROR	An I/O error occurred on the disk. See additional considerations that follow.

Additional Considerations:

In a multi-job environment, there is the chance of a race condition that would allow a batch job that was waiting for the file to gain access to the file after TIPFCS issued the CLOSE but before TIPFCS is able to issue the OPEN. In this case, the OPEN may fail (with DM88) and the program receives PIB-IO-ERROR status.

3.13.6. FCS-GET — Indexed: Read by Key

Read a record with a specific key from a file. Using FCS-GET implies that the record is not locked for update.

This section discusses the behaviour of FCS-GET assuming that the file is not already set in sequential mode.

Syntax:

```
CALL  'TIPFCS'  USING  FCS-GET
                       file-pkt
                       record
                     [ key       ]
                     [ index-num ]
```

Where:

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

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. The Logical File definition in the TIP/30 catalogue may have specified a default index number by use of the KEYREF= keyword (see description of that keyword in the documentation of the CAT transaction FILE command). If an explicit KEYREF= value is not provided, the default index corresponds to the primary key of the file.

Error Conditions:

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.

3.13.7. FCS-GET — Indexed: Read Sequential

Read the next record from a file that has already been set in sequential mode (by a prior call to one of the various type of FCS-SETL-xx function). 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 copy element.

file-pkt Logical file name packet.

record Area where record data is placed.

Error Conditions:

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.

Note: An FCS-GET issued for a file in sequential mode never returns logically deleted records — TIPFCS automatically skips these records when in sequential mode. Records deleted by the RCB method cannot be read under any circumstances and cannot be returned either!

3.13.8. FCS-GET — Indexed: Read Nth Duplicate

A common processing requirement is to retrieve a specific record within a set of identical key values — naturally, the implication is that we are dealing with a MIRAM file with a secondary key that allows duplicates.

The FCS-GET function may be used to retrieve a specific record within a set of duplicate records.

Since this operation mimics the processing that the application program would normally have to perform in sequential mode, the treatment of deleted records is that same as that observed during sequential FCS-GET operations: deleted records are ignored by this FCS function — the appearance of a deleted record does not affect the "count" while the file system is searching for the Nth occurrence.

When using logical record deletion for the file, the deleted records must be read and skipped by the file system (the records must be physically read to determine that they are flagged as deleted records!). If RCB deletion is being used for the file, deleted records are not an issue since they cannot be read by any program.

Syntax:

```
CALL  'TIPFCS'  USING  FCS-GET
                       file-pkt
                       record
                       key
                       index-num
                       dup-count
```

Where:

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

file-pkt Logical file name packet.

record Area where record data is placed.

key Record key.

index-num Binary halfword holding the intended index number.

dup-count Binary fullword holding the ordinal number of the desired record. For example: 100 means "return the 100th record".

Note: The presence of 6 parameters on this function call is the method that FCS uses to detect that this special use of FCS-GET is desired. All parameters must be supplied.

Error Conditions:

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 Nth record does not exist.
PIB-EOF	End of file is reached.
PIB-DUPS-AHEAD	This value 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.

3.13.9. FCS-GETRN — Indexed: Read Relative Number

Read a record from an indexed MIRAM file via a relative record number. Using FCS-GETRN implies that the record is not locked for update.

Syntax:

```
CALL  'TIPFCS'  USING  FCS-GETRN
                       file-pkt
                       record
                       rel-rec-num
```

Where:

FCS-GETRN

Function code from the TC-FCS copy element.

file-pkt Logical file name packet.

record Area where the record data is placed.

rel-rec-num A binary fullword containing the relative record number of the record to read. After reading a record from a MIRAM file, the TIP/30 File Control System places the relative record number of that record in the PIB field PIB-MIRAM-REL-REC-NUM so that interested programs can save this value as an independent marker to the record.

Programs that need to read through an indexed MIRAM file and are not concerned with the order of the records, can read the file using FCS-GETRN (incrementing the requested relative record number before each call).

This technique can be employed, for example, when a search of a file is requested and the data is not part of any key. Although the program could easily search the file in sequential mode, reading the file using repeated FCS-GETRN calls bypasses the overhead of manipulating the index.

Error Conditions:

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, has been deleted using RCB (Record Control Bytes), is flagged deleted using a logical delete flag, or is beyond the limits of the file. If the record is logically deleted, the record data is returned in the specified record area.

Additional Considerations:

Programs may issue this call whether or not the file is in sequential mode. The appropriate record and status is returned without disturbing the current sequential read position (subsequent FCS-GET operations resume where they were suspended).

3.13.10. 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/30 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-GETUP or FCS-ADD the same record. Use this function only when the file is in random processing mode — TIP/30 does not support record updating for a file in sequential mode.

Syntax:

```
CALL  'TIPFCS'  USING  FCS-GETUP
                       file-pkt
                       record
                     [ key       ]
                     [ index-num ]
```

Where:

FCS-GETUP

Function code from the TC-FCS copy element.

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. The Logical File definition in the TIP/30 catalogue may have specified a default index number by use of the KEYREF= keyword (see description of that keyword in the documentation of the CAT transaction FILE command). If an explicit KEYREF= value is not provided, the default index corresponds to the primary key of the file.

Error Conditions:

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-HELD	The record is currently locked by some other process in the TIP/30 system. Normally, programs that receive PIB-HELD will 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.
PIB-WRONG-MODE	FCS-GETUP issued for a file that is not set for random processing.

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 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.

The key information that is stored in the key holding table (and is used by TIP/30 to enforce record locks), is a length computed as the lesser of:

- the length of the primary key for the file
- the value specified for the KEYHOLD= parameter for the FILE
- the width of the TIP/30 key holding table (see TIPGEN keyword KEYTABLE=).

3.13.11. FCS-NEXT — Indexed: Get Next Record

The FCS-NEXT function retrieves the next record (sequentially) from an indexed file. This function code is the equivalent of issuing calls (in sequence) for FCS-SETL-GT, FCS-GET and FCS-ESETL.

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.

Syntax:

```
CALL  'TIPFCS'  USING  FCS-NEXT
                       file-pkt
                       record
                     [ key       ]
                     [ index-num ]
```

Where:

FCS-NEXT Function code from the TC-FCS copy element.

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.

> **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-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 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. The Logical File definition in the TIP/30 catalogue may have specified a default index number by use of the KEYREF= keyword (see description of that keyword in the documentation of the CAT transaction FILE command). If an explicit KEYREF= value is not provided, the default index corresponds to the primary key of the file.

Error Conditions:

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-WRONG-MODE	FCS-NEXT issued for a file that is not currently set for random processing.
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.

3.13.12. FCS-NOUP — Indexed: Cancel Update

The FCS-NOUP function call is used to "unlock" a record that has been acquired via a prior call to FCS-GETUP. In certain situations, a program may issue an FCS-GETUP and lock a record only to later determine that an update is not appropriate.

Syntax:

```
CALL  'TIPFCS'  USING  FCS-NOUP
                       file-pkt
                     [ key ]
```

Where:

FCS-NOUP Function code from the TC-FCS copy element.

file-pkt Logical file name packet.

key Specific 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-FUNCTION	The file is not assigned to the program.
PIB-NOT-HELD	The record was not held.
PIB-WRONG-MODE	FCS-NOUP for a file that is not set for random processing.

3.13.13. 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 (up to 12) file names as defined in the program's TIP/30 catalogue entry. If a program needs to access more than 12 files, some of them must be opened by issuing explicit calls to TIPFCS with the FCS-OPEN function.

For the FCS-OPEN function to be successful, the file to be opened must be:

- defined in the TIP/30 job control stream — user data files are normally defined in the job control Proc named "TIPDATA"
- defined in the TIP/30 Catalogue (this is where the connection is made between a logical file name (LFN) and the physical file name (LFD)
- defined in the TIP/30 Generation parameters (see the FILE generation statement).

If there are no other users of the file and the file is specified in the TIP/30 generation parameters as "OPEN=NO", TIPFCS physically attempts to OPEN the file by issuing a Data Management OPEN.

Syntax:

```
CALL  'TIPFCS'  USING  FCS-OPEN
                       file-pkt
                     [ file-desc ]
```

Where:

FCS-OPEN Function code from the TC-FCS copy element.

file-pkt Logical file name packet.

file-desc File descriptor packet — see separate description of the copy element "TC-FDES". If omitted, the name in the file-pkt is used to build a file descriptor.

Error Conditions:

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.

3.13.14. FCS-PUT — Indexed: Update Record

Update (rewrite) 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 copy element.

file-pkt Logical file name packet.

record Record area containing the record contents.

Error Conditions:

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/30 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/30 (see discussion of record locking techniques).
PIB-WRONG-MODE	Write operations are not permitted for the file.

3.13.15. 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.

Note: This function does not return a record — it simply establishes a starting point for sequential reading.

Subsequent calls with a FCS-GET function retrieve records in sequence. If the file is currently set in sequential mode by another process, TIPFCS queues this request until the file is set back to random mode. You should ensure that an ESETL is issued for the file prior to requesting input from the terminal — if you do not, the program may abort with a "Resource Lock Exception" error message.

Refer also to the description of the FCS-ESETL function.

Syntax:

```
CALL  'TIPFCS'  USING  FCS-SETL
                       file-pkt
                     [ key       ]
                     [ index-num ]
                     [ key-len   ]
```

Where:

FCS-SETL Function code from the TC-FCS copy element.

file-pkt Logical file name packet.

key Record key.

If omitted, processing begins with the first record in the file according to the primary key. (If this parameter is omitted, all following parameters must be omitted too since COBOL has no provision for omitting parameters in the middle of a list).

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. The Logical File definition in the TIP/30 catalogue may have specified a default index number by use of the KEYREF= keyword (see description of that keyword in the documentation of the CAT transaction FILE command). If an explicit KEYREF= value is not provided, the default index corresponds to the primary key of the file.

key-len **MIRAM only.** 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.

Error Conditions:

PIB-FUNCTION	The file is not assigned to the program.
PIB-IO-ERROR	An I/O error occurred on the disk.

3.13.16. 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 function with a dummy key consisting of low-values.

Note: 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.

If the file is currently set in sequential mode by another process, TIPFCS queues this request until the file is set back to random mode. Ensure that an ESETL is issued for the file prior to requesting input from the terminal otherwise the program may abort with a "Resource Lock Exception" error message.

Refer also to the description of the FCS-ESETL function.

Syntax:

```
CALL  'TIPFCS'  USING  FCS-SETL-BOF
                       file-pkt
                     [ index-num ]
```

Where:

FCS-SETL-BOF

Function code from the TC-FCS copy element.

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. The Logical File definition in the TIP/30 catalogue may have specified a default index number by use of the KEYREF= keyword (see description of that keyword in the documentation of the CAT transaction FILE command). If an explicit KEYREF= value is not provided, the default index corresponds to the primary key of the file.

Error Conditions:

PIB-FUNCTION	The file is not assigned to the program.
PIB-IO-ERROR	An I/O error occurred on the disk.

3.13.17. 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 is only operational on MIRAM files; if the function is used on an ISAM file, it is treated as an ordinary FCS-SETL.

Note: This function does not return a record — it simply establishes a starting point for sequential reading. Subsequent calls with a FCS-GET function retrieve records in sequence.

If the file is currently set in sequential mode by another process, TIPFCS queues this request until the file is set back to random mode. Ensure that an ESETL is issued for the file prior to requesting input from the terminal otherwise the program may abort with a "Resource Lock Exception".

Refer also to the description of the FCS-ESETL function.

Syntax:

```
CALL  'TIPFCS'  USING  FCS-SETL-EQ
                       file-pkt
                     [ key       ]
                     [ index-num ]
                     [ key-len   ]
```

Where:

FCS-SETL-EQ

Function code from the TC-FCS copy element.

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. The Logical File definition in the TIP/30 catalogue may have specified a default index number by use of the KEYREF= keyword (see description of that keyword in the documentation of the CAT transaction FILE command). If an explicit KEYREF= value is not provided, the default index corresponds to the primary key of the file.

key-len **MIRAM only.**

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.

Error Conditions:

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:

Use of FCS-SETL-EQ may result in a sequential reading start position that represents a logically deleted record. In this case, the subsequent call with an FCS-GET function skips the logically deleted record (or records) and returns the next record in sequence that is not logically deleted.

The first FCS-GET operation that is issued after a FCS-SETL-EQ may return a record that does NOT have the key that was specified in the FCS-SETL-EQ.

3.13.18. 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 is intended for MIRAM files; if the function is used on an ISAM file, it is treated as FCS-SETL.

Note: 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.

If the file is currently set in sequential mode by another process, TIPFCS queues this request until the file is set back to random mode. Ensure that an ESETL is issued for the file prior to requesting input from the terminal otherwise the program may abort with a "Resource Lock Exception" error.

Refer also to the description of the FCS-ESETL function.

Syntax:

```
CALL  'TIPFCS'  USING  FCS-SETL-GT
                       file-pkt
                     [ key       ]
                     [ index-num ]
                     [ key-len   ]
```

Where:

FCS-SETL-GT

Function code from the TC-FCS copy element.

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. The Logical File definition in the TIP/30 catalogue may have specified a default index number by use of the KEYREF= keyword (see description of that keyword in the documentation of the CAT transaction FILE command). If an explicit KEYREF= value is not provided, the default index corresponds to the primary key of the file.

key-len **MIRAM only.** 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.

Error Conditions:

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.

3.13.19. FCS-SKIP — Indexed: Skip Sequentially

The FCS-SKIP function is appropriate only for an indexed MIRAM file set in sequential mode. A specified number of records are skipped. Subsequent calls with a FCS-GET function (read sequentially) continue at the point where the FCS-SKIP ended.

Note: *This function does* **NOT** *return a record — it simply establishes a starting point for subsequent sequential reading.*

If the MIRAM file is specified with RCB record deletion, the records are very quickly skipped by utilizing the index of the file — the records are not physically read.

If the MIRAM file is specified with logical record deletion, the records must be physically read to determine whether or not a record has been flagged deleted.

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 copy element.

file-pkt Logical file name packet.

skip-count Binary fullword holding the number of records to skip.

Error Conditions:

PIB-FUNCTION	The file is not assigned to the program.
PIB-WRONG-MODE	The file is not MIRAM type, or is not already in sequential mode.
PIB-IO-ERROR	An I/O error occurred on the disk.
PIB-EOF	End of file reached.

3.14. TIPFCS for Direct Files

This section describes TIP/30 file control system operations you may specify for direct access files. Direct files include non-indexed MIRAM or IRAM and DA (direct access). Records are referenced by a relative record number; for example, record number 1 is the first record in the file. For DAM, the relative record number is also known as the "block number".

In all cases the key passed to **TIPFCS** is a binary fullword that holds the relative record number of the record to be processed.

3.14.1. FCS-ADD — Direct: Add Record

The FCS-ADD function code 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 copy element.

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 reentrant execution.*

Error Conditions:

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 journaled (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.

3.14.2. FCS-CLOSE — Direct: Close File

The FCS-CLOSE function call indicates that a program is relinquishing access to a file. TIP/30 removes the corresponding entry for the file from the Active File Table (AFT) of the issuing process.

If there are no other online users of the file and the file was generated with OPEN=NO, TIPFCS physically closes the file by issuing a "CLOSE" imperative macro to Data Management.

Syntax:

```
CALL  'TIPFCS'  USING  FCS-CLOSE
                       file-pkt
```

Where:

FCS-CLOSE

Function code from the TC-FCS copy element.

file-pkt Logical file name packet.

Error Conditions:

PIB-FUNCTION File is not assigned to the program.

Additional Considerations:

This function is used for files that were opened by issuing an FCS-OPEN function (files explicitly accessed by the program).

3.14.3. 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/30 generation parameters for the file:

- Record Control Byte (RCB) deletion (allowed by MIRAM)
- Logical record deletion.

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 copy element.

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 referenced by the process.

Error Conditions:

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 was not successfully done for this record or the record lock was released.

Additional Considerations:

If the file is defined to use "logical record deletion" (for example: X'FF' in a particular byte), the application program need not move the delete value to the appropriate location in the record area — the FCS-DELETE activity does this automatically.

3.14.4. FCS-FLUSH — Direct: Flush File

The FCS-FLUSH function requests that the TIP/30 file system physically CLOSE and reopen a specific file.

Use the FCS-FLUSH with discretion since it is a relatively time consuming operation that makes the file inaccessible to everyone for a short time.

Syntax:

```
CALL  'TIPFCS'  USING  FCS-FLUSH
                       file-pkt
```

Where:

FCS-FLUSH

Function code from the TC-FCS copy element.

file-pkt Logical file name packet.

Error Conditions:

PIB-FUNCTION	The File is not assigned to the program.
PIB-IO-ERROR	An I/O error occurred on the disk.

3.14.5. FCS-GET — Direct: Read Record

Read a specific record from a direct file. Using FCS-GET implies that 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 copy element.

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-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. If the record is logically deleted, the record data is returned in the specified record area (to allow the program to reset the delete flag character and add the record by using the FCS-ADD function).

3.14.6. FCS-GETUP — Direct: Read With Lock

Read the record with the specified relative record number with intent to update. The relative record number is placed in the TIP/30 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 copy element.

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-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/30 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.

3.14.7. FCS-NOUP — Direct: Cancel Update

You may use the FCS-NOUP function call to release the lock on a record that was acquired via a previous call to FCS-GETUP. In certain situations, a program may issue a FCS-GETUP and lock a record and only then 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 copy element.

file-pkt Logical file name packet.

rel-rec-num

Binary fullword containing the relative record number of the specific record to be released.

Omit this parameter to release all records currently held by this process (for the specified file).

Error Conditions:

PIB-FUNCTION	The File is not assigned to the program.
PIB-NOT-HELD	The specified record was not held.

3.14.8. FCS-OPEN — Direct: Open File

Make the specified file available for processing by programs at the calling terminal. TIP/30 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 TIP/30 generation parameters specify the file as "OPEN=NO", TIPFCS will physically attempt to OPEN the file by issuing a Data Management OPEN.

This function is needed only for files which are not implicitly opened as result of the FILES= keyword in the program's TIP/30 Catalogue entry.

For the FCS-OPEN function to be successful, the file to be opened must be:

- defined in the TIP/30 job control stream — user data files are normally defined in the job control Proc named "TIPDATA"
- defined in the TIP/30 Catalogue (this is where the connection is made between a logical file name (LFN) and the physical file name (LFD)
- defined in the TIP/30 Generation parameters (see the FILE generation statement).

Syntax:

```
CALL  'TIPFCS'  USING  FCS-OPEN
                       file-pkt
                     [ file-desc ]
```

Where:

FCS-OPEN Function code from the TC-FCS copy element.

file-pkt Logical file name packet.

file-desc File descriptor packet — see separate description of the copy element TC-FDES. If this parameter is omitted, TIP/30 uses the name in the file-pkt to build a file descriptor.

Error Conditions:

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.

3.14.9. 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 copy element.

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-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/30 key holding table. This may be a result of not issuing a prior FCS-GETUP to lock the record for update or TIP/30 has discarded the previously acquired record lock (see discussion of record locking elsewhere in this manual).

3.15. TIPFCS for Sequential Files

This section describes TIP/30 file control system operations for sequential files. Sequential files include:

- Sequential MIRAM
- IRAM
- PRINT
- PUNCH
- TAPE.

A sequential file must be designated in the TIP/30 generation parameters 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/30 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/30 File Control System (TIPFCS).

The operating system's spooling facilities normally process printer and punch files. This spooling activity is transparent to FCS.

Print files use a standard Unisys variable length print line. The layout of the print line is the same as the layout required by the TIP/30 printing interface "TIPPRINT". (Refer to that section of this documentation or the copy element TC-PLINE).

3.15.1. FCS-CLOSE — Sequential: Close File

The FCS-CLOSE function indicates that a program is relinquishing access to a file. TIP/30 removes the corresponding entry for the file from the Active File Table (AFT) of the issuing process.

If there are no other online users of the file and the file was generated with OPEN=NO, TIPFCS will physically CLOSE the file by issuing a "CLOSE" imperative macro to Data Management.

Syntax:

```
CALL  'TIPFCS'  USING  FCS-CLOSE
                       file-pkt
```

Where:

FCS-CLOSE

Function code from the TC-FCS copy element.

file-pkt Logical file name packet.

Error Conditions:

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). When a FCS-CLOSE function is issued for a PRINT or PUNCH file, the TIP/30 file system issues a breakpoint for the file.

3.15.2. 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/30 generation parameters 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 of that 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 copy element.
file-pkt	Logical file name packet.
record	Area where record data is to be placed.

Error Conditions:

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.

3.15.3. FCS-OPEN — Sequential: Open File

Make the specified file available for processing by programs at the calling terminal. TIP/30 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/30 generation parameters as "OPEN=NO", TIPFCS will physically OPEN the file by issuing a Data Management OPEN.

For the FCS-OPEN function to be successful, the file to be opened must be:

- defined in the TIP/30 job control stream — user data files are normally defined in the job control Proc named "TIPDATA"
- defined in the TIP/30 Catalogue (this is where the connection is made between a logical file name (LFN) and the physical file name (LFD)
- defined in the TIP/30 Generation parameters (see the FILE generation statement).

Syntax:

```
CALL  'TIPFCS'  USING  FCS-OPEN
                       file-pkt
                     [ file-desc ]
```

Where:

FCS-OPEN Function code from the TC-FCS copy element.

file-pkt Logical file name packet.

file-desc File descriptor packet — see separate description of the copy element "TC-FDES".

If omitted, the name in the file-pkt parameter is used to build a file descriptor.

Error Conditions:

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.

3.15.4. 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 may be interleaved.

Syntax:

```
CALL  'TIPFCS'  USING  FCS-PUT
                       file-pkt
                       record
```

Where:

FCS-PUT	Function code from the TC-FCS copy element.
file-pkt	Logical file name packet.
record	Record area containing data for the record to be added. 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. Refer to the description of the structure of print line records in the section describing TIPPRINT and the supplied copy element TIP/TC-PLINE.

Error Conditions:

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 an 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.

3.16. TIPFCS for Dynamic Files

TIP/30 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/30 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 8 characters long) — an example is: EDP/BATCH/007
- Dynamic files are allocated by TIP/30 from available blocks in the TIP$RNDM file.
- The TIP/30 generation parameter FCSEXTENT= determines the initial allocation of 512-byte blocks (default value: 40 blocks).
- Dynamic files may extend to include up to a maximum of:

```
(48 extents)  x (FCSEXTENT= number of blocks)
              x (512 bytes per block)
```

- 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.
- When dynamic files are created, the application program is responsible for initializing the allocated blocks (blocks in the TIP$RNDM file are reused as needed, but they are not initialized by TIPFCS).
- 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 files support the following functions (the function names refer to function codes defined by the FCS copy element TC-FCS).

Table 3-8. TIPFCS Functions for Dynamic Files

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.

3.16.1. 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 copy element.

file-pkt Logical file name packet

file-desc File descriptor packet. Refer to the description of the copy element 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          PIC X(9).
02  TAXTABLE-FDES.        COPY TC-FDES OF TIP.
    ...
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-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.

3.16.2. 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 copy element.

file-pkt Logical file name packet

file-desc File descriptor packet.

Refer to the description earlier of the copy element TC-FDES.

Example:

```
CALL 'TIPFCS' USING  FCS-ASSIGN
                     DATA-ENTRY-BATCH-FILE
                     DATA-ENTRY-BATCH-FDES.
```

Error Conditions:

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.

3.16.3. 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 copy element.

file-pkt Logical file name packet.

Error Conditions:

PIB-FUNCTION	The file is not assigned to the program.

3.16.4. 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 userid 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-FILE-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 copy element.

file-pkt Logical file name packet

file-desc File descriptor packet.

Refer to the earlier description of the copy element TC-FDES.

Error Conditions:

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.

3.16.5. FCS-GET — Dynamic: Read Record(s)

Records in FCS Dynamic files are referenced by relative record number. The program specifies a relative record number (as a fullword) to read.

If the optional parameter REC-COUNT is specified, FCS will read 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 will read 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.

Note: *When TIPFCS returns blocks to the calling program, the 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 copy element.
file-pkt	Logical file name packet
record	Record area with a size 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.

Example:

```
05  LFN-PKT          PIC X(9).
05  REL-REC-NUM      PIC 9(7) COMP SYNC.
05  REC-COUNT        PIC 9(7) COMP SYNC.
05  DYN-REC.
  10  FILLER         PIC 9(7) 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
                        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.

3.16.6. 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/30 catalogue 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 copy element.

file-pkt Logical file name packet

file-desc File descriptor packet.

Refer to the earlier description of the copy element TC-FDES.

Error Conditions:

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.

3.16.7. 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.

Files are expanded to the dynamic file upper limit:

```
(48 extents) x (FCSEXTENT= blocks) x (512 bytes)
```

FCSEXTENT= is a TIP/30 generation parameter that normally defaults to 40.

Syntax:

```
CALL  'TIPFCS'  USING  FCS-PUT
                       file-pkt
                       record
                       rec-num
                     [ rec-count ]
```

Where:

FCS-PUT Function code from the TC-FCS copy element.

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.

3.16.8. 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/30 terminates abnormally or if the transaction aborts; one of the PMDA processing functions issues a FCS-SCRATCH function for any temporary dynamic files that it finds in the aborting program's Active File Table (AFT).

Syntax:

```
CALL  'TIPFCS'  USING  FCS-SCRATCH
                       file-pkt
```

Where:

FCS-SCRATCH

Function code from the TC-FCS copy element.

file-pkt Logical file name packet

Error Conditions:

PIB-FUNCTION The file is not assigned to the program.

PIB-WRONG-MODE The file is not a dynamic file.

Additional Considerations:

After TIP/30 scratches the dynamic file, it frees the blocks of the TIP$RNDM file formerly occupied by the file.

Note: *TIP/30 does not erase the contents of the blocks — if the file contained sensitive information, the program could first rewrite all records with low-values and then use FCS-SCRATCH to erase the file.*

3.17. TIPFCS for Edit Buffers

The text editors supplied with TIP/30 do all editing in a special purpose FCS dynamic file called an "edit buffer".

TIP/30 uses a two part name (unlike dynamic files that have three part names) 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 created the edit buffer. This is the assumption made by the TIP/30 text editors; however, this is not a hard and fast rule.

TIPFCS maintains the file structure used for edit buffers within a TIP/30 permanent dynamic file. An edit buffer consists of:

- a control record (block #1)
- index blocks
- data blocks.

Although edit buffers may define their own record length, the most popular record length (used by the TIP/30 editors) is 85 bytes (80 bytes of data; followed by 5 bytes of control information). Each data block (512 bytes) may hold up to six 85-byte records. The first 80 characters are the data in the line image from the library. The 81st byte is a binary version number.

Note: *TIP/30 reserves bytes 82 through 85. Records in an edit buffer are accessed by a line number relative to one.*

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).

User-written TIP/30 programs may find this file structure convenient in applications where they must create a dynamic file for purposes of manipulation in a line by line manner.

3.17.1. 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 copy element.

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-EOF	This error status indicates that the edit buffer is full.
PIB-NOT-FOUND	The edit buffer is not assigned to the process.

Additional Considerations:

The record is written to the file at the specified position. TIP/30 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.

3.17.2. 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.

Note: *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 copy element.

file-pkt Logical file name packet.

Error Conditions:

PIB-NOT-FOUND The edit buffer is not assigned to the process.

3.17.3. 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 copy element.

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-NOT-FOUND The edit buffer is not assigned to the process.

Additional Considerations:

TIP/30 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!

3.17.4. 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.

The FCS-FLUSH function forces TIPFCS to update all modified blocks. This function must be the last function used prior to a FCS-CLOSE when the program has modified the edit buffer.

Syntax:

```
CALL  'TIPFCS'  USING  FCS-FLUSH
                       file-pkt
```

Where:

FCS-FLUSH

Function code from the TC-FCS copy element.

file-pkt Logical file name packet.

Error Conditions:

PIB-NOT-FOUND The edit buffer is not assigned to the process.

3.17.5. 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 copy element.
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.

Error Conditions:

PIB-EOF	The record number is out of bounds.
PIB-NOT-FOUND	The edit buffer is not assigned to the process.

3.17.6. FCS-OPEN — Edit: Open Buffer

TIP/30 Edit Buffers are a specific implementation of an access method that uses TIP/30 dynamic files as the storage medium. Edit buffers are "line-oriented" in the sense that they manipulate "lines" of data. The most common implementation of edit buffers (used by TIP/30 editors) specifies a line length of 85 characters. TIP/30 always references the lines in an edit buffer by positive whole numbers that range from one in increments of one.

Syntax:

```
CALL  'TIPFCS'  USING  FCS-OPEN
                       file-pkt
                       file-desc
                     [ buffer      ]
                     [ num-buffers ]
                     [ line-length ]
```

Where:

FCS-OPEN Function code from the TC-FCS copy element.

file-pkt Logical file name packet.

file-desc File descriptor packet (see copy element TC-FDES). The example which 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/30 free memory pool (TIP/30 generation parameter FREEM=).

If this parameter is provided, it must represent a FULLWORD aligned area of at least 3x512 bytes — also see description of the following parameter.

Note: *Omit this (and following parameters) if the program is to open the edit buffer with the intention of shared read access.*

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 85.

Note: *The TIP/30 editors default to creating edit buffers that have a line length of 85 characters (80 bytes of user data plus 5 bytes of control information).*

Error Conditions:

PIB-NOT-FOUND	The edit buffer is not assigned to the process.
PIB-DUP-AFT-NAME	The logical file name is already in use by the process.

Example:

```
... in the program's WORKING-STORAGE ...
77  NUM-BUFFERS              PIC 9(2) COMP SYNC VALUE 3.
... in the program's work area ...
02  EDIT-BUF-DESC.  COPY TC-FDES OF TIP.
  05  EDIT-BUF-LFN             PIC X(9).
  05  EDIT-WORKAREA.
      10  EDIT-BUFFER-WORD1    PIC 9(7) COMP SYNC.
      10  EDIT-BUFFER-WORD1R REDEFINES EDIT-BUFFER-WORD1.
          15  FILLER           PIC X(2).
          15  EDIT-LINES       PIC 9(4) COMP.
      10  FILLER               PIC X(508).
      10  FILLER               PIC X(512).
      10  FILLER               PIC X(512).
      10  FILLER               PIC X(512).
    ... in the PROCEDURE DIVISION ...
MOVE 'WORKFIL'         TO EDIT-BUF-LFN
MOVE 'EDP'             TO FDES-USERID
MOVE 'SOMEDATA'        TO FDES-CATALOG
                          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.

3.17.7. 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 copy element.
file-pkt	Logical file name packet.
record	Record area.
line-num	Binary fullword holding the relative record number to be replaced.

Error Conditions:

PIB-EOF	The line number is out of bounds.
PIB-NOT-FOUND	The edit buffer is not assigned to the process.

3.17.8. 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 copy element.

file-pkt Logical file name packet identifying the edit buffer.

Error Conditions:

PIB-NOT-FOUND The edit buffer is not assigned to the process.

Additional Considerations:

TIP/30 creates edit buffers as "permanent" dynamic files — this prevents their disappearance when a system crash occurs. Since they are permanent dynamic files, they must be explicitly scratched to erase them from the TIP$RNDM file. The TIP/30 CAT program (Catalogue Manager) and the TIP/30 text editors can also be used to discard (scratch) edit buffers.

3.18. TIPFCS for Library Files

TIP/30 programs may access library elements stored in a SAT format library. TIP/30 supports limited (read) access to load or object modules and permits (read) access to the directory of a library.

Note: *TIP/30 does not support access to MIRAM libraries.*

Library elements contain lines of data that are up to 128 bytes in length. TIP/30 assumes that the record area that is designated by the program for read or write operations is 128 bytes long.

TIP/30 supports the following function codes for library elements:

Table 3-9. Functions for Library Access

Function	Description
FCS-OPEN	Open file/element.
FCS-GET	Get next input record (line).
FCS-PUT	Output next output record (line).
FCS-CLOSE	Close library element. If reading: de-access file. If writing: the old module is flagged "deleted" and the latest element flagged "active".
FCS-NOUP	Close file (cancel update). If reading: de-access file. If writing: old module is not flagged deleted.

3.18.1. Library File Descriptor

The layout of the FILE-DESCRIPTOR packet for library files is described in the copy element TC-FDES in the TIP/30 library:

```
       01  LIB-FDES.      COPY TC-FDES OF TIP.
*****************************************************************
*                 FCS FILE DESCRIPTOR PACKET                    *
*****************************************************************
       05  FDES-USER-ID                PICTURE X(8).
       05  FDES-CATALOG                PICTURE X(8).
       05  FDES-FILE-NAME              PICTURE X(8).
       05  FDES-PASSWORD               PICTURE X(8).
       05  FDES-FCS-CLASS              PICTURE X.
       05  FDES-FCS-TYPE               PICTURE X.
       05  FDES-FCS-PERM               PICTURE X.
       05  FDES-FCS-LOCK               PICTURE X.
*****************************************************************
*        ADDITIONAL FIELDS FOR LIBRARY ELEMENT ACCESS           *
*****************************************************************
       05  FDES-ELEMENT                PICTURE X(8).
       05  FDES-COMMENTS               PICTURE X(30).
       05  FDES-DATE                   PICTURE X(8).
       05  FDES-TIME                   PICTURE X(5).
*****
```

Where:

FDES-USERID

The group name (or user name) associated with the TIP/30 catalogue entry for the library file.

If this field is spaces or low-values, TIPFCS will perform a "standard order of search" for the correct catalogue entry to reference.

FDES-CATALOG

Logical file name for the library (as specified in the TIP/30 catalogue).

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.

FDES-FILE-NAME

Logical file name for the library (as specified in the TIP/30 catalogue).

FDES-FCS-CLASS

S	Search only for an OS/3 file (this is the preferred value for opening library elements).

Space	(or low-value). The catalogue search may "find" an edit buffer or dynamic file.

FDES-FCS-TYPE

Library element type codes. TIP/30 supports the following values:

S	Source module.
M	Macro or proc.
I	Internal symbol dictionary for load module.
D	Read detailed directory of file.
F	Read fast directory of file (no comments or time stamp).
Note:	*Types I, D, and F are read-only access.*

FDES-FCS-PERM

Specified when the element is opened.

If read access desired, set to "R".

If write access desired, set to "W".

The following fields are uniquely used for access to library elements:

FDES-ELEMENT

Element (module) name within library.

This field contains the actual element name for "S" or "M" elements.

This field may contain a prefix if the type is "D" or "F" (directory retrieval).

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

3.18.2. 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. TIP/30 always (physically) closes library files — they do not remain open.

Syntax:

```
CALL  'TIPFCS'  USING  FCS-CLOSE
                       file-pkt
```

Where:

FCS-CLOSE

Function code from the TC-FCS copy element.

file-pkt Logical file name packet.

Error Conditions:

PIB-FUNCTION	File is not assigned to the program.

3.18.3. 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 copy element.
file-pkt	Logical file name packet.
record	Record area where line data is placed. The record area is a fixed size of 128 bytes.

Error Conditions:

PIB-FUNCTION	The file is not assigned to the program
PIB-IO-ERROR	An I/O error occurred on the disk.
PIB-EOF	The end of the element has been reached.
PIB-WRONG-MODE	The library file was not opened for input processing.

3.18.4. 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 activated — the previous version of the element, if any, will remain the current element.

Syntax:

```
CALL  'TIPFCS'  USING  FCS-NOUP
                       file-pkt
```

Where:

FCS-NOUP Function code from the TC-FCS copy element.

file-pkt Logical file name packet.

Error Conditions:

PIB-FUNCTION File is not assigned to the program.

3.18.5. 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). TIPFCS issues a Data Management OPEN for the library and attempts to make the specified element available.

Syntax:

```
CALL  'TIPFCS'  USING  FCS-OPEN
                       file-pkt
                       file-desc
```

Where:

FCS-OPEN Function code from the TC-FCS copy element.

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 OF TIP.
02  LIB-LFN    PIC X(9).
02  LIB-REC    PIC X(128).

... 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-CLASS
MOVE 'TC-FDES'    TO FDES-ELEMENT
CALL 'TIPFCS' USING FCS-OPEN
                    LIB-LFN
                    LIB-FDES
```

The above example opens the library element "TIP/TC-FDES" for read operations (FDES-FCS-PERM).

Note: *The name in the LFN file packet ("INFILE") can be any name the programmer chooses — the TIP/30 file system uses the name to determine which file the program is referring to during subsequent calls to TIPFCS.*

Error Conditions:

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.
	Note: *This is a warning that 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	Some other TIP/30 application program or batch job has exclusive access to the library.

3.18.6. FCS-PUT — Library: Write Line

The FCS-PUT function will output the next line (sequential fashion) to the library element.

Syntax:

```
CALL  'TIPFCS'  USING  FCS-PUT
                       file-pkt
                       record
```

Where:

FCS-PUT	Function code from the TC-FCS copy element.
file-pkt	Logical file name packet.
record	128 character record area.

Error Conditions:

PIB-FUNCTION	The File is not assigned to the program.
PIB-IO-ERROR	An I/O error occurred on the file.
PIB-EOF	The file is full.
PIB-WRONG-MODE	The library file was not opened for output processing.

3.19. TIP/30 Print Facility (TIPPRINT)

TIP/30 native mode programs may call the reentrant subroutine TIPPRINT to perform printing functions. TIPPRINT directs print lines to any of the following "destinations":

- a batch print file (via the OS/3 spooling system)
- an OS/3 sequential output file
- an auxiliary (communications) printer
- an MS-DOS file (assuming that the terminal is a personal computer running with appropriate terminal emulation software and hardware)
- a terminal (in full screen display mode).
- A UNIX system (assuming that there is an appropriate interface between the OS/3 system and a UNIX machine)
- An OFIS Link/80 file (assuming that the OFIS Link/80 interface is installed).

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. TIPPRINT uses the filename (2nd parameter) to determine the destination of the print file (illustrated in the list above).

TIPPRINT deals with variable length print lines that must contain a (device independent) carriage control code (see the Unisys publication *OS/3 BASIC DATA MANAGEMENT USER GUIDE [UP-8068] Table 7-1, Device Independent Control Character Codes*).

If a user program needs to print only on the main site printer and the application does not need the capability to print on an auxiliary device printer, use direct calls to the TIP/30 File Control System (FCS). When the destination is a batch printer file TIPPRINT issues calls directly to TIPFCS.

A TIP/30 native mode program issues calls to the TIPPRINT subroutine to perform the following functions:

OPEN Initiate 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.

Additional Considerations:

TIP/30 sends all auxiliary device messages generated by TIPPRINT to the printer via the ICAM MEDIUM terminal queue.

Generate ICAM with FEATURES=(OPCOM,OUTDELV) to support TIPPRINT auxiliary printing. TIPPRINT always waits for delivery notification from ICAM when performing auxiliary device I/O. Since TIPPRINT is usually buffering print lines (to send as a single message rather than several little messages), the user program should avoid having a serial resource locked when calling TIPPRINT.

The program may not have a file (or files) in SEQUENTIAL mode (SETL) or be imparted to a data base when issuing a call to TIPPRINT.

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.

3.19.1. TIPPRINT Print Destinations

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.

3.19.1.1. ROLL — Single Line Terminal Output

Specifying the character string "ROLL" as a destination tells TIPPRINT to "roll" out 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/30 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.

3.19.1.2. 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** if you wish to see the next screen full of information
- **No** to return PIB-BREAK status to the calling program and thus halt printing as soon as possible.

AUX0 is the default print destination for many of the TIP/30 utility programs that display information on the terminal. If you specify this destination for an IBM-327x style terminal, TIP/30 changes it to "ROLL" since the 327x does not support this type of output.

3.19.1.3. AUXn — Auxiliary Device

You may direct TIPPRINT output to a terminal auxiliary device (typically a printer):

```
AUXnTTTT
```

You can, theoretically, connect AUX devices (for example, printers, cassettes, or diskettes) on any one of a number of auxiliary interfaces (numbered 1 through F). Specify the device number **n** (1 through F) immediately following the string "AUX". The last four characters **TTTT** represent the desired terminal name.

Note: For historical purposes, TIP/30 considers the strings "COP" and "DCT" to be synonyms of "AUX".

The default terminal name (if the TTTT field is spaces or low values) is the terminal that is calling TIPPRINT. You can route output to any other terminal in the network by placing the desired terminal name in the TTTT field.

TIPPRINT recognizes the following special character strings as reserved terminal names:

***BYP** The bypass terminal (as determined by searching the TIP/30 CLUSTER generation statements — see the CLUSTER statement in *TIP/30 Generation, Maintenance and Installation, ARP-600-05.*

***MST** The master terminal of the calling terminal's cluster. See *BYP above.

TIP/30 ignores the TTTT specification if either of the above special names cannot be resolved. The output is routed to the calling terminal.

For output sent to an auxiliary device, TIP/30 first displays the data on the terminal associated with the auxiliary device (TTTT). TIPPRINT places an appropriate terminal

control code at the end of each screen full of data to cause the terminal to PRINT its display contents on the indicated aux device.

Some terminals have a "bypass printer" mechanism that consists of memory that holds the data destined for the "screen". The advantage of this mechanism is that it is relatively transparent to the terminal operator; printing utilizes the bypass memory rather than tying up the operator's display.

Note: *Data that will be printed must first be output to the "screen". The terminal must advance to the next line on the screen when it receives a carriage-return character. This implies that auxiliary printers cannot support the traditional "print without spacing".*

TIPPRINT simulates the ability to print without advancing the paper (sometimes called "overprinting"), by enforcing the following rules when such a print line is sent to an AUX device:

- A print line that specifies "print with no spacing" is NOT discarded, but the data will be merged with the next line that specifies "print with spacing"
- Underscore characters override spaces
- Printable characters override spaces and underscores
- The last data received overrides previous data.

Example of Print With No Space:

```
Line #1 (print no space):    "This is a test line              "
Line #2 (print no space):    "_________________________________"
Line #3 (print & skip 1)"    "          TEST                 001"
RESULT PRINT LINE:           "This_is_a_TEST_____________001"
```

In previous releases of TIPPRINT, lines containing carriage control codes specifying "print with no space" were discarded!

3.19.1.4. MS-DOS File — d:xxxxx

TIPPRINT can output "print lines" to a terminal that is a personal computer (PC) with the appropriate combination of hardware and software:

- a Unisys Terminal Emulation Package (STEP) or
- Computer Logic's Personal Emulator Package (PEP). To support this, TIPPRINT recognizes a printer destination of the form:

```
d:xxxxxx
```

Where:

- **d** is the MS-DOS drive designator
- **:** is literally a colon character
- **xxxxxx** is the desired MS-DOS filename.

Since the filename packet is limited to eight characters, TIPPRINT automatically supplies an MS-DOS extension of "PRN". Filenames are, thus, limited to six characters.

To specify a larger filename, or to create a different file extension, specify the complete MS-DOS filename by filling in the appropriate fields in the INFO-PKT during the FCS-OPEN call to TIPPRINT.

> **WARNING**
>
> Due to present limitations in the STEP and PEP packages, you CANNOT specify MS-DOS path information for this type of file transfer. PEP/STEP creates the filename in the active directory of the disk drive specified.

TIPPRINT transfers data to the MS-DOS file in a manner similar to that used for auxiliary printing: the data is output to the screen and then transferred to the appropriate MS-DOS file. TIPPRINT fills in the correct information in the PC control page (STEP or PEP). This type of data transfer allows the transmission of displayable graphics characters only.

There is no provision for specifying which display to use to transfer data to the PC. TIPPRINT does not alter the terminal index number in the STEP or PEP control page — TIPPRINT assumes that the transfer is taking place on the correct rid/sid.

Note: Users of the Computer Logics PEP board (and the Unisys STEP board) are aware that the file transfer capability does NOT provide any way to specify a complete MS-DOS filename — the MS-DOS file name is assumed to be in the current subdirectory for the specified drive.

This restriction can be circumvented by making use of the MS-DOS SUBST command. The SUBST command allows you to designate a pseudo drive name for a specific path name. In some versions of MS-DOS, you can also redirect a real drive name to a different directory path.

Refer to the appropriate MS-DOS documentation for the specific details of the SUBST command for your PC. The following example illustrates how to use this MS-DOS command.

Example:

```
MS-DOS:
        SUBST H: C:\BUDGET\1988
```

The example above substitutes the MS-DOS drive name H: for the path C:\BUDGET\1988. By using drive H: in the TIPPRINT destination specification, MS-DOS creates the file in the desired directory.

There may be revision levels of PEP and STEP that do not recognize some MS-DOS drive names as valid drives. There are also restrictions that some versions of MS-DOS impose on the operation of "drives" specified with a SUBST command — refer to the appropriate PC documentation.

Some versions of MS-DOS require adjustment of the LASTDRIVE= specification in the CONFIG.SYS file to accommodate drive names used in a SUBST command.

3.19.1.5. PRNTR — Batch Printer(s)

TIPPRINT can output "print lines" to a standard OS/3 file. The filename specified in the file packet is a TIP/30 logical filename. You must define the filename in the TIP/30 catalogue.

If the file resolves to be a print file, the print lines will be sent to the OS/3 spooling system. TIP/30 automatically breakpoints the print file when a FCS-CLOSE is issued to TIPPRINT.

Note: *You may generate more than one print file in the TIP/30 system — "PRNTR" is automatically defined in the TIP/30 system.*

3.19.1.6. SMIRAM File — Output to File

A standard logical file name can be used as a TIPPRINT destination to route print lines to an output sequential file.

If the filename resolves to be an OS/3 data management file, TIPPRINT issues an FCS-PUT function to the file with the print line (including the five-byte header information) as the data. This works best if the file is a sequential MIRAM output file.

3.19.1.7. OFIS Link/80

The reserved names "WORKING" and "IN-MAIL" represent OFIS Link/80 destinations. TIPPRINT routes the print lines to the appropriate OFIS Link/80 file.

Note: The program may place information in the INFO-PKT when FCS-OPEN is issued to specify a OFIS Link/80 drawer and folder other than the special names "WORKING" or "IN-MAIL".

To use this facility, you must install (and correctly configure) the TIP/30 — OFIS Link/80 interface software Refer to the following TIP/30 publications:

- *OFIS Link/80 Administration Guide — ARP-607-00*
- *OFIS Link/80 User Guide — ARP-612-00.*

3.19.1.8. UNIX: — UNIX Files

TIPPRINT can output "print lines" to a file on a Unisys UNIX system (Series 5000 machine). TIPPRINT will route the data to the receive file that is designated on the UNIX side of the interface. To utilize this facility, you must properly set up the UNIX machine to support file transfer and you must be logged on to TIP/30 using the UNISCOPE Emulator.

For further information, consult the latest version of the publication *Unisys 5000 and 7000 Series System Communication UNISCOPE Emulation Guide, UP-11869 (Series 5000 and 7000 System Library).* The section titled "FILE TRANSFER OPERATION" contains a brief outline of the necessary steps.

Note: The specification of the receiving filename is done on the UNIX side of the interface; hence, you may only specify the reserved name "UNIX:" in the TIPPRINT filename packet.

3.19.2. 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 copy element TIP/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.

Refer also to description of TIPPRINT destinations in "3.19.1. TIPPRINT Print Destinations" on page 3-105.

The filename may be the name of a batch print file (for example, PRNTR) or may be the name of an auxiliary print device.

If the first four characters of the filename are "ROLL", TIPPRINT outputs print lines by passing the data to the TIP/30 subroutine "ROLL" (output one line at a time after moving the screen contents up one line).

If the first four characters of the filename are "AUX0", TIPPRINT accumulates print lines and display the print lines one screen full at a time and prompt the terminal operator for continuation.

If the first three characters of the filename are "AUX", "COP" or "DCT", TIPPRINT routes printing to the auxiliary printer. In this case, TIPPRINT uses the fourth character of the filename to specify the auxiliary device number (usually "1" through "F").

You may use the trailing four characters of the filename to indicate a specific terminal name for the auxiliary I/O data transfer.

The default terminal is the terminal being used by the program calling TIPPRINT.

Use the terminal name "*BYP" to direct output to the bypass terminal of the associated CLUSTER.

Use the terminal name "*MST" to direct output to the master terminal of the associated CLUSTER.

TIPPRINT also can transfer print lines to an MS-DOS file. To do this, specify the filename as an MS-DOS filename, for example:

```
X:YYYYYY
```

"X" is the diskette drive designation (A: thru Z:) and "YYYYYY" is a six character filename.

TIPPRINT sets the MS-DOS file extension to ".PRN"

To create an MS-DOS file, the terminal must be an IBM compatible PC that has the appropriate hardware and software to support UNISCOPE terminal emulation and file transfer (either the Unisys "STEP" package or the COMPUTER LOGICS "PEP" package).

INFO-PKT

Information packet required ONLY on the call to TIPPRINT with the FCS-OPEN function.

The supplied copy element TIP/TC-PRINT defines the format of the information packet:

```
       05 INFO-PKT    COPY TC-PRINT OF TIP.
*
**  COPY ELEMENT FOR TIPPRINT INFORMATION PACKET
*
          10  PRINT-BUF-LEN                PICTURE 9(4) COMP-4 SYNC.
          10  PRINT-PAG-LEN                PICTURE 9(4) COMP-4 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-SPERRYLINK-FILE-NAME
              REDEFINES PRINT-FULL-FILE-NAME.
              15  PRINT-DRAWER             PICTURE X(8).
              15  PRINT-FOLDER             PICTURE X(12).
          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            OCCURS 15 TIMES
                                            PICTURE 9(4) COMP-4.
```

Where:

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.

If this field is set to zero (or spaces) the default value is the value specified by the PRINTLPP= keyword in the TIP/30 generation parameters.

Note: *This keyword is not applicable when the destination printer is a batch device (such as PRNTR). In such cases, Data Management determines the overflow status according to the VFB information that is defined for the specified print file. TIPPRINT returns an overflow status when it receives such an indication from Data Management.*

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:

- The name of a valid terminal in the network
- The value "*CON" to indicate the OS/3 operator console or,
- The value "*RET" to indicate that no error message is to be sent.

  In the case of "*RET", TIPPRINT will not output any error message and will simply return to the calling program with PIB-BREAK status in the PIB.

  Specify the reserved terminal name "*MST" to cause the error message to be routed to the MASTER terminal of the associated terminal CLUSTER.

  Specify the reserved terminal name "*BYP" to cause the error message to be routed to the BYPASS terminal of the associated terminal CLUSTER.

PRINT-TOP-OF-FORM

Specify as either "Y" or "N" or space.

Y	TIPPRINT automatically forces a skip to top of form before starting any printing.
N	TIPPRINT does not automatically skip to top of form.
space	TIPPRINT takes the default as specified by the PRINTTOF= keyword in the TIP/30 generation parameters.

PRINT-LINE-FEED

Specify as either "Y" or "N" or space.

Some communications 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.

Set this field to a space to indicate that the default is to be used:

- PRINTLF= in the TIP/30 TIPGEN parameter or
- The PRINTLF= keyword in a CLUSTER statement associated with the terminal.

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 (by pressing MSG-WAIT) because the last activity at the terminal was an input message and ICAM will not allow two input messages without an intervening output message (issuing the NOW PRINTING message meets this requirement!).

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 upper case.
N	Indicates that no translation is to occur.
space	Indicates that the system default is to be taken (as specified in the PRINTUC= keyword of the TIPGEN parameters).

PRINT-RESERVED

This field is reserved for future use and is currently not examined by TIPPRINT.

PRINT-VFB-INFO

If you set this field to a "Y", TIPPRINT expects to find valid VFB information in the field PRINT-VFB-CHANNEL (see description following).

Any other value in this field implies that TIPPRINT may ignore the contents of PRINT-VFB-CHANNEL.

PRINT-FULL-FILE-INFO

Set this field to an "S" to indicate that TIPPRINT is to route the print lines to a OFIS Link/80 document.

TIPPRINT expects the program to set the fields PRINT-DRAWER and PRINT-FOLDER to the desired OFIS Link/80 filename information.

Using "S" in this field allows the program to output print lines to a OFIS Link/80 document other than "WORKING" or "IN-MAIL"; you may specify either of these by placing the appropriate name in the FILE-PKT.

Set this field to a "D" to indicate that TIPPRINT is to route the print lines to an MS-DOS file on a PC.

If you wish to create an MS-DOS file with a filename that is no more than six characters and has a file extension of "PRN", simply put the MS-DOS filename in the FILE-PKT (for example, A:TEST).

TIPPRINT expects that you will set the fields PRINT-MSDOS-DRIVE, PRINT-MSDOS-FILE and PRINT-MSDOS-EXTENSION to the desired MS-DOS filename information.

Use a "D" in this field to allow your program to output print lines to an MS-DOS file with a full eight character filename and/or a file extension other than "PRN".

Note: *Restrictions in the STEP/PEP software make specification of MS-DOS path names impossible; the file will always be written to the active directory on the specified drive. See also discussion in* *"3.19.1. TIPPRINT Print Destinations" on page 3-105.*

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 userid 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 depending on the setting of the TIP/30 generation parameter PRintTTL=.

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

See prior description of PRINT-FULL-FILE-INFO.

PRINT-VFB-CHANNEL

See prior description of PRINT-VFB-INFO.

This item is an array of 15 binary HALFwords that TIPPRINT may reference if, and only if the field PRINT-VFB-INFO contains a "Y". Each halfword corresponds to channels 1 through 15 of a virtual carriage control tape. The value in the item represents a 1-relative line number that corresponds to that channel number.

If a print line contains a skip code such as "print and skip to channel #n", TIPPRINT will use the corresponding entry in this array to determine which relative line number to advance to after printing the line. Using skip codes that refer to array entries which contain invalid entries result in a skip to the home position.

Note: *TIPPRINT simply computes the appropriate number of LINE-FEED characters required to move from the current line on the page to the desired line on the page.*

Example: PRINT-VFB-CHANNEL (5) contains 37 and the program is currently "at" line 12 on the page. TIPPRINT will react to a skip code of "print and skip to channel 5" by outputting the print line (as it normally would) followed by enough line feed characters to move to line 37.

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 1,024 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.

The third byte of the buffer will contain the ICAM status code if an "unrecoverable" error occurs during printing to an auxiliary device. TIPPRINT uses all other areas of the print buffer in ways that may change from release to release.

WARNING

Programs should not make any assumptions about any observed contents of this buffer!

You may copy the supplied copy element TIP/TC-PBUFR into your program's work area to define the buffer:

```
        COPY  TC-PBUFR  OF  TIP.
*
*                    TIPPRINT BUFFER PACKET
*     ---   USER PROGRAM SHOULD NOT MODIFY THESE FIELDS   ---
*
      05  TIPPRINT-BUF.
          10  FILLER                PICTURE 9(8)  COMP-4  SYNC.
          10  FILLER                PICTURE X(2556).
      05  TIPPRINT-BUFFER  REDEFINES  TIPPRINT-BUF.
          10  BU-PAGE-LENGTH        PICTURE 9(4)  COMP-4  SYNC.
          10  BU-ICAM-STATUS        PICTURE X.
          10  FILLER                PICTURE X(2557).
*
```

Where:

BU-PAGE-LENGTH

While the TIPPRINT interface is open, this field contains the number of lines per page that TIPPRINT has determined from the INFO packet or from the CLUSTER or other generation parameters.

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:

Character	Description
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-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/30 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.

3.19.3. 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 (refer also to the description of the FCS-FLUSH function call to TIPPRINT).

TIPPRINT buffers print lines when they are destined for a communications or auxiliary printer or an MS-DOS file — TIPPRINT simply calls standard TIPFCS routines to send output to a Data Management printer file like PRNTR.

Syntax:

```
CALL  'TIPPRINT'  USING   FCS-PUT
                          FILE-PKT
                          PRINT-LINE
                          BUFFER.
```

Where:

FCS-PUT

This function code (normally defined via the supplied copy element TIP/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 copy element TIP/TC-PLINE to define this area:

```
       COPY  TC-PLINE  OF  TIP.
*
**  COPY ELEMENT FOR TIPPRINT LINE PACKET
*
     05  PRINT-LINE.
         10  LI-LENGTH                    PICTURE 9(4) COMP-4 SYNC.
         10  FILLER                       PICTURE XX.
         10  LI-DI-CONTROL                PICTURE X.
             88  LI-DI-HOME    VALUE ='27'.
             88  LI-DI-SPACE1  VALUE ='01'.
             88  LI-DI-SPACE2  VALUE ='02'.
             88  LI-DI-SPACE3  VALUE ='03'.
         10  LI-DATA                      PICTURE X(132).
```

Where:

PRINT-LINE

A variable length record containing a length field, a DI code (for carriage control), and the data to be printed.

The above copy element 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.

Note: *The length specified must include the five bytes preceding the actual data. In the copy element 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.

Note: *The minimum specification for this field is a value of five bytes. Some carriage control codes cannot specify data at the same time; ie: 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 spacing codes may be used if TIPPRINT is printing on a terminal auxiliary device although it may be better to use the codes as described in the *OS/3 BASIC DATA MANAGEMENT USER GUIDE [UP-8068] Table 7-1 — Device Independent Control Character codes.*

Standard FORTRAN skip codes are:

space	Single space
0	Double space
-	Triple space
1	Skip to the top of a new page

TIPPRINT recognizes a special DI-code — "B".

This code indicates that TIPPRINT is to output the data portion of the print line using BLOCK characters (similar to an OS/3 WRTSML function).

TIPPRINT recognizes a carriage return character (X'0D') as a signal to begin a new line of block characters. You may use this special DI-code to create custom title pages or separator pages.

TIPPRINT recognizes a special DI-code — "V" when the output is an auxiliary device. This code instructs TIPPRINT to 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 an auxiliary device to be able to directly control the device — some printers are designed to react to character sequences to perform advanced functions like bar code printing or double striking.

Note: *ICAM UNISCOPE protocol cannot allow certain characters to be sent to a terminal because their transmission as text would violate the rules of the protocol.*

Another consideration is that UNISCOPE terminals may react to certain escape sequences (character strings that begin with the ESC character) by performing built-in firmware functions (such as: clearing the screen, erasing to the end of the line, and so on) and may not permit escape sequences to be sent to the auxiliary printer.

To ensure that printer control codes can be handled, Personal Computer (PC) users who are using a PEP or STEP hardware/software interface can configure PEP or STEP to perform "screen to printer translation". This configuration option permits one or more characters to be translated into ESC before the data is sent to the auxiliary printer.

For example, the vertical bar character can be translated to the ESC character (X'1B'). To send the printer control sequence "ESC 3", the program places the string "|3" in the appropriate place in the print line data — PEP or STEP takes care of translating the "|" into an ESC character.

LI-DATA This field contains the text of the print line.

BUFFER The buffer that is assigned for TIPPRINT's use as described in the previous section TIPPRINT: open.

Additional Considerations:

The supplied copy element TIP/TC-DI defines some commonly used carriage control codes. Since this copy element contains VALUE clauses you must place it in the WORKING-STORAGE SECTION of your program.

```
       COPY  TC-DI  OF  TIP.
*
*       DEFINE CODES USED FOR PRINTER CARRIAGE CONTROL
*
     05  TC-DI-1                        PIC 9(4) COMP-4 VALUE 9985.
     05  TC-FILLER1 REDEFINES TC-DI-1.
         10  TC-DI-HOME                 PIC X.
         10  TC-DI-PRINT-SPACE1         PIC X.

     05  TC-DI-2                        PIC 9(4) COMP-4 VALUE 515.
     05  TC-FILLER2 REDEFINES TC-DI-2.
         10  TC-DI-PRINT-SPACE2         PIC X.
         10  TC-DI-PRINT-SPACE3         PIC X.

     05  TC-DI-3                        PIC 9(4) COMP-4 VALUE 1029.
     05  TC-FILLER3 REDEFINES TC-DI-3.
         10  TC-DI-PRINT-SPACE4         PIC X.
         10  TC-DI-PRINT-SPACE5         PIC X.

     05  TC-DI-4                        PIC 9(4) COMP-4 VALUE 1543.
     05  TC-FILLER4 REDEFINES TC-DI-4.
         10  TC-DI-PRINT-SPACE6         PIC X.
         10  TC-DI-PRINT-SPACE7         PIC X.

     05  TC-DI-5                        PIC 9(4) COMP-4 VALUE 2057.
     05  TC-FILLER5 REDEFINES TC-DI-5.
         10  TC-DI-PRINT-SPACE8         PIC X.
         10  TC-DI-PRINT-SPACE9         PIC X.

     05  TC-DI-6                        PIC 9(4) COMP-4 VALUE 2576.
     05  TC-FILLER6 REDEFINES TC-DI-6.
         10  TC-DI-PRINT-SPACE10        PIC X.
         10  TC-DI-PRINT-NO-SPACE       PIC X.
```

PIB-FULL TIP/30 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 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/30 will probably abort the program and issue a resources locked, waiting message.

This PIB-STATUS is applicable only to non-batch print destinations.

PIB-LOCKED

TIP/30 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/30 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/30 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.

Note: *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/30 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/30 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/30 also returns PIB-BREAK status if, for example, the printer is out of paper. Your program should take appropriate defensive action; it should probably should stop printing in any case.

If an I/O error occurs on an auxiliary device, TIP/30 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. The program can check the field BU-ICAM-STATUS (in the TIPPRINT buffer — see ("3.19.2. FCS-OPEN — Open TIPPRINT Interface" on page 3-111) to report potential ICAM errors.

3.19.4. 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.

Note: *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 copy element TIP/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 (TIPPRINT: open).

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 TIPPRINT: open.

Additional Considerations:

The FLUSH operation delivers any buffered print data that is in the TIPPRINT buffer (this would normally only occur when the buffer was 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:

TIP/30 may return a PIB-STATUS value of "PIB-BREAK" if there is a problem with the printer (see previous section TIPPRINT: Put).

3.19.5. 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). The close function breakpoints the printer file if the print file that TIPPRINT was using was a real Data Management Printer file (like PRNTR). If TIPPRINT does not encounter a CLOSE function call unpredictable results may occur — one real possibility is the loss of the last buffer of print lines.

Syntax:

```
CALL  'TIPPRINT'  USING   FCS-CLOSE
                          FILE-PKT
                          dummy
                          BUFFER.
```

Where:

FCS-CLOSE

This function code (normally defined via the supplied copy element TIP/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 TIPPRINT: open.

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:

TIP/30 returns a PIB-STATUS value of "PIB-BREAK" if there is a problem with the printer (see previous section TIPPRINT: Put).

3.20. PC File Transfer

TIP/30 provides a set of reentrant subroutines that TIP/30 native mode programs may call to transfer data to and from a Personal Computer (PC):

TIPH2P Copy from HOST to the PC.

TIPP2H Copy from PC to the HOST.

A TIP/30 native mode program that uses these routines must be running on a PC with the PEP (Personal Emulation Package from Computer Logics Inc.) or STEP (Unisys Terminal Emulation Package) hardware and software running UNISCOPE terminal emulation.

Note: Due to restrictions in these packages, it is not possible to simultaneously perform input and output operations at the same PC.

The user interface with PCXFER is similar to that used by TIPFCS calls (ie: function-code, filename, record). PCXFER, however, requires a fourth parameter; this parameter is always the name of a user supplied work area that PCXFER uses as a buffer. PCXFER uses the filename (2nd parameter) to indicate the MS-DOS filename (source or destination).

PCXFER handles variable length records. There is no maximum length for these records. A TIP/30 native mode program issues calls to the PC transfer subroutine 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.

Additional Considerations:

To support PC File Transfer from user written online programs:

1. Generate ICAM with FEATURES=(OPCOM,OUTDELV)
2. Specify the TIP/30 job control option PCXFER=YES.

PCXFER always waits for delivery notification from ICAM when performing file transfers. For efficiency, PCXFER buffers records to fill a screen; therefore, the program must be careful to avoid having a serial resource locked when calling TIPH2P or TIPP2H.

This implies that the program may not have a file (or files) in sequential mode (SETL)[16] or be imparted to a data base when a call is issued to TIPH2P or TIPP2H.

16. With the exception of files specified as MULTISEQ=YES in the TIP/30 Generation parameters.

3.20.1. 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).

Note: *Users of the Computer Logics PEP board (and the Unisys STEP board) are aware that the file transfer capability does NOT provide any way to specify a complete MS-DOS filename — the MS-DOS file name is assumed to be in the current subdirectory for the specified drive.*

This restriction can be circumvented by making use of the MS-DOS **SUBST** command. The SUBST command allows you to designate a pseudo drive name for a specific path name. In some versions of MS-DOS, you can also redirect a real drive name to a different directory path.

Refer to the appropriate MS-DOS documentation for the specific details of the SUBST command for your PC. The following example illustrates how to use this MS-DOS command.

Example:

```
SUBST H: C:\BUDGET\1988
```

The example above substitutes the MS-DOS drive name **H:** for the path **C:\BUDGET\1988**. By using drive **H:** as the destination drive specification, MS-DOS creates the file in the desired directory.

There are versions of PEP (and there may be versions of STEP) that do not recognize some MS-DOS drive names as valid drives. There are also restrictions that MS-DOS imposes on the operation of "drives" specified with a SUBST command.

Some versions of MS-DOS require adjustment of the LASTDRIVE= specification in the CONFIG.SYS file to accommodate drive names used in a SUBST command.

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 copy book 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 PC, if the filename and extension match an existing file on the diskette, the data in that file is overwritten — this is a consequence of the way PEP/STEP file transfer is designed.

The program does not receive any notification if this occurs. If the file does not exist it is allocated.

The format of the file packet is defined by the supplied copy book TIP/TC-PCFIL. This copy element should be included in the LINKAGE SECTION of the program:

```
    05 FILE-PKT.   COPY TC-PCFIL OF TIP.
*
*    FILE PACKET FOR HOST/PC 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).
```

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.

Note: *See the description of the use of the MS-DOS command "SUBST" at the beginning of this section.*

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

The last four characters contain the "ACK" or "NAK" returned by the PC software.

Note: *End of description of fields in TC-PCFIL copy element.*

INFO-PKT

Information packet required only on the call to TIPH2P or TIPP2H with the FCS-OPEN function. The supplied copy book TIP/TC-PCINF defines the format of the information packet. Place this copy element in a convenient area of the LINKAGE SECTION of the program:

```
           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).
```

PCINF-BUF-LEN

Specifies the length of a buffering area in the user program into which PCXFER: OPEN 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. A further consideration on the buffer size is discussed in "PCXFER Masking" on page 3-145.

PCINF-ERR-TERM

Not used at this time.

PCINF-INDEX

The number of the display screen to be used in the transfer.

If this field is a space, the index is set to 1.

If this field contains "?", the index value on the control page is not changed.

PCINF-OPTIONS

As defined by STEP or PEP software.

Defaults are in effect when this field contains spaces and vary according to the value contained in the field: PCINF-CONTROL-CODE.

This field normally does not require attention as the transfer subroutine sets the appropriate options for file transfers.

If you set your own PEP options this may restrict record lengths to 1910 characters and result in less efficient transfer rates.

PCINF-SEPARATOR

Record separator required when transferring EBCDIC or ASCII files.

The default value when this field is spaces is PIC X(2) VALUE '18' (representing the value X'18' — hexadecimal).

PCINF-END-OF-FILE

This field is only used on transfers from the host to an MS-DOS file.

This is a character string that STEP/PEP uses to identify the end of file on transfers from host to MS-DOS files.

This field is normally set to spaces — this instructs PCXFER to unlock the keyboard on the last transferred screen that STEP/PEP accepts as EOF.

PCINF-PROMPT

This field is only used on transfers from an MS-DOS file to the host.

This field is normally set to space — this allows the default value of "1E" (an SOE character ▸) to be used.

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:

Code	Description
space	Used when transferring purely graphic character data. In this mode of operation, spaces at the end of a line are suppressed.
B	Implies that records are subject to trailing space suppression, the remaining data is "hexified" as in the "H" mode.
H	Used when transferring binary or packed decimal data. The entire record is "hexified" and transferred in core image mode.
N	No trailing space suppression is to take place.
T	Used when transferring records containing EBCDIC and binary data to or from the PC. The data is "hexified" and requires EBCDIC to ASCII, or ASCII to EBCDIC translation. For further information see "PCXFER Masking" on page 3-145.

PCINF-COMMENTS

This is a 60 character field that may contain any desired comment or header information. This field is recognized only on a call to FCS-OPEN.

When this field contains some value other than spaces, a header record is written as the first line of the MS-DOS file during an open call to TIPH2P. The record header consists of the prefix "$*LIBHDR*$" followed by the 60 character comment field.

When the comment field contains spaces no header record is written.

During an open call to TIPP2H the first record is read and checked for the header prefix; if this is a header record the 60 characters following the prefix are placed in the comment field.

The comment field contains spaces if no header record is present.

Only displayable characters are allowed in the comments field.

PCINF-COMPRESS

Setting this field to "Y" causes PCXFER to apply a compression algorithm to the data that is sent to the PC. Use of the compression facility can reduce the

amount of data sent to the PC (and therefore, proportionally reduce the transfer time.

To be able to decompress the data on the PC, the supplied decompression PC program must be available on the PC.

For more information, see "PCXFER Compression" on page 3-148.

Setting this field to a space or 'N' indicates that no data compression is to be attempted.

PCINF-RESERVED

This field is reserved for future use.

Note: *End of description of fields in TC-PCINF copy element.*

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. The 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.

Use the supplied copy book TIP/TC-PCBUF to define the buffer! Place this copy element in any convenient area of 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).
```

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.

Note: *End of description of fields in TC-PCBUF copy element.*

Error Conditions:

PIB-FUNCTION	The program appears to executing on a terminal that is not equipped with a PEP or STEP interface, or the first parameter is invalid (not one of: FCS-OPEN, FCS-PUT, FCS-FLUSH or FCS-CLOSE).
PIB-BREAK	Buffer not initialized, not opened or error on the previous FCS-GET.
PIB-IO-ERROR	Invalid parameter list.
PIB-NO-MEM	Buffer length too small (less than 1024 bytes).
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.

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 unwanted data in the PEP/STEP control page settings.

3.20.2. FCS-GET — Input Record from PC

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 copy book 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. This is the same packet as described in the previous section (TIPH2P/TIPP2H: FCS-OPEN).

RECORD-PKT

This is the record area into which FCS-GET returns the MS-DOS data. Use the supplied copy book TIP/TC-PCREC to define this area! This copy element must appear in an area of the LINKAGE SECTION of the program:

```
       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
*
```

Where:

RECORD-PKT

A variable length record containing a length field, and the data to transferred from the MS-DOS file.

The program must define the appropriate record fields immediately after PCREC-DATA. Multiple record types are handled by redefinition.

PCREC-LENGTH

The length of the record packet.

Note: *The length specified must include the 4 bytes preceding the actual data. See also description of the field PCREC-LENGTH in the description of the FCS-OPEN function call.*

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. After a call to FCS-GET this field is set by the PCXFER interface to the actual record length returned from the MS-DOS file.

PCREC-CONTROL

If set to "M" this field indicates the masking feature is on when translation is in effect. See section "PCXFER Masking" on page 3-145.

PCREC-DATA

This field contains the data of the record to be transferred.

Note: *End of description of fields in the copy element TC-PCREC.*

BUFFER

The buffer that is assigned for use by TIPP2H as described in the previous discussion of TIPH2P/TIPP2H: FCS-OPEN.

Error Conditions:

PIB-EOF	This status is returned when STEP/PEP detects end of file or the MSG WAIT key is pressed.
PIB-NO-MEM	This status is returned if the record length is longer than the buffer provided. This applies only if the default options are not used.

PIB-BREAK

This status is returned if an error has been detected at the PC. If PEP is configured correctly the error can be further analyzed by looking at the contents of PCFIL-ACKNOWLEDGE:

NAK1	Disk full transfer not completed.
NAK2	Disk not ready.
NAK3	Disk is write protected.
NAK4	I/O error on diskette.

3.20.3. FCS-PUT — Output Record to PC

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 copy book 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. This is the same packet as described in the previous section (TIPH2P and TIPP2H: FCS-OPEN).

RECORD-PKT

This is the record packet that contains the data to be transferred.

Use the supplied copy book TIP/TC-PCREC to define this area! This copy element must be placed in any convenient area of the LINKAGE SECTION of the program:

```
       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
*
```

Where:

RECORD-PKT

A variable length area containing a length field, and the data to be transferred to the MS-DOS file.

The program must define the data fields immediately after PCREC-DATA. Multiple record types are handled by redefinition.

PCREC-LENGTH

The length of the entire record packet.

This field should be specified explicitly before each call to FCS-PUT.

Note: *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 this field is set to "M", the masking feature is set on when translation is being done. See section "PCXFER Masking" on page 3-145.

PCREC-DATA

This field contains the data of the record to be transferred.

Note: *End of description of fields in TC-PCREC copy element.*

BUFFER

The buffer that is assigned for use by the TIPH2P subroutine as described in the previous section TIPH2P: FCS-OPEN.

Error Conditions:

PIB-NO-MEM This status is returned if the record length is longer than the buffer provided. Only applies when not using default options.

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 PC. Further analyse the problem by looking at the contents of PCFIL-ACKNOWLEDGE. The following values appear in that field (assuming STEP/PEP has been configured with the default error responses):

NAK1	Disk full transfer not completed.
NAK2	Disk not ready.
NAK3	Disk is write protected.
NAK4	I/O error on diskette or the interface has not been correctly opened.

PIB-EOF This status is returned if the terminal user has forced a premature end of file condition by aborting the file transfer process (by pressing the [MSG WAIT] key for example).

3.20.4. 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 copy book 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. This is the same packet as described in the previous section (TIPH2P: FCS-OPEN).

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 TIPH2P's use as described in the previous section TIPH2P: FCS-OPEN.

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).

3.20.5. 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
```

Where:

FCS-CLOSE

This function code, as defined by the supplied copy book 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. This is the same packet as described in the previous section (TIPH2P and TIPP2H: FCS-OPEN).

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.

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 PC software/hardware is notified that the file transfer operation is complete — this can prevent subsequent file transfer attempts from having problems.

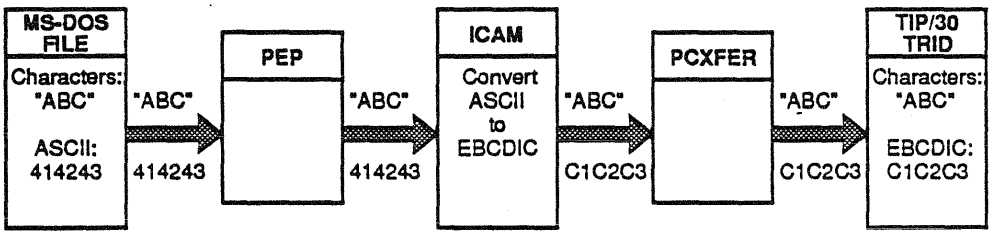
3.20.6. PCXFER Masking

Programs using PCXFER require the masking feature of the transfer subroutines whenever the data to be transferred includes EBCDIC or ASCII data mixed with binary or packed decimal fields.

UNISCOPE protocol cannot handle data of this type because the data may contain bit patterns that are reserved for use by the protocol itself. Both TIPH2P/TIPP2H and STEP/PEP must therefore "hexify" this data.

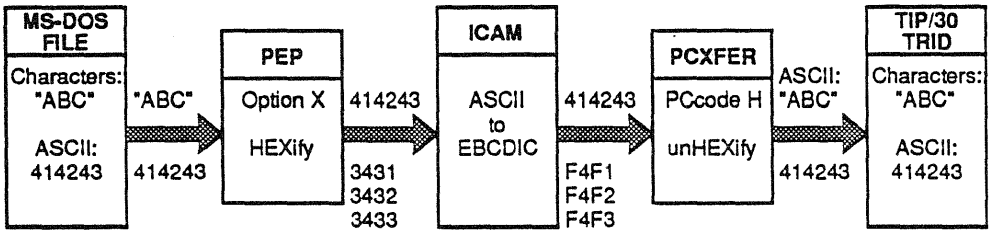
The following diagrams show the effect of different values in the field PCINF-CONTROL-CODE on the data that is transferred:

Example Normal Transfer (space):



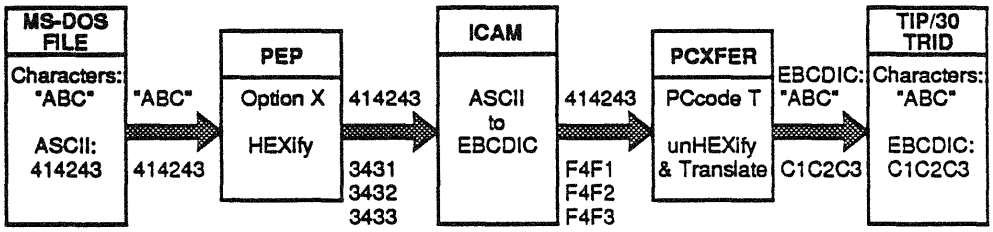
In this example ICAM translates the printable data from ASCII to EBCDIC so that you do not need to be concerned with it. This data must contain only printable data.

Example Hexification; no translation (H):



In this case we have asked PEP and PCXFER to "hexify" the data. It is assumed that some portion of the record contains non-printable data. ICAM still does a translation but on the expanded or "hexified" data. This leaves us (after "unhexification") with ASCII data in our TIP/30 program. This method would be fine if we only intend to return this data to another PC using the HOST as temporary storage place and a connection between remote end users.

Example Hexification and Translation (T):



In this example, we are transferring data with a mix of printable characters and binary or numeric packed data. In this case, however, we would like the data to be in usable form on the HOST. PCXFER translates the data from ASCII to EBCDIC after "unhexification" of the data.

This brings us to another problem. If we allow all the data to be translated by PCXFER including the binary or packed data fields. The resulting data in those fields would be unpredictable. The following paragraphs describe a masking procedure which will allow you to indicate which portions of a record are to be translated and which are to be left untouched.

To mask a character for translation, the corresponding byte in the mask record is set to a space (X'40'). Any other value placed in the mask inhibits the translation of the character in the corresponding data location. The method for setting the mask record is depends on the direction of the data transfer.

3.20.6.1. Transfer from MS-DOS File

When transferring from the MS-DOS file to the host, move spaces to PCREC-DATA then move LOW-VALUES to those fields that do not require translation.

Next set PCREC-CONTROL to a value of "M" to indicate masking is required and then issue the FCS-GET function as described in previous section (TIPP2H: FCS-GET).

The record is returned as it normally would be — except that the appropriate fields are not translated.

The program must place the mask record in PCREC-DATA before each use of the FCS-GET function.

3.20.6.2. Transfer to MS-DOS File

The process is slightly more difficult when the transfer is done from the host to the MS-DOS file since your program must provide the mask to TIPH2P before sending the record data. The mask information is saved in the top of the buffer area, you must provide a buffer area one record size larger than normal to contain the mask information.

When your program issues the FCS-PUT function with the mask in the record area and PCREC-CONTROL set to "M", TIPH2P saves the mask and translates all subsequent records passed to TIPH2P. The program should set PCREC-CONTROL back to a space when record data is sent. It can send the mask again any time the interface is open.

3.20.7. PCXFER Compression

Data compression may be requested when transferring data to a PC by placing a 'Y' in the PCINF-COMPRESS field before opening the transfer interface. This will cause the data to be character compressed.

Any string of 4 or more identical characters will be compressed into a 3 character code. If you have also requested "hexification" of the data (by setting PC-CONTROL-CODE to an "H" or "T"), the data will not be "hexified" but will be compressed and folded. Depending on the data which you are transferring this can substantially reduce the amount of data sent down the communication line.

The data file received at the PC is in compressed format and may not be usable on the PC unless you only intend to copy it to another TIP/30 system. A PC program has been provided that decompresses and, if necessary, unfolds the file that has been copied to the PC.

The PC program is named PC-COMP. It must be copied to the PC with the following command entered at the TIP/30 command line prompt:

```
TIP?▶COPY,X  TIP/TC$COMP,,C:PC-COMP.EXE
```

Of course, the example above assumes that the program is to be copied to the "C" drive on the PC; adjust the command as necessary for your PC[17].

If TC-COMP is executed without command line parameters, the following syntax help information is displayed:

```
C:>TC-COMP

Error => Invalid command syntax.

Usage is :
     TC-COMP <command> <source-file> <target-file>

          where command is :
             -C  - Compress using ASCII compression.
             -D  - Decompress file.
             -H  - Compress using hexification.
```

Additional Considerations:

The "-C" command is used to compress a PC file that is known to contain only displayable ASCII characters. The "-H" command is used to compress and fold a PC file (this is required if the file contains non-displayable characters).

The source-file is required; the target-file defaults to the source-file.

When a file is copied to the host TIP/30 system, the transfer code automatically determines if the file needs to be uncompressed or unfolded. The data is always returned to its natural state.

17. The PC program can be given any name; TC-COMP.EXE is a suggestion.

3.21. TIP/30 DMS Interface

This section provides information regarding the implementation of the various features provided by the Multi-thread DMS interface, primarily as they apply to programs written in a native TIP/30 environment.

3.21.1. DMS Interface: XR3IMS

TIP/30 utilizes the same interface to DMS as is used by the IMS product. There are several considerations for using DMS with TIP/30 that are discussed in detail in following subsections. To fully comprehend these discussions you must have a working knowledge of the various DMS components and their acronyms, such as DMCL and DUPL. You can obtain this information by reviewing the various DMS manuals supplied by the manufacturer.

3.21.2. DMCL Considerations

Since TIP/30 uses the IMS/DMS interface, you must generate your database DMCLs that are to be accessed via online programs with the ONLINE QUICK-BEFORE-LOOKS statement.[18] Although various DMS options allow you to execute online programs that update the database without using QUICK-BEFORE-LOOKS, this is highly undesirable. Suppression of QBL facilities implies that should a program terminate abnormally without having fully completed a series of related updates, the database may be left in a physically, as well as logically, compromised state.

3.21.3. DBMS Start up

You must supply statements within the DUPL commands used for DBMS start up. The statements that are of primary concern are the MAXIMUM ONLINE-THREADS and ONLINE-TERMINALS statements.

MAXIMUM ONLINE-TERMINALS must always specify a value that is at least equal to the number of terminals that will access DMS. Take care to count all logical terminals, not just physical ones. Also be sure to include DMS transactions that are either user initiated, or program initiated (for example, via a call to the TIPFORK subroutine) at screen bypass terminals.

Set MAXIMUM ONLINE-THREADS according to the DBMS option that is selected in the TIPGEN parameters. Set the ONLINE-THREADS statement to be equal to the number of threads indicated by the TIPGEN DBMS=DMS specification. Both single and multi-thread IMS emulated programs will function under the TIP/30 — DMS interface.

18. Before Release 12 of OS/3 the word "ONLINE" was coded as "IMS". "ONLINE" now is used as a more generic term.

3.21.4. COBOL Compile PreProcessing

When preprocessing TIP/30 COBOL programs for the purpose of compiling them, code the DMLP DUPL as though an IMS program is being compiled, for example: PRE-PROCESS IMS PROGRAM....

Note: *As of Release 12.0 of OS/3, the preprocessor allows use of the statement: PRE-PROCESS TIP PROGRAM. This causes the preprocessor to generate calls to the module "TIPDMS" (rather than "XR3IMS"). This minor change eliminates the possible accidental inclusion of the IMS interface module in a TIP/30 program.*

A TIP/30 copy module is supplied that can be used for the DMS-STATUS section. This module, TIP/TC-DMSST, is intended to be the equivalent of the IMS module IMSTAT74.

The copy element TC-DMSST expects that the MCS area includes a COPY statement for the element TC-DMSER from the TIP library. The TC-DMSER copy element contains message data items used by TC-DMSST to construct error messages which are displayed under certain error conditions.

The copy element TIP/TC-DMSST contains:

```
/
 DMS-STATUS SECTION.
*    ************************************************
*    *   DMS-STATUS SECTION FOR TIP/30 PROGRAMS      *
*    *   (SHOULD NOT BE USED IN TIP SUBPROGRAMS. USE *
*    *    TC-DMSSP INSTEAD, TO AVOID RECURSIVE       *
*    *    SUBPROGRAM CALLS!)                         *
*    ************************************************
*
*    TC-DMSST     VERSION 3.2      11/10/88
*
 STATUS-PARA.
*
     IF (ERROR-STATUS NOT = ZEROES) AND
        (ERROR-STATUS NOT = '1601')
        PERFORM DMS-ABORT
        CALL 'TIPDUMP'.
*
*    END OF DMS-STATUS SECTION.
*
 DMS-ABORT SECTION.
*
** REPORT A DMS ERROR TO USER
*
 ABORT-PARA.
     MOVE SPACES              TO  TF-DMSER-DATA.
     MOVE ERROR-STATUS        TO  ERROR-STATUS-DMSER.
     MOVE RB-ERROR-CODE       TO  RB-ERROR-CODE-DMSER.
     MOVE ERROR-RECORD        TO  ERROR-RECORD-DMSER.
     MOVE ERROR-SET           TO  ERROR-SET-DMSER.
```

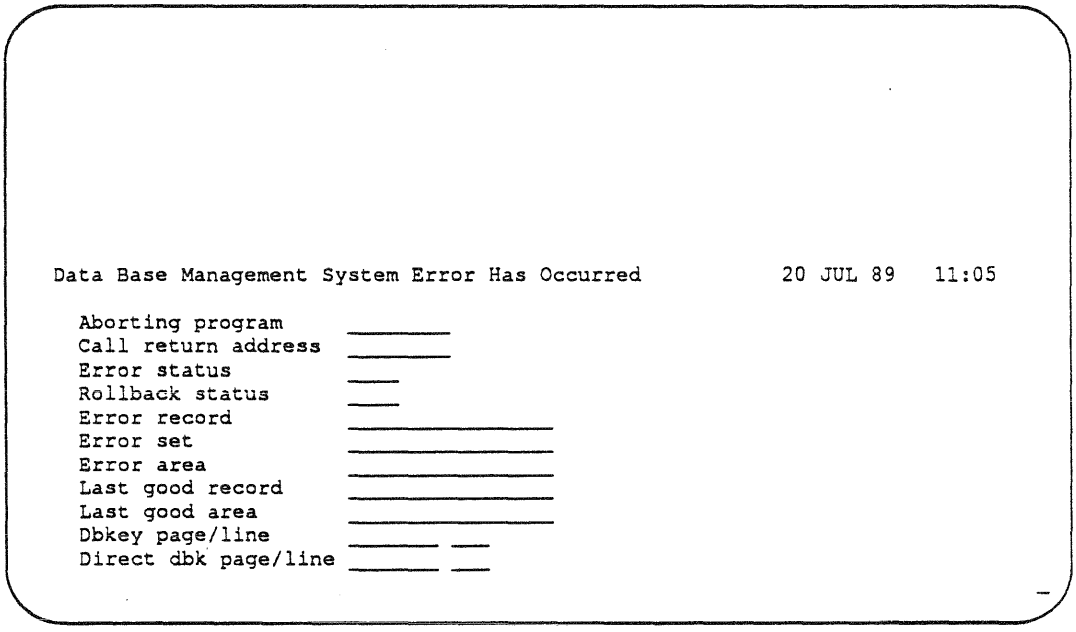
```
           MOVE ERROR-AREA          TO  ERROR-AREA-DMSER.
           MOVE RECORD-NAME         TO  RECORD-NAME-DMSER.
           MOVE AREA-NAME           TO  AREA-NAME-DMSER.
           DIVIDE DBKEY BY 256      GIVING DBKEY-DMSER.
           COMPUTE DBKEY-LIN-DMSER = DBKEY - (DBKEY-DMSER * 256).
           DIVIDE DIRECT-DBK BY 256 GIVING DIRECT-DBK-DMSER.
           COMPUTE DIRECT-DBK-LIN-DMSER = DIRECT-DBK -
                                          (DIRECT-DBK-DMSER * 256).
           MOVE CALL-RETURN-ADDR    TO  CALL-RETURN-DMSER.
           MOVE 'TF$DMSER'          TO  MCS-NAME.
           MOVE 116                 TO  MCS-COUNT.
           MOVE SPACE               TO  MCS-FUNCTION.
           IF RB-ERROR-CAUSE-DMSER NOT = ZEROS
              MOVE 'TT-DMS'         TO PIB-TRID
              CALL 'TIPSUBP' USING MCS
              IF ERROR-TEXT-DMSER NOT = SPACES
                 CALL 'TIPMSGEO' USING ERROR-TEXT-DMSER.
           CALL 'TIPMSGO' USING MCS.
       DMS-ROLLBACK SECTION.
      *
      *    ****************************************************
      *    *   DMS-ROLLBACK SECTION FOR TIP/30 PROGRAMS       *
      *    ****************************************************
      *
       DMS-ROLLBACK-PARA.
      *    ****************************************************
      *    *   USER PROGRAM SHOULD DEFINE DMS-ROLLBACK-LOCK   *
      *    *   PARAGRAPH TO HANDLE WAIT LOCK ERRORS IN-LINE.  *
      *    *   YOU MAY CODE "GO TO DMS-ROLLBACK-CONTINUE." IF *
      *    *   YOU WANT WAIT LOCK ERRORS TO SIMPLY ABORT.     *
      *    ****************************************************
           IF (RB-ERR-CAUSE = '15' OR '18' OR '19') OR
      *    ****************************************************
      *    *   CONTROL PASSES TO: DMS-ROLLBACK-LOCK           *
      *    *    WHEN STANDARD RECORD/PAGE LOCK ABORTS OCCUR,  *
      *    *    -OR- IF THE USER HAS SET THE FREE-CKPT-CODE   *
      *    *    TO A VALUE OF 'BACK' (THIS COULD BE DONE ONCE *
      *    *    AT THE TIME OF IMPART). THIS LATER TECHNIQUE  *
      *    *    CAN BE USED WHEN THE USER PROGRAM DESIRES TO  *
      *    *    HANDLE -ALL- ROLLBACK ERRORS.  THIS WOULD BE  *
      *    *    REQUIRED, FOR INSTANCE, IF THE PROGRAM WAS    *
      *    *    RUNNING IN BACKGROUND, WHERE THE STANDARD     *
      *    *    'ERROR SCREEN' TERMINATION WOULD NOT BE       *
      *    *    APPROPRIATE.                                  *
      *    ****************************************************
              (FREE-CKPT-CODE = 'BACK')
              GO TO DMS-ROLLBACK-LOCK.

       DMS-ROLLBACK-CONTINUE.
           PERFORM DMS-ABORT.
```

```
     MOVE 300                  TO PIB-WAIT-TIME.
     IF ERROR-TEXTX-DMSER = SPACE
*       ALLOW TERMINAL USER TO READ ERROR MESSAGE
        CALL 'TEXT' USING ERROR-TEXT3-DMSER
*        THEN DIE WHEN ERROR IS NOT TRANSLATABLE
        CALL 'TIPDUMP'
     ELSE
*       ALLOW TERMINAL USER TO READ ERROR MESSAGE
        CALL 'TEXT' USING ERROR-TEXT3-DMSER
*        THEN DISPLAY IT AND RETURN NORMALLY
        CALL 'TIPRTN'.
*
*    END OF DMS-ROLLBACK SECTION.
```

Copy element TIP/TC-DMSER contains:

```
*
** MCS-DATA LAYOUT FOR 'TF$DMSER', TO REPORT DMS ERRORS
*
     03  TF-DMSER-DATA.
         05  CALL-RETURN-DMSER    PICTURE X(08).
         05  ERROR-STATUS-DMSER   PICTURE X(04).
         05  RB-ERROR-CODE-DMSER.
             10  FILLER               PICTURE XX.
             10  RB-ERROR-CAUSE-DMSER PICTURE XX.
         05  ERROR-RECORD-DMSER   PICTURE X(16).
         05  ERROR-SET-DMSER      PICTURE X(16).
         05  ERROR-AREA-DMSER     PICTURE X(16).
         05  RECORD-NAME-DMSER    PICTURE X(16).
         05  AREA-NAME-DMSER      PICTURE X(16).
         05  DBKEY-DMSER          PICTURE 9(07).
         05  DBKEY-LIN-DMSER      PICTURE S9(03).
         05  DIRECT-DBK-DMSER     PICTURE 9(07).
         05  DIRECT-DBK-LIN-DMSER PICTURE S9(03).
         05  FILLER               PICTURE X(4).
         05  ERROR-TEXT-DMSER.
             10  ERROR-TEXT1-DMSER.
                 15  ERROR-TEXTX-DMSER PICTURE X.
                 15  FILLER            PICTURE X(79).
             10  ERROR-TEXT2-DMSER     PICTURE X(80).
             10  ERROR-TEXT3-DMSER     PICTURE X(80).
```

The status coding is designed to output a screen format (assumed name: TF$DMS0_) explaining the error situation. The following screen format is supplied with the TIP/30 system:



The code in the TIP/30 supplied DMS-STATUS module (TC-DMSST) calls TIPSUBP to call a resident subprogram named TT-DMS[19] This call is issued for the purpose of translating certain common DMS error return codes into messages which can be understood by application terminal operators.

19. The source coding for this module is included in the TIP library for installations that need to alter it to accommodate local language.

This method has been chosen to minimize the need for memory at run time. However, this technique does not work if the routines which use the TIP/30 supplied status module are themselves used in a subprogram since a subprogram cannot call another subprogram. Subprograms which use the DMS interface must use the TIP/30 supplied status module named "TC-DMSSP" (from the TIP library):

```
/
 DMS-STATUS SECTION.
*    ********************************************************
*    *   DMS-STATUS SECTION FOR TIP/30 RESIDENT SUBPROGRAMS *
*    *   (CAN BE USED IN REGULAR PROGRAMS, BUT TC-DMSST     *
*    *    IS SMALLER AND MORE EFFICIENT!)                   *
*    ********************************************************
*
*    TC-DMSSP      VERSION 3.2       11/10/88
*
 STATUS-PARA.
*
     IF (ERROR-STATUS NOT = ZEROES) AND
        (ERROR-STATUS NOT = '1601')
        PERFORM DMS-ABORT
        CALL 'TIPDUMP'.
*
*    END OF DMS-STATUS SECTION.
*
 DMS-ABORT SECTION.
*
** REPORT A DMS ERROR TO USER
*
 ABORT-PARA.
     MOVE SPACES              TO  TF-DMSER-DATA.
     MOVE ERROR-STATUS        TO  ERROR-STATUS-DMSER.
     MOVE RB-ERROR-CODE       TO  RB-ERROR-CODE-DMSER.
     MOVE ERROR-RECORD        TO  ERROR-RECORD-DMSER.
     MOVE ERROR-SET           TO  ERROR-SET-DMSER.
     MOVE ERROR-AREA          TO  ERROR-AREA-DMSER.
     MOVE RECORD-NAME         TO  RECORD-NAME-DMSER.
     MOVE AREA-NAME           TO  AREA-NAME-DMSER.
     DIVIDE DBKEY BY 256      GIVING DBKEY-DMSER.
     COMPUTE DBKEY-LIN-DMSER = DBKEY - (DBKEY-DMSER * 256).
     DIVIDE DIRECT-DBK BY 256 GIVING DIRECT-DBK-DMSER.
     COMPUTE DIRECT-DBK-LIN-DMSER = DIRECT-DBK -
                                    (DIRECT-DBK-DMSER * 256).
     MOVE CALL-RETURN-ADDR    TO  CALL-RETURN-DMSER.
     MOVE 'TF$DMSER'          TO  MCS-NAME.
     MOVE 116                 TO  MCS-COUNT.
     MOVE SPACE               TO  MCS-FUNCTION.
     MOVE SPACES              TO ERROR-TEXT-DMSER.


*    ANALYZE THE ROLLBACK CAUSE ALREADY SET IN THE MCS AREA!
```

```
      IF RB-ERROR-CAUSE-DMSER = '10'
      *    INSUFFICIENT MAIN STORAGE FOR LOCKS
           MOVE
                'Notify DATA PROCESSING that the LOCK POOL is too sma
      -         'll'                TO ERROR-TEXT1-DMSER
           MOVE 'the transaction may be able to work if you try it ag
      -         'ain!'              TO ERROR-TEXT2-DMSER

      ELSE IF RB-ERROR-CAUSE-DMSER = '15' OR '18' OR '19'
      *    RECORD/AREA WAIT TIME EXCEEDED (RO CONFLICT)
           MOVE 'The records needed by this transaction were in use e
      -         'lsewhere in the system,'
                                    TO ERROR-TEXT1-DMSER
           MOVE  'you should rekey the transaction code and try again
      -          '!'                TO ERROR-TEXT2-DMSER

      ELSE IF RB-ERROR-CAUSE-DMSER = '30'
      *    REQUESTED DMCL NOT LOADED
           MOVE 'The database needed by this program is not currently
      -         ' available,'       TO ERROR-TEXT1-DMSER
           MOVE 'contact DATA PROCESSING to determine when it will be
      -         ' made available!'  TO ERROR-TEXT2-DMSER

      ELSE IF RB-ERROR-CAUSE-DMSER = '32'
      *    INSUFFICIENT MAIN STORAGE IN SUBSCHEMA POOL
           MOVE 'Notify DATA PROCESSING that the SUBSCHEMA POOL is to
      -         'o small,'          TO ERROR-TEXT1-DMSER
           MOVE 'the transaction may be able to work if you try it ag
      -         'ain!'              TO ERROR-TEXT2-DMSER

      ELSE IF RB-ERROR-CAUSE-DMSER = '39'
      *    SUBSCHEMA DATE/TIME MISMATCH
           MOVE 'Notify DATA PROCESSING that this transcation has a D
      -         'ATE/TIME mismatch,'
                                    TO ERROR-TEXT1-DMSER
           MOVE 'they will need to recompile this program BEFORE it c
      -         'an be used again!' TO ERROR-TEXT2-DMSER

      ELSE IF RB-ERROR-CAUSE-DMSER = '40'
      *    ONLINE TERMINAL LIMIT EXCEEDED
           MOVE 'Notify DATA PROCESSING that the DMS TERMINAL LIMIT w
      -         'as exceeded,'      TO ERROR-TEXT1-DMSER
           MOVE 'the transaction may be able to work if you try it ag
      -         'ain!'              TO ERROR-TEXT2-DMSER

      ELSE IF RB-ERROR-CAUSE-DMSER = '43' OR '45'
      *    DATABASE OR AREA SHUTDOWN
           MOVE 'The database has been made unavailable due to an err
      -         'or,'               TO ERROR-TEXT1-DMSER
           MOVE 'contact DATA PROCESSING to determine when it will be
      -         ' made available!'  TO ERROR-TEXT2-DMSER
```

```
     ELSE IF RB-ERROR-CAUSE-DMSER = '51' OR '52' OR '53'
*       NEW IMPART INHIBITED FOR DBMS, DBA, OR DMCL
        MOVE 'The database needed by this program is not currently
-            ' opened,'           TO ERROR-TEXT1-DMSER
        MOVE 'contact DATA PROCESSING to determine when it will be
-            ' re-opened!'        TO ERROR-TEXT2-DMSER

     ELSE IF RB-ERROR-CAUSE-DMSER = '58' OR '59'
*       ONLINE ACCESS INHIBITED FOR DMCL OR DBMS
        MOVE 'The database needed by this program is currently res
-            'tricted from online access,'
                                  TO ERROR-TEXT1-DMSER
        MOVE 'contact DATA PROCESSING to determine when it will be
-            ' made available!'   TO ERROR-TEXT2-DMSER

     ELSE IF RB-ERROR-CAUSE-DMSER = '97'
*       SVC PROCESSING ERROR
        MOVE 'The computer job that controls the database does not
-            ' appear to be running,'
                                  TO ERROR-TEXT1-DMSER
        MOVE 'contact DATA PROCESSING to determine when it will be
-            ' made available!'   TO ERROR-TEXT2-DMSER.
     IF ERROR-TEXT-DMSER NOT = SPACES
        CALL 'TIPMSGEO' USING ERROR-TEXT-DMSER.
     CALL 'TIPMSGO' USING MCS.
 DMS-ROLLBACK SECTION.
*
*    ************************************************
*    *   DMS-ROLLBACK SECTION FOR TIP/30 PROGRAMS   *
*    ************************************************
*
 DMS-ROLLBACK-PARA.
*    ************************************************
*    *   USER PROGRAM SHOULD DEFINE DMS-ROLLBACK-LOCK   *
*    *   PARAGRAPH TO HANDLE WAIT LOCK ERRORS IN-LINE.  *
*    *   YOU MAY CODE "GO TO DMS-ROLLBACK-CONTINUE." IF *
*    *   YOU WANT WAIT LOCK ERRORS TO SIMPLY ABORT.     *
*    ************************************************
     IF (RB-ERR-CAUSE = '15' OR '18' OR '19') OR
*    ************************************************
*    *   CONTROL PASSES TO: DMS-ROLLBACK-LOCK       *
*    *    WHEN STANDARD RECORD/PAGE LOCK ABORTS OCCUR,  *
*    *    -OR- IF THE USER HAS SET THE FREE-CKPT-CODE   *
*    *    TO A VALUE OF 'BACK' (THIS COULD BE DONE ONCE *
*    *    AT THE TIME OF IMPART). THIS LATER TECHNIQUE  *
*    *    CAN BE USED WHEN THE USER PROGRAM DESIRES TO  *
*    *    HANDLE -ALL- ROLLBACK ERRORS.  THIS WOULD BE  *
*    *    REQUIRED, FOR INSTANCE, IF THE PROGRAM WAS    *
*    *    RUNNING IN BACKGROUND, WHERE THE STANDARD     *
```

```
*    *    'ERROR SCREEN' TERMINATION WOULD NOT BE        *
*    *    APPROPRIATE.                                   *
*    ******************************************************
        (FREE-CKPT-CODE = 'BACK')
        GO TO DMS-ROLLBACK-LOCK.

 DMS-ROLLBACK-CONTINUE.
     PERFORM DMS-ABORT.
     MOVE 300                  TO PIB-WAIT-TIME.
     IF ERROR-TEXTX-DMSER = SPACE
*       ALLOW TERMINAL USER TO READ ERROR MESSAGE
        CALL 'TEXT' USING ERROR-TEXT3-DMSER
*        THEN DIE WHEN ERROR IS NOT TRANSLATABLE
        CALL 'TIPDUMP'
     ELSE
*       ALLOW TERMINAL USER TO READ ERROR MESSAGE
        CALL 'TEXT' USING ERROR-TEXT3-DMSER
*        THEN DISPLAY IT AND RETURN NORMALLY
        CALL 'TIPRTN'.
*
*    END OF DMS-ROLLBACK SECTION.
```

This module is functionally equivalent to TC-DMSST, but will not result in a recursive subprogram call.

3.21.5. TIP/30—DMS Programming

TIP/30 native programs utilize the multi-thread online version of DMS. In addition to allowing multiple online programs to be simultaneously IMPARTed to DMS, resulting in concurrent database access, this interface affords significant capability to maintain a DMS "Success Unit"[20] beyond the scope of a single dialog with a terminal user.

Specifically, programs may hold various DMS record locks and roll back capabilities across screen outputs and inputs and across program transfer of control. You should use the DML UNBIND and BIND verbs in this environment to properly coordinate the use of these facilities.

A program that has issued an IMPART to DMS may not transfer control to another program or solicit input without issuing either a DEPART or an UNBIND statement. Neither of these DML verbs is executed immediately by TIP/30. Instead, TIP/30 internally notes their use and the verbs are executed only when their use can be properly coordinated with processing that might be occurring against conventional OS/3 files, such as MIRAM files. This is done so that should a user or system initiated "roll back" be requested, the database and the conventional files can be kept in proper synchronization.

20. See the DMS DML manual and the IMS/DMS manual for discussions on success units.

All user intentions for special processing with regard to the holding of various locks or establishing new roll back points are indicated by use of the PIB-LOCK-INDICATOR field. If special processing is required, set this field just prior to issuing any of the following TIP/30 calls:

- TIPDXC
- TIPJUMP
- TIPMSGI
- TIPMSGRV
- TIPRTN
- TIPSUB
- TIPSUBP
- TIPTIMER
- TIPXCTL
- PROMPT
- PROMPTX
- PROMPTX8
- TEXT
- TEXT80
- TIPTERM (T-GET)

These calls divide into the same two groups previously described:

1. Requesting Input and
2. Transferring program control.

When you use the PIB-LOCK-INDICATOR, it describes what action should be taken for both DMS and conventional files. This discussion however, is limited to the effect on the DMS environment.

The following table shows each of the combinations that exist with regard to issuing either DEPART/UNBIND and setting the PIB-LOCK-INDICATOR.

DML Verb	PIB-LOCK-INDICATOR	Currency Locks	Keep Locks	Update Locks	Area Status
DEPART	space	Release	Release	Release	Closed
UNBIND *	space	Release	Release	Release	Open
DEPART	H or R	Release	Release	Held	Closed
UNBIND *	R	Release	Release	Held	Open
UNBIND *	H	Release	Held	Held	Open
UNBIND *	O	Release	Release	Roll back	Open
DEPART	O	Release	Release	Roll back	Closed

Legend:

*	Assumes that the subsequent BIND accesses the same subschema.
R	Causes the "success unit" to be extended by holding update locks, but RELEASES all other record locks.
H	Does all that "R" does, but HOLDS all DMS KEEP locks as well.
O	Causes all updates and locks created within the current "success-unit" to be "rolled back". This is the logical equivalent to a DML ROLLBACK or DEPART WITH ROLLBACK, when used with the UNBIND or DEPART, respectively.

As noted, the various UNBIND status codes apply when the UNBIND/BIND sequence is used with a single subschema. When this is not the case, the UNBIND will resemble a DEPART with an H/R/O lock indicator. Note also that these same conditions exist when a program or subprogram UNBIND/BIND is issued, such as might occur with a TIPSUBP.

When a BIND is issued after a previous UNBIND, use the DML OPEN AREA verb only in those instances indicated above, that show that the areas will have been closed, such as when a different subschema is being requested.

Much flexibility exists because a given program may UNBIND from DMS and then jump either laterally to another program (TIPXCTL), or go up or down the stack (TIPSUB/TIPRTN), and during each of these operations it may continue to impose various locks upon records in DMS (and FCS).

Consider the following points when doing this type of programming:

- If DMS locks are to be continually imposed throughout a series of program transfers, the PIB-LOCK-INDICATOR must be explicitly set at each transfer point to continue the imposition of the DMS locks.

  This is especially important if one of the intermediate programs does not access DMS. That program must still set the indicator appropriately, or all locks will be released, and a new "success unit" created.

- Some program, in a program transfer chain, must eventually issue a DEPART with a blank PIB-LOCK-INDICATOR, to indicate a full release of DMS resources. If a program ever attempts to return control to the bottom of the TIP/30 stack, or in other words, a program attempts to call TIPRTN with no programs "beneath" it, and internal TIP/30 information indicates that a proper DEPART has not been done, that program is terminated and error message TI094 is issued.

- Because a DMS program may not always be initiated from the bottom of the TIP/30 "stack", the problem described in the previous bullet item might not always be detectable. The potential exists, due to the PCS TIPSUB/TIPRTN calls, for a program that has been called by TIPSUB to IMPART and erroneously fail to DEPART, before TIPRTN to the originating program.

  To eliminate this potential problem, TIP/30 enforces a restriction on DMS programs that remain "connected" to DMS across program boundaries. Specifically, no program may TIPRTN below the stack level at which it first issued an IMPART, without having issued a DEPART with a space in PIB-LOCK-INDICATOR.

In other words, you may IMPART at a low stack level, and remain connected to DMS up through higher stack levels (TIPSUBs), but you may not initially IMPART at a high stack level and remain connected to DMS down through lower stack levels (TIPRTN).

- It is important to realize that the use of the PIB-LOCK-INDICATOR within a DMS environment, while soliciting an input message, begins to increase the chance that some other DMS program in the system will be "waited" and possibly rolled back by DMS because it could not gain access to a record being held by the first program.

  All aspects of these locks are controlled totally by DMS itself, and should be well understood, as discussed in the DMS Data Manipulation Language manual.

- DMS controls its online user environment on the basis of "terminal-id". If a user program only issues an UNBIND, or attempts to hold locks while soliciting an input message (such as TIPMSGI), the terminal on which it is executing is technically still attached to DMS. If the end user then attempted to use the TIP/30 escape feature[21] of TIP/30 to execute an entirely different program (possibly another DMS program) total confusion may result, because DMS would interpret the calls from these programs as one continuous "event".

  To prevent this from happening, use specify ESCAPE=NO in the TIP/30 Catalogue definition for programs of this type. This effectively turns off the ESCAPE feature while executing that program.

3.21.6. FCS Calls Related to DMS

As has been discussed above, DMS access and conventional file access are closely monitored by TIP/30 to ensure that "checkpoints" and "roll backs" are always coordinated. Consequently, one of the available FCS calls that is provided by TIP/30 has meaning to DMS users as well.

Use of the FCS-TREN function causes the establishment of a new "commit" point. This causes a TIP/30 internal call to DMS to provide for equivalent processing. The FREE WITH CHECKPOINT DML verb is actually issued internally by TIP/30.

Use of the FCS-TREN function, with a PIB-LOCK-INDICATOR setting of "O", causes conventional files to be rolled back to the previous "commit" point. This will also cause DMS updates to be rolled back as well. This is the logical equivalent to the DML verb, ROLLBACK.

Use the FCS-TREN function if you want to initiate either a ROLLBACK or FREE WITH CHECKPOINT.

21. See the documentation of the ESCAPE= TIP/30 generation keyword.

3.21.7. Catalogued DMCL Names

DMS users should be aware of an optional facility that can be invoked for any DMS programs executing under TIP/30. The method that DMS utilizes to properly select the database a user desires when he issues an IMPART, is to examine the DMCL name in the DMCA of the user program. This, and only this, determines which database will be used.

If you want to have both a production and test database for the exact same schema, each with its own DMCL to control it, the normal DMS procedures require compiling the program with a DMS INVOKE statement referencing the DMCL of the test database, for the purpose of program testing. Once tested, you must change the INVOKE statement to reference the production database DMCL, and recompile the program.

TIP/30 provides facilities that make this unnecessary. The TIP/30 catalogue FILE statement provides a facility by which DMCL names can be defined to relate "logical" DMCL names to "physical" ones, on a group by group basis, as is done with conventional files.

When a DMS IMPART is issued from a program running under TIP/30, the catalogue is examined to determine if a FILE DMCL= entry exists for the requested DMCL, using standard order of search processing. If no record is found, the program is simply allowed to IMPART to the requested DMCL. However, if a FILE DMCL= entry exists, the DMCL name that was compiled in the program will be overridden when the IMPART is issued to DMS.

Thus, it is possible to have production and test databases for one schema, or even multiple production databases for that schema, all being accessed by one compiled version of a program. See the FILE statement as described for the CAT utility transaction program for proper syntax.

Refer to the description of the TIP/30 run-time parameter DMSCAT= for additional information.

3.22. TIP/30 Journal File

The TIP/30 File Control System (FCS) automatically writes BEFORE and/or AFTER images of updated records to the TIP/30 Journal file (TIP$JRN) or the TIP/30 Log File (TIP$LOG). Parameters specified for each file in the TIP/30 generation parameters control the writing of before and after images.

FCS also allows a user program to write records to the journal file. Such records can be written to the journal file, for example, to mark certain exceptional events or to be able to monitor transaction usage.

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 a following section).

Syntax:

```
CALL  'TIPFCS'  USING  FCS-JOURNAL
                       file-pkt
                       JRN-RECORD
```

Where:

FCS-JOURNAL

Function code from the TC-FCS copy element.

file-pkt

Logical filename packet.

This parameter is not used. It is required to maintain symmetry with other TIPFCS calls.

JRN-RECORD

The record to be written to the TIP$JRN or TIP$LOG 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.

For more information see the description of the journal file record layouts in the following section.

3.22.1. Journal File Record Format

The TIP/30 Journal file (TIP$JRN) or the TIP/30 Log File (TIP$LOG) contains variable length records. Each record has a journal prefix which 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 supplied copy book TIP/TC-JRN describes the layout of the various records that may appear in the journal file.

```
* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
*                                                                       *
*            TIP/30 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/30 USER LOGOFF
*             LGON  - TIP/30 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/30 STATISTICS RECORD
*             TREN  - END OF TRANSACTION MARKER
*             USER  - USER WRITTEN JOURNAL RECORD
*                     - FORMAT DEFINED BY PROGRAM WHICH WRITES
*                       THE RECORD TO THE JOURNAL FILE
*
*         JRN-UID              - RECORD WRITTEN FOR THIS USERID
*         JRN-TRID             - EXECUTING THIS TRANSACTION PROGRAM
*         JRN-LFD              - FILE LFD NAME
*         JRN-DATE             - DATE STAMP (YYMMDD)
*         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-ROLLBACK         - SET TO "RLBK" IF THIS JOURNAL
```

```
*                          RECORD WAS WRITTEN AS A RESULT OF
*                          ONLINE ROLLBACK (TRANSACTION
*                          ABORTED OR FCS-BACK OR REQUESTED
*                          ROLLBACK VIA PIB-LOCK-INDICATOR)
/
           15  JRN-REC-LEN          PICTURE 9(4) COMP-4 SYNC.
           15  FILLER               PICTURE X(2).
           15  JRN-REC-TYPE         PICTURE X(4).
               88  JRN-AFTR         VALUE   'AFTR'.
               88  JRN-BEFR         VALUE   'BEFR'.
               88  JRN-CKPT         VALUE   'CKPT'.
               88  JRN-DELT         VALUE   'DELT'.
               88  JRN-LGOF         VALUE   'LGOF'.
               88  JRN-LGON         VALUE   'LGON'.
               88  JRN-NEW          VALUE   'NEW '.
               88  JRN-PREN         VALUE   'PREN'.
               88  JRN-PRST         VALUE   'PRST'.
               88  JRN-STAT         VALUE   'STAT'.
               88  JRN-TREN         VALUE   'TREN'.
               88  JRN-USER         VALUE   'USER'.
           15  JRN-UID              PICTURE X(8).
           15  JRN-TRID             PICTURE X(8).
           15  JRN-LFD              PICTURE X(8).
           15  JRN-LFN REDEFINES
                       JRN-LFD      PICTURE X(8).
           15  JRN-DATE             PICTURE 9(6) COMP-3.
           15  JRN-TIME             PICTURE 9(6) COMP-3.
           15  JRN-TID              PICTURE X(4).
           15  JRN-DIRECT-BLK-NO    PICTURE 9(8) COMP-4.
           15  JRN-ACCT             PICTURE X(4).
           15  JRN-ROLLBACK         PICTURE X(4).
               88  JRN-RLBK         VALUE   'RLBK'.

       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-DATE   DATE OF LOGON (YYMMDD)
*         JRN-LGOF-LGON-TIME   TIME OF LOGON (HHMMSS)
*         JRN-LGOF-DATE        DATE OF LOGOFF (YYMMDD)
*         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) COMP-3.
              15  JRN-LGOF-LGON-MM     PICTURE 9(2) COMP-3.
              15  JRN-LGOF-LGON-SS     PICTURE 9(2) COMP-3.
              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(4) COMP-4.
              15  JRN-LGOF-MSGOUT      PICTURE 9(4) COMP-4.
              15  JRN-LGOF-LGON-DATE   PICTURE 9(6) COMP-3.
              15  JRN-LGOF-LGON-TIME   PICTURE 9(6) COMP-3.
              15  JRN-LGOF-DATE        PICTURE 9(6) COMP-3.
```

```
            15  JRN-LGOF-TIME        PICTURE 9(6) COMP-3.
            15  JRN-LGOF-AVG-RESP    PICTURE 9(8) COMP-4.
            15  FILLER               PICTURE X(4052).
            15  FILLER               PICTURE X(4092).
*
*       LOGON RECORD:
*
*         JRN-LGON-DATE        DATE OF LOGON (YYMMDD)
*         JRN-LGON-TIME        TIME OF LOGON (HHMMSS)
*
         10  JRN-LGON-REC  REDEFINES  JRN-DATA.
            15  FILLER               PICTURE X(20).
            15  JRN-LGON-DATE        PICTURE 9(6) COMP-3.
            15  JRN-LGON-TIME        PICTURE 9(6) COMP-3.
            15  FILLER               PICTURE X(4064).
            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)
*                             PREN
*                             PRST
*                             TREN
*
*          CONTAIN ---NO--- DATA OTHER THAN THE PREFIX.
*
* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
```

JRN-PREFIX

A fixed length prefix that appears on the front of ALL records in the journal file.

JRN-REC-LEN

Binary halfword 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/30 user.

JRN-TRID

The TIP/30 transaction name that was executing when this record was written.

JRN-LFD

The applicable file LFD name (applies to file related journal records).

JRN-DATE

The date stamp of this record in YYMMDD 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-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/30.

JRN-ROLLBACK

This field contains the character string "RLBK" if this journal record was written by TIP/30 online roll back. This occurs, for example, when an AFTR image is written as a result of an update being rolled back after a transaction aborted.

JRN-DATA

A group item indicating the start of the "data" portion of the journal record.

Note: *The copy element reserves a great deal of space to accommodate a fairly large record — the record length of the journaled record can be large.*

3.22.2. Batch Journal File Access

The TIP/30 library contains an object module (element name = TI$JRN) which must be linked with user batch programs that read the TIPJRN, TIPB4 or TIP$LOG file.

This object module contains entry points to the subroutines that provide I/O services for the batch program.

You may write a batch program to use these supplied subroutines to read the TIP/30 journal file (or TIPHST, TIPB4 or TIP$LOG file).

Note: *You must place your batch program load module in the TIP$LOD library and execute the batch program from that library. The access subroutines supplied with TIP/30 use other supplied routines to perform the actual file I/O. These subordinate routines are not linked in the load module but are fetched at execution time from the job step library.*

The supplied job stream TJ$LST, in the TIP/30 job control library, sorts and lists the TIP$JRN file (or TIP$B4 or TIP$LOG). Refer to the description of the TJ$LST job stream in *TIP/30 Generation, Maintenance and Installation.*

3.22.2.1. TIPJRNOP — Batch Journal File Open

This subroutine OPENs the input "journal" file. The subroutine OPENs the first LFD that it finds from the following list:

1. TIP$JRN
2. TIP$B4
3. TIP$LOG.

The JCL for your program should specify one of the LFD names from the preceding list (whichever file is to be read). If the program wishes to read a TIP/30 History file, simply define the history file with an LFD of TIP$JRN.

Syntax:

```
CALL  'TIPJRNOP'
```

Where:

No parameters required.

Additional Considerations:

TIP/30 does not provide an error status; if the input file fails to open, the batch program is terminated abnormally and the supplied interface routines send an error message to the operator console.

3.22.2.2. 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'
```

Where:

No parameters required.

3.22.2.3. 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.

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 copy element (TIP/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.

Note: *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 copy element.

Section 4
TIP/30 System Generation

The TIP/30 system is generated by preparing a library element containing TIP/30 generation statements and parameters that define and select various aspects of the TIP/30 system.

This input stream contains generation control statements, parameters, keywords and options that are read and interpreted by a supplied batch program (TB$GEN).

The output of the TB$GEN program is a job stream containing:

- an ASSEMBLY step (to assemble the TIP/30 Control Area — TCA)
- a LNKEDT step to link the TCA into a load module and (if necessary) a load module that is used by the TIP/30 offline recovery program
- LIBS steps to copy these load modules to the appropriate libraries and to erase the job that was dynamically built by the TB$GEN program.

The TB$GEN program assumes the responsibility of conforming to assembler coding conventions.

The user prepares a library element containing free-format statements that specify the TIP/30 requirements for the site and specifies that library element as the input to the TJ$PARAM job (the parameter processor).

The parameter processor allows the use of comments in standard assembler format (an asterisk in column 1) and allows the use of a slash in column 1 to indicate "skip to a new page" to improve the readability of the printed output.

The parameter processor does not require the use of any sort of "continuation flag". Each statement begins with the name of the statement, any required *positional* parameters, followed by the keywords selected for that statement.

Example:

```
FILE PAYMAST,MIRAM  BLKSIZE=256 KEYLEN=8 KEYLOC=0
                    RECSIZE=256
                    HOLD=TR DELETE=(X'FF',8).
```

Each generation statement (not each line) should be terminated with a period after the last keyword that applies to that statement.

The output of the parameter processor is a printed report indicating the result of the analysis of the parameters. The parameter processor also generates a table that cross

references each file in the generation with the page number in the parameter listing and an indication of which files occupy various internal file buffers.

Each generation of TIP/30 must be given a TCA name. This name is used to identify the particular generation. It is possible and quite reasonable to have a number of "variants" of TIP/30 generated, each for a specific use.

TIP/30 does not dynamically reference the TCA once TIP/30 is running. This means that you can generate TIP/30 (even the TCA that is running) while TIP/30 is executing.

This practice may be somewhat cavalier — the recommended procedure is to establish a cycle of TCA names (the first 3 characters of the TCA name must be unique) to allow you to fall back to a working generation if there are problems with the latest generation parameters.

There are three basic generation control statements:

TIPGEN Selects general options. This statement is **required**. Only one TIPGEN statement may be specified.

FILE Defines an online file that TIP/30 may access. One FILE statement is required for each online file.

CLUSTER Define logical terminal groups and/or override generation options for specific terminals. CLUSTER statement(s) are optional.

These generation control statements have required positional parameters and optional keywords that specify variable information. The following sections describe the generation control statements and their parameter and keyword requirements.

Many of the keywords described have short forms. Keywords are illustrated as a combination of upper and lower case letters. The short form of a keyword is the sequence of just the upper case letters. You may omit lower case letters (but they must be correctly placed if specified!).

When a keyword requires a numeric value, the value is always interpreted as a decimal number (base 10). Do not enter numbers with any imbedded commas (for example: specify 30000 not 30,000).

When a keyword requires a character string as a value, enclose the value in single quotes only if the string contains a period or one or more imbedded spaces.

Some keywords do not have a meaningful default value. The presence (or absence) of the keyword dictates the action of the parameter processor. As a general rule, the facility or feature governed by the keyword is not relevant unless the keyword is present.

The TB$GEN program supports the use of a COPY statement as a means of including groups of generation statements from a library element. This coding convenience allows (for example) groups of related FILE statements to be included in different generation parameter streams.

Syntax:

```
COPY  lfd,elt
```

Where:

lfd The first parameter after the reserved word COPY is the LFD name of the library to read. The most appropriate place to include the job control for this library is in the TIP/30 job control proc named TIPLIBS.

This job control proc is included in the supplied job control for the TIP/30 parameter processor.

elt The second parameter is the element name to read. This library element is assumed to by a source type element.

4.1. TIPGEN Definition

The TIPGEN control statement begins the definition of the characteristics of the TIP/30 system. This statement must appear only once in a particular generation stream and must be the first generation statement encountered.

The information constructed by the generation process is known as the TIP/30 Control Area (TCA).

The TCA name is used to uniquely identify a set of TIP/30 generation parameters. The name of the TCA is a required execution-time parameter that must be supplied in the job control imbedded data set for TIP/30.

Note: *The TCA name must be at least 3 characters long, and the first 3 characters must be unique.*

Syntax:

```
TIPGEN tcaname
       keyword=value  keyword=value
       keyword=value  keyword=value.
```

The parameter processor treats the input as a string of text; more than one keyword *may be* specified on an input line — however, you may find it convenient to specify each keyword on a separate line to permit the lines containing keywords to be sorted (see the Full Screen Editor "SA" command).

Table 4-10. TIPGEN Statement

TIPGEN Statement		
Keyword	**Description**	**Default**
tcaname	Name assigned to this set of generation parameters.	Required.
AFT=	Average number of active files per program.	See text.
BaCK=	Maximum number of background programs.	2.
BackPRI=	Background transaction priority.	2.
B4=	Use quick before image file (TIP$B4).	See text.
CATPooL=	Number of catalogue pool entries.	6.
CURrency=	Currency symbol.	"$".
DBMS=	Database to be used.	-
DECIMAL=	Define decimal point character.	DECIMAL.
DMSAWT=	DMS area wait time (seconds).	0.
DMSRWT=	DMS record wait time (seconds).	0.

continued ...

TIPGEN Statement		
Keyword	**Description**	**Default**
EDiTstmp=	Default update stamping for FSE editor.	See text.
ESCape=	System escape character.	"@".
FaSTLoaD=	Fastload index size.	(25,50).
FCSxtent=	FCS Dynamic file extent size.	40.
FiLeBufs=	Number of file buffers.	3.
FileTab=	Internal FILE cat record table.	Yes.
FREEmem=	Size of TIP/30 free memory pool.	2560.
GDA=	Global data area size.	0.
IMS=	IMS emulation options.	-
IMSDT=	Delay time for IMS delayed internal to self.	0.
IMSROW=	Terminal row number for IMS messages.	1.
IMStranL=	IMS transaction code length.	5.
IMSUNSDT=	IMS emulator send messages to down terminals.	No.
JOB=	Next gen job name to create.	See text.
JouRNaL=	Use journaling to TIP$JRN disk file.	No.
KeYTaBLe=	Key holding table size.	-
LANGuage=	Local language.	ENGLISH.
LIST=	Generation list option.	No.
LOCAP=	Global ICAM locap name.	-
LOG=	Use journaling to TIP$LOG tape file.	No.
LoGoN=	Logon is required.	Yes.
MaXCaLLs=	Define limits on program CALLs.	(512,50000).
MaXPRoG=	Minimum required paged memory.	30000.
MAXTiMe=	Program timeout (seconds).	30.
MCSPooL=	TIP/30 screen format pool size.	(3,500).
NCS=	OFIS LINK national character set.	-
NETwork=	ICAM network name and password.	(TIPC,TIPPWD).
NumGRPS=	Number of elective groups for a user.	2.
PRintLF=	Default TIPPRINT LF= option.	No.
PRintLPP=	Default TIPPRINT lines per page.	60.

continued ...

TIPGEN Statement		
Keyword	**Description**	**Default**
PRintTOF=	Default TIPPRINT top of form.	No.
PRintTTL=	Default TIPPRINT header page wanted.	Yes.
PRintUC=	Default TIPPRINT uppercase translation.	Yes.
PRIority=	Number of transaction priority levels.	2.
PRSTEN=	Journal program start/end information.	No.
ReaDYmsg=	Send ready message at startup.	No.
RESMEM=	Configure IMS transaction buffers.	-
SchdPRI=	Execution priority of TIP/30 scheduler.	2.
SFSPooL=	SFS format pool size.	10.
shutDowN=	Transaction to run at TIP/30 EOJ.	-
SITEid=	Site identification.	TIP30.
startUP=	Transaction to run at TIP/30 startup.	-
STatS=	Statistics interval.	(15,240).
TeRMS=	Maximum terminals in network.	Required.
termSiZe=	Terminal screen size.	(24,80).
TermtYPe=	Type of terminal.	UNISCOPE.
TIMeoff=	Automatic logoff timeout (minutes).	10.
TIMeouT=	External succession timeout (minutes).	540.
TIPFILES=	JPROC to be called by job.	TIPFILES.
UpPeR=	Override TIP/30 uppercase translate table.	-
UserPRI=	Execution priority of transactions.	1.
WORK1=	ASM work file 1.	RES.
WORK2=	ASM work file 2.	RUN.
XMIT=	UTS400 control page option.	-
XmitALL=	UTS400 Fn key to XMIT ALL.	-
XmitCHan=	UTS400 Fn key to XMIT CHAN.	-
XmitVAR=	UTS400 Fn key to XMIT VAR.	-
. (period)	Mark last keyword specified.	

Where:

tcaname

Required positional parameter. A minimum of three characters is required — a maximum of 6 characters is allowed.

This name is used as the load module name for the TIP/30 Generation and must be specified as the first (positional) parameter of the TIPGEN control statement.

If any file in the generation is defined with HOLD=TR the generation process will also link edit a recovery load module. The recovery load module is assigned the load module name xxx$RC (where xxx is the first three characters of the tcaname) — this is why the first three characters of the TCA name must be unique.

AFT=n

The average number of active files expected per transaction program. This parameter allocates memory for the "Active File Table" for each executing online program.

If omitted, TIP/30 ensures that there is enough room for every terminal to have every file open concurrently. This is clearly a waste of memory at most sites; the parameter should be specified as a reasonable value.

When a program "opens" a file and the program's AFT is full, additional space is allocated from the TIP/30 Free Memory Pool (FREEmem=). Each entry in the AFT is small (approximately 32 bytes).

BaCK=n

The maximum number of background online programs that may be running concurrently.

One background (process) table is reserved for handling console operator commands. The value specified should take this into account.

Default: BACK=2

BackPRI=n

The default scheduling priority for programs that run in background.

The value specified represents an offset from (1 + the execution priority that is specified on the EXEC job control statement for TIP/30).

Default: 2 (implying EXEC+1+2)

If the value specified for this keyword exceeds the number of generated priority levels (PRIORITY=), the generation program sets BACKPRI=PRIORITY.

B4=YES

Configure a TIP$B4 file. The TIP$B4 file is used for online record rollback when transactions abort.

This parameter is independent of the specification for journaling (JRNL=YES) or tape logging (LOG=YES).

Whenever records of files defined as HOLD=TR are read with lock (ie: GETUP), a copy of the record image is placed in the TIP$B4 file. This "before image" may be used to roll back the update if the transaction aborts.

Default=YES if any file is defined as HOLD=TR.

CATPooL=n

The number of entries in the CATALOGUE record pool. All records read from the TIP/30 Catalogue are eligible to remain in the pool.

TIP/30 will keep this number of the most recently read catalogue records in memory.

TIP/30 catalogue entries (and therefore these pool entries) are 256 bytes in length.

Default: CATPOOL=6

The size of the CATPOOL can be overridden at execution time by specifying a new value for the pool for the file TIP$CAT by using the run-time keyword FiLePooL= in "TIP/30 Job Control Options" on page 6-1.

CURrency=

The floating currency symbol used by TIP/30 Screen Format System (MCS).

Default: CURRENCY=$ (a dollar sign).

DBMS=

DMS	Interface to single thread DMS.
DMT,n	Interface to multi-thread DMS, where "n" is the number of threads to allow.

DECIMAL=COMMA

Reverses the meaning of the decimal point and the comma for the purpose of editing numeric fields in TIP/30 screen formats.

Eg: 3.100,10 instead of 3,100.10

Default: DECIMAL=DECIMAL

DMSAWT=

DMS area wait time — defines the maximum time (in seconds) that DMS waits for an area to become available.

Default: 0 seconds (DMS will cause immediate rollback if the area is locked by some other DMS user).

DMSRWT=

DMS record wait time — defines the maximum time (in seconds) that DMS waits for a data base record to become available.

Default: 5 seconds

EDiTstmp=

This keyword controls the DEFAULT update stamping technique used by the TIP/30 Full Screen Editor (FSE).

Each user of FSE may change the update stamping technique during an individual FSE session.

YES	Use "standard" stamping (the version number in columns 73:75, an asterisk in column 76 of lines that change).
NO	No update stamping desired.
DATE	Place the current date (YYMMDD format) in columns 73:78 of lines that change.
USERID	Place the userid in columns 73:80.

Defaults: COBOL, DOCUMENT, and ASSEMBLER all default to "YES". Any other language type defaults to "NO".

ESCape=

Defines the TIP/30 system escape character.

Default: ESCAPE=@ (commercial at-sign)

If this character is the first character of any input message, TIP/30 "escapes" to the transaction name following the escape character.

Escaping to another transaction program causes the currently executing transaction to be suspended while the next transaction is executed. When the next transaction terminates, control returns to the original transaction — which is still awaiting the input message that was intercepted by TIP/30. For further information refer to the description of the TIPFORK subroutine call described in the Program Control System documentation.

Certain characters cannot be specified as the system escape character, namely: period, comma or less-than. A popular alternative character is the vertical bar character (|).

To disable the Escape feature of TIP/30, specify ESCAPE=NO.

FaSTLoaD=(m,n)

If this keyword is specified, TIP/30 maintains a table of "fast load information" that facilitates the rapid loading of program load modules. By using information in this table, TIP/30 can minimize operating system overhead when loading load modules from the TIP$LOD library.

Two subparameters are provided to maintain downward compatability with previous TIP/30 releases. The two values are summed to compute the desired number of table entries. Each entry in the table is approximately 20 bytes of data.

Default: FASTLOAD=(25,50)

FCSxtent=n

The number of blocks of the TIP$RNDM file used as both the initial and the secondary allocation for a TIP/30 dynamic file (edit buffers are specific implementations of TIP/30 dynamic files).

Default: FCSXTENT=40.

A dynamic file may have a maximum of 48 logical extents (each of which is the number of blocks specified by the value of FCSxtent=).

The value of FCSEXTENT= cannot be changed if there are any existing dynamic files. See the the description of the batch program TB$CRB in "Batch Program TB$CRB" on page 8-14 — the restore operation can specify a new FCSxtent= value when reloading the TIP$RNDM file.

FiLeBufs=n

The number of internal file buffers that TIP/30 is to configure.

Each file buffer holds the system control blocks, index area, I/O area and work area for one or more files.

Default: FILEBUFS=3

Minimum: 3

Maximum: 25

Each file defined in the TIP/30 generation parameters must be specified to be either a resident file or assigned to a file buffer.

Files that are assigned to the same buffer share the buffer space. When TIP/30 needs to perform I/O operations for a file, the control blocks and I/O areas for the file may have to be swapped into the file buffer.

Assign unrelated files to the same buffer to minimize the need to swap or exchange buffer contents.

FileTab=

Specifies whether TIP/30 is to maintain an internal table of file catalogue record information (this applies only to data management files).

Default: FILETAB=YES

TIP/30 can access catalogue record information for (data management) files faster when this table is configured. Each table entry is approximately 24 bytes of data.

FREEmem=n

The amount of additional memory to be reserved for the TIP/30 "free" memory pool. TIP/30 uses this memory pool for occasional functions. For example an OS/3 spool work area required by the SPL and TLIB programs is approximately 1500 bytes.

TIP/30 will always reserve at least 3K bytes — the actual amount that is reserved in advance depends on such variables as the number of terminals.

The amount specified by this keyword is in addition to the amount that TIP/30 reserves in advance.

Default: FREEmem=2560.

TIP/30 attempts to use this free memory area to hold the contents of the CDA during inter-program transfer of control (the alternative is to use the TIP$SWAP file).

In an environment that supports a great deal of TIPSUB, TIPXCTL, or TIPFORK activity, a large specification for FREEmem= may be a wise use of (spare?) memory.

The IMS emulator always uses the TIP$SWAP file to hold the CDA during IMS **external** succession — the CDA may be held in free memory for other types of succession.

GDA=n

The size (in bytes) of the optional Global Data Area.

The address of the Global Data Area (GDA) is passed as a parameter to TIP/30 transaction programs. The GDA may be used as a common storage area by all TIP/30 transactions. For further information, refer to the section of the reference manual entitled "PCS — Program Control System".

Default: GDA=0

IMS=

Unusual IMS emulation options. Omit this keyword unless you specifically need one or more of the "features" offered.

Specified as (up to) seven parameters. The order of the parameters is not significant.

D2I	Interpret delayed internal succession as internal succession.
DNOWRK	Do not clear WORKAREA on delayed internal succession.

IMANULL Clear Input Message Area (IMA) to low-values (X'00') instead of spaces.

MT Emulate IMS multi-thread (this is the default).

RETHLD Coerce the IMS emulator to return "record held" status on a call to GETUP if the record is locked by another transaction progam. This status is normally only returned by IMS single thread; by using this parameter, the IMS emulator will return record held status whether or not single thread is specified.

The status code returned is (3,18) — decimal.

SFS Support Screen Format Services (SFS).

ST Emulate IMS single-thread. The IMS emulation routines will enforce single thread processing (an IMS transaction must complete before another IMS transaction may begin).[22]

Example: IMS=(MT,IMANULL,SFS)

Default: the IMS emulator will behave according to the specifications of the Unisys documentation for the IMS multi-thread product.

IMSDT=n

A value specifying the number of seconds that an IMS program is to be delayed (via an internal call to TIPTIMER) if the program performs delayed internal succession to itself.

A modest delay prevents such programs from completely monopolizing the processor.

Default: 0 seconds

IMSROW=n

Define the terminal row number where IMS system messages are output.

Examples of such IMS messages are: "Transaction completed" and "No Message in Queue".

Default: IMSROW=1

IMStranL=n

The maximum number of characters in an IMS transaction name for IMS programs running under emulation.

Default: IMSTRANL=5

IMSUNSDT=

YES/NO option controlling whether or not the IMS emulator should send messages to terminals that are down (and let ICAM queue them) or to return appropriate error status.

Default: IMSUNSDT=NO (do not send messages to down terminals).

22. This has no effect on TIP/30 transactions which continue to run multi-thread.

JOB=(jobname,queue,acct,execpri)

Up to four positional parameters defining the attributes of the JOB that is created by the parameter processor (as the second step of the TIP/30 generation process).

The order of these parameters is significant.

jobname Job name to create. Default "TJ$GEN".

queue Job queue to be used to run the job (Low, Medium, High, Preemptive). Default is H — high queue.

acct Job account number for the job (default is "TIP").

execpri Execution priority to be placed on each EXEC statement in the generated job. Default: no explicit execution priority is specified (job runs at the default batch job priority).

Note: *The generated job will be written to the YJCS library unless a library with an LFD name of "TIP$JCS" is defined in the job control for the TJ$PARAM job.*

JouRNaL=YES

The TIP$JRN file is to be used by TIP/30 to journal specified file before/after images etc.

Default: JOURNAL=NO

WARNING

Specifying JOURNAL=YES causes the default value for ALL (non-library) files to be JOURNAL=YES and AFTER=YES.

If you specify JOURNAL=YES in the TIPGEN statement remember that you must specifically state JOURNAL=NO for files that you do not want journaled.

KeYTaBLe=

=(m,n) The value "m" is the maximum number of records that may be concurrently held for update within the TIP/30 system. For a file that is generated with HOLD=TR, ADD, CHANGE and DELETE operations are considered an update operation.

The value "n" is the size in bytes of the largest key that may be held for update.

KeYTaBLe=m

An alternative way of specifying the maximum number of records that may be concurrently held for update by the TIP/30 system.

The second format (specifying only a single value) is recommended for the KEYTABLE= specification because it leaves the computation of the largest key to the parameter processor. This eliminates the possibility of overlooking a file and understating the size of key holding table entries.

If an online program (TIP/30 native mode or an IMS program running under emulation) attempts to lock a record (this includes ADD, DELETE or CHANGE operations) and the TIP/30 key holding table is full, the program is aborted.

The default number of entries in the key holding table is set to allow 1 entry for each file that is generated as HOLD=YES and 3 entries for each file that is generated as HOLD=UP or HOLD=TR.

LANGuage=

The desired local (or default) language used by TIP/30.

Default: ENGLISH

Other choices currently supported are: FRENCH, GERMAN, SWEDISH, FINNISH, ITALIAN, SPANISH.

This setting affects the result of a call to the TIPDATE subroutine and the value that is returned by the MCS special heading field $DDMMMYY$

LIST=YES

List all of the generated code for the TIP/30 system generation. This is not recommended because it causes the next job (an assembly) to generate an expanded output listing.

Default: LIST=NO

LOCAP=

The name of the Local Application (LOCAP) in a Global ICAM system that TIP/30 is to use.

This parameter may be overridden by the identically named run-time job control option.

No default value.

If this keyword is omitted in both the generation parameters and in the run-time options, TIP/30 assumes that the interface is with a dedicated ICAM.

LOG=YES

Indicates that journal data is to be written to the TIP$LOG log tape.

Default: LOG=NO

> **WARNING**
>
> Specifying LOG=YES causes the default value for all (non-library) files to be JOURNAL=YES and AFTER=YES.
>
> If you specify LOG=YES in the TIPGEN statement remember that you must specifically state JOURNAL=NO for files that you do not want journaled.

LoGoN=NO

Specifies that terminals do not require users to log on to TIP/30 before running transactions.

You may alter this specification on a CLUSTER basis by coding in subsequent CLUSTER statements.

Default: LOGON=YES

MaXCaLLs=(sw,abort)

Two subparameters that establish thresholds on certain types of TIP/30 CALLs:

sw Maximum number of calls that a transaction program issues before getting automatically switched to lowest priority level (default is 512 calls).

abort Maximum number of calls that a transaction program issues before getting automatically **aborted** (default is 50000 calls).

The internal counter that is compared to the abort limit is "reset" whenever a call is issued that allows the process to be eligible to be swapped out of main memory (eg: TIPTIMER, TIPMSGI, TIPSUB, TIPXCTL etc). Calls to TIPFCS do NOT allow swapping to occur and therefore do not cause the counter to be reset.

Setting the abort limit to a less generous value may help catch programs that would otherwise loop forever (continuously reading the same record, for example).

MaXPRoG=n

The minimum number of bytes to reserve for "paged memory". At initialization, TIP/30 ensures that at least this amount of memory is allocated to paged memory (the memory used to hold online program load modules and activation records).

If this amount of paged memory cannot be allocated, TIP/30 initialization fails to complete. In this case, a larger memory allocation on the JOB card is required.

Default: MAXPROG=30000

MAXTiMe=n

The maximum time (in seconds) that may elapse before an online program calls a TIP/30 subroutine.

If an online program does NOT call a TIP/30 subroutine within this number of seconds, TIP/30 aborts the program[23] under the assumption that the program is probably looping or is monopolizing the TIP/30 system.

Programs that consume a large amount of CPU time should issue a call to the TIPTIMER subroutine to surrender control of the processor at convenient intervals (this practice satisfies the MAXTIME= constraint and prevents the program from starving other transactions).

Default: MAXTIME=30 (seconds).

23. With a fictional "Process Timeout" exception error code.

MCSPooL=(m,n)

Define main storage pooling for TIP/30 screen formats. Value "m" is the number of TIP/30 screen format pool entries reserved in memory; "n" is the size of each pool entry in bytes.

Default: MCSPOOL=(3,500)

TIP/30 screen formats that have an internal size greater than "n" bytes cannot be pooled in memory and must always be read from the TIP$MCS file.

The internal size of TIP/30 screen formats is displayed by the MSGAR utility transaction and is summarized by the batch job TJ$LC.

The number of screen formats to pool (but not the size of the pool entry) can be overridden at execution time by specifying a new value for the pool for the file TIP$MCS by using the run-time keyword FiLePooL= in "TIP/30 Job Control Options" on page 6-1.

NCS=

National Character Set for OFIS Link/80. Refer to separate documentation describing the TIP/30 interface to OFIS Link/80.

NETwork=(cca,pwd)

cca The ICAM CCA name.

pwd The ICAM CCA password (optional).

Default: NETWORK=(TIPC,TIPPWD)

You may override this parameter with a TIP/30 job control specification.

NumGRPS=

The number of elective groups to which a user may belong.

The number of elective groups specified here is in addition to the user's private group (the userid) and the universal group (TIPY).

Default: NUMGRPS=2

Maximum: 16

Minimum: 2

PRintLF=

Default: PRINTLF=NO

Default TIPPRINT option for appending a line feed to each message transmitted to an AUX device. TIPPRINT normally buffers a number of output print lines so that a screen full of data can be sent to the terminal and printer as one output message.

Some printers automatically provide a line feed character when a screen full of data is sent to the printer. Many printers do not provide this option; in this case, TIPPRINT must append a Line Feed character to the end of every buffer that is output.

If an online program does not specify a value for the PRINT-LINE-FEED field when the TIPPRINT interface is opened, this specification (possibly overridden by a CLUSTER statement) is taken as the default by TIPPRINT.

If this value is incorrectly specified, reports printed on auxiliary printers may appear with an occasional extra blank line or a missing blank line.

Specify this keyword as required by the majority of auxiliary printers in use at your site and override the specification on CLUSTER statements for the terminals or printers that are different.

PRintLPP=n

Default TIPPRINT value for number of lines per page for an auxiliary device.

If an online program does not specify a value for the PRINT-PAGE-LEN field when the TIPPRINT interface is opened, this specification (possibly overridden by a CLUSTER statement) is taken as the default by TIPPRINT.

Default: PRINTLPP=60

PRintTOF=YES

Default TIPPRINT value for guaranteeing a form feed before and after printing to an auxiliary printer.

When YES is specified, TIPPRINT ensures that a form feed is issued when an FCS-OPEN function or an FCS-CLOSE function is performed.

If the first (or last) output via TIPPRINT already causes a form feed, TIPPRINT does not insert an extra one.

If an online program does not specify a value for the PRINT-TOP-OF-FORM field when the TIPPRINT interface is opened, this specification is taken as the default by TIPPRINT.

Default: PRINTTOF=NO

PRintTTL=

Default TIPPRINT value that controls whether or not TIPPRINT prints a title page.

If a user online program does not specify a value for the PRINT-TITLE field when the TIPPRINT interface is opened, this specification is taken as the default by TIPPRINT.

Default: PRINTTTL=YES

PRintUC=YES

Default TIPPRINT value for forcing alphabetic characters in a print line to upper case.

If an online program does not specify a value for the PRINT-UPPER-CASE field when the TIPPRINT interface is opened, this specification is taken as the default by TIPPRINT.

Default: PRINTUC=YES

PRIority=n

The number of transaction priority levels to establish. These priority levels are relative to (1 + the execution priority specified for TIP/30 in the EXEC statement in the TIP/30 job control).

Default: PRIORITY=2 (implies EXEC+1+1 and EXEC+1+2)

Range: 2 through 10 inclusive

Example: PRI=4 and TIP/30 executing at priority 1, means that transactions may run at levels 3 through 6.

See also discussion in PCS section of this manual.

PRSTEN=YES

Specifies whether statistical journal records PRST (program start) and PREN (program end) are to be written to the TIP$JRN or TIP$LOG file.

Default: PRSTEN=NO

This keyword has no effect unless TIP/30 journaling (or logging) is in effect (JOURNAL=YES or LOG=YES).

ReaDYmsg=YES

TIP/30 must send a "ready" message (a greeting) to terminals that connect to TIP/30.

Default: READYMSG=NO

The text of the ready message is specified by the job control keywords Banner1='...' and Banner2='...'.

You may override this specification on a CLUSTER basis or in the TIP/30 job control options.

Note: *This ready message is always sent to local workstations and dynamic sessions (those that use $$SON in a Global ICAM environment).*

Specification of this option may increase ICAM ARP utilization (remember that an ARP shortage in ICAM is usually a fatal error).

RESMEM=(n1,n2,n3)

This keyword configures IMS transaction buffers. This facility may be required to emulate IMS transaction programs that make use of transaction buffers.

The three values specified represent, respectively:

1. the number of buffers to establish in the pool (this value is required if the RESMEM= keyword is specified)
2. optional maximum number of buffers to acquire if the main pool is depleted (default for this value is zero)
3. the maximum number of buffers that a particular transaction may acquire; this parameter may not exceed 16 — default value is 4.

If this keyword is omitted, no IMS transaction buffers are established.

For additional information see *IMS Support Functions Programming Guide — UP-11907*.

SchdPRI=n

The relative execution priority of the TIP/30 scheduler task.

The value specified represents an offset from (1 + the execution priority that is specified on the EXEC job control statement for TIP/30).

Default: 2 (implying EXEC+1+2)

You should not specifiy a value that is **LESS THAN** the value specified for USERPRI=.

SFSPooL=n

The number of SFS (Screen Format Services) screen formats to be pooled in main storage.

Default: SFSPooL=10 (max is 255).

This keyword is only relevant if TIP/30 is running IMS programs (under emulation) that are using Unisys Screen Format Services (SFS).

This value is used by SFS to pool SFS screen formats in main storage. The pool is shared by all TIP/30 terminals accessing Screen Format Services.

shutDowN=trid

The transaction name of an online program automatically started in background (via a call to TIPFORK) when the TIP/30 EOJ console command (or transaction) is executed.

The shutdown program will not be started until there are no users remaining on the TIP/30 system.

You must catalogue this program in group TIPY or a group to which the optional userid *CONSOLE* has access.

Default: no program will be scheduled

Also see the description of the SHUTDOWN utility transaction.

SITEid=siteid

A character string to use as a descriptive identifer for the TIP/30 system. This string is often set to the company's name or to some string that represents the purpose of the TIP/30 system such as "Test System" etc.)

This information is used to construct the TIP/30 ready message.

You may specify up to 12 characters (in quotes if the string contains a period character or one or more space characters).

Default: SITEID=TIP30

startUP=trid

The transaction name of a program that is automatically started in background (via a call to TIPFORK) as soon as TIP/30 has completed initialization.

You must catalogue this program in group TIPY or a group to which the optional userid *CONSOLE* has access.

Default: no program will be scheduled

Also see the description of the STARTUP utility transaction.

STatS=(jrn,con)

jrn Specifies the time interval (in minutes) to control writing of statistics records (type code "STAT") to the TIP$JRN or TIP$LOG file.

Default: 15 (minutes). Specify zero if no STAT type records are to be written.

con Specifies the time interval (in minutes) to control the display of TIP/30 statistics on the system console.

Default: 240 (4 hours). Specify zero to inhibit the display of this information on the console.

TeRMS=n

Maximum number of terminals TIP/30 may attach.

In a dedicated ICAM network, the specified value must be greater than or equal to the number of TERM statements specified in the ICAM generation.

In a GLOBAL ICAM network, the specified value determines the maximum number of concurrent sessions that may be active (this includes static and dynamic sessions).

This keyword must be specified.

termSiZe=(rows,cols)

The number of rows and columns (respectively) for terminals in the network.

Default: TERMSIZE=(24,80)

You may override this value on a CLUSTER basis by specifications in a subsequent CLUSTER statement.

TermtYPe=

The most common type of online terminal in the network.

Default: TERMTYPE=UNISCOPE

Other choices:

U10	U20	U20D	U30	U40	U400
U400F	TTY	U200	SPC	SVT	

U400F is a U400 with the character protect feature installed.

SPC is a Personal Computer which is emulating a Uniscope Terminal by using a Computer Logics PEP board or a Unisys STEP board and related PC software.

SVT is an SVT-112x terminal

You may override this value on a CLUSTER basis by specifications in CLUSTER statement(s).

TIMeoff=n

The time (in minutes) that must elapse before TIP/30 automatically logs off an idle user.

A user is considered idle if he is logged on and waiting at the TIP/30 system prompt (example: TIP?▸).

Default: TIMEOFF=10

TIMeouT=n

The maximum amount of time (minutes) that an IMS program waits in external succession or a TIP/30 program waits for TIPDXC (delayed transfer control).

Default: TIMEOUT=540 (also the maximum allowed value).

TIPFILES=

Name of a job control PROC used (instead of TIPFILES) by the next step of the generation procedure (TJ$GEN job).

Default: TIPFILES=TIPFILES

UpPeR=

Override the translate table that TIP/30 uses to map lower case alphabetics to upper case.

This keyword is used primarily in European countries to control the correct character set mappings for local graphic characters.

The parameters are pairs of character values representing the hex value of the lower case character and the corresponding upper case character (respectively).

Example: UPPER=(X'81',X'A1',X'82',X'A2') convert X'81' to X'A1' etc.

Default: TIP/30 uses the standard (English) translate table.

UserPRI=n

The transaction scheduling priority for transactions that are NOT running in background.

The value specified represents an offset from (1 + the execution priority that is specified on the EXEC job control statement for TIP/30).

Default: USERPRI=1 (implying EXEC+1+1)

The execution priority of a particular transaction may be explicitly specified in the TIP/30 catalogue entry for the PROGram — the value specified for this keyword defines the desired **default** transaction priority.

WORK1=vol[,dvc]

vol The disk volume that is to be specified for the WORK1 file in the generation job (TJ$GEN) that is automatically scheduled by the parameter processor.

Eg: WORK1=TEMP01,50

The abbreviations "RES" or "RUN" are allowed.

dvc The device number to be used (may be omitted if the "vol" parameter was specified as RES or RUN).

Default: use WORK1 job control proc.

WORK2=vol[,dvc]

Similar to the above description of WORK1.

XMIT=

TIP/30 alters the terminal control page XMIT value (at LOGON time) to this value (for terminals with a control page!).

Choices: ALL CHAN VAR

Default: no alteration of the control page

XmitALL=n

Specifies a function key number that is interpreted by TIP/30 as a request to place the cursor in the bottom right corner of terminal followed by a TRANSMIT ALL sequence.

Eg: XMITALL=22 causes function key 22 to behave in the manner described above.

Default: functionality not assigned.

XmitCHan=n

Similar to XMITALL= except that a TRANSMIT CHANGED sequence is performed.

Default: functionality not assigned.

XmitVAR=

Similar to XMITALL=, except that a TRANSMIT UNPROTECTED sequence is performed.

Default: functionality not assigned.

. (period)

Place a period at the end of the last keyword specified for the TIPGEN statement to signal the end of the TIPGEN statement.

4.2. FILE Definition

Every online file must be defined in the TIP/30 Generation parameters. A minor exception is made in the case of OS/3 SAT libraries that may be dynamically accessed by TIP/30.

TIP/30 must construct the control blocks required by OS/3 Data Management to be able to access the files.

With the exception of OS/3 libraries, the following rules apply:

- A file must be defined in the TIP/30 generation to be accessed via TIP/30
- The job control statements for the file must be present in the TIP/30 run-time job control stream.

The definition of a file in the TIP/30 generation parameters must correctly reflect the physical characteristics of the actual file or unpredictable results may occur.

You must designate the files that are defined to TIP/30 as **resident** or **buffered**. Identifying a file as a resident file means that the control blocks and I/O areas for that file are permanently allocated space in the TIP/30 memory region and I/O to the file may proceed directly.

The control blocks and I/O areas of buffered files share a "file buffer". TIP/30 may have to swap occupants of file buffers from main storage to disk (and back) be able to perform I/O for a file that is currently not in the file buffer.

Files that are accessed frequently are good resident file candidates (memory permitting of course); designate infrequently used files as buffered files.

A library cannot be specified as a resident file.

TIP/30 restricts the number of buffered files to 99. This is not an arbitrary restriction — each buffered file is implemented as an OS/3 load module overlay of the TCA. OS/3 has a limit of 99 overlays since the last two characters of the load module name are reserved for the two-digit overlay number.

Several files are automatically known to TIP/30 and cannot be explicitly stated in the generation parameters (although they MUST be defined in the TIP/30 Catalogue):

- system libraries (YJCS, YSRC, etc)
- the TIP and TIP$LOD libraries
- the system printer (PRNTR)
- the system punch (PUNCH).

To define a file in the TIP/30 generation parameters, the FILE statement is used. One FILE statement is coded for each online file. The FILE statements appear after the TIPGEN statement described in the previous section.

Syntax:

```
FILE  lfdname,type
      keyword=value    keyword=value
      keyword=value    keyword=value.
```

Table 4-11. FILE Statement

FILE Statement	
Keyword	**Description**
lfdname,type	Required positional parameters.
ACCess=	File share access.
AFTER=	Journal after images.
AUTOIO=	CDM file, force I/O.
BEFoRe=	Journal before images.
BLKsiZe=	Block size.
BUFfer=	File buffer number.
CLOSE=	Leave file initially closed.
COMSToRe=	Common storage file.
DeLeTe=	Deleted record convention.
FILesiZe=	Maximum number of records.
HoLD=	Type of record locking.
INDsiZe=	Index buffer size.
IO=	INPUT/OUTPUT/INOUT file.
JouRNaL=	Journal or log changes to this file?
KeyHoLD=	Bytes of key to hold.
KeYLeN=	Length of (only) key.
KeYLoC=	Bytes ahead of (only) key.
KEY1=	MIRAM Key 1.
KEY2=	MIRAM Key 2.
KEY3=	MIRAM Key 3.
KEY4=	MIRAM Key 4.
KEY5=	MIRAM Key 5.
MuLTiSeQ=	Allow multiple sequential readers.
OPEN=	File open procedures.

continued ...

FILE Statement	
Keyword	**Description**
OPTioNal=	Optional file.
PCYLofl=	Percent cylinder overflow.
PKEY=	Identify which key is primary key.
POOL=	Number of records to pool.
PRINTOV=	Forms overflow reporting.
ReCForM=	Record format.
RECsiZe=	Record size.
RESident=	Control blocks resident.
RCB=	MIRAM file with RCB?
USEFiLe=	Use parameters of another file.
VSEC=	Variable sector size.
. (period)	Mark end of last keyword for this file.

Where:

lfdname The LFD name of the file as specified in the TIP/30 job control.

This positional parameter is required.

type The type of file (see discussion following). This positional parameter is required.

Valid file types are:

DAM Direct access.

DMIRAM Relative MIRAM.

IRAM IRAM (keyed).

ISAM Indexed Sequential.

LIB SAT library.

MEM Memory resident DMIRAM.

TIP/30 supports a file type "MEM" — indicating that the records of the file (that is treated as relative MIRAM) are all to be kept in main storage. These records must be loaded in advance (often a system startup transaction is used to accomplish this).

MIRAM Indexed MIRAM.

PRINT Sequential print file (spooled by operating system).

PUNCH Sequential punch file (spooled by operating system).

SAM Sequential file.

SMIRAM Sequential MIRAM.

TAPE Sequential TAPE file.

ACCess=

The type of access to the file needed by TIP/30.

Default: ACCESS=EXCR (TIP/30 has exclusive access for updating, but allows other readers).

If a "// DD ACCESS=" statement appears in the job control definition of this file, that value overrides the value specified in the TIP/30 generation parameters.

Although Data Management supports "SADD" (shared add), and "UCP" (user controlled by program) these access types are NOT recommended for use with files that are accessed by TIP/30.

Performance degradation and/or data corruption is possible with the use of SADD or UCP access!

Reference: *OS/3 Consolidated Data Management Concepts and Facilities (UP-9978)* section 5.4.3.

AFTER=

YES specifies that "after" images of modified records (this includes added, changed, or deleted records!) are to be written to the TIP$JRN or TIP$LOG file.

Default: AFTER=YES (if journaling or logging is in effect for this file), otherwise AFTER=NO.

AUTOIO=

This keyword applies only to files which are accessed using CDM (Consolidated Data Management).

Default: If TIP/30 is generated with Consolidated Data Management support (CDM=YES or CDM=MIXED) and OS/3 release is 11 or later, the default is AUTOIO=WRITE. Otherwise, default is AUTOIO=NO

NO Specifies that read requests may be satisfied from data already in memory and write operations must always be "written through" to the disk.

TIP/30 internally specifies the appropriate settings for Data Management to control read and write operations.

YES Specifies that a physical I/O must be performed for read and write operations for this file - irrespective of the contents of memory.

TIP/30 internally specifies the appropriate settings for Data Management to control read and write operations.

WRITE A version of AUTOIO=NO (see above) that is performed by OS/3 Data Management rather than dynamically specified by TIP/30.

The AUTOIO=WRITE specification may be used only with MIRAM files accessed via Consolidated Data Management and using OS/3 release 11 (or later).

This specification is highly recommended (assuming that Release 11 or later is in use).

BEFoRe=

BEFORE=YES specifies that "before" images of changed records (includes added, changed, or deleted records!) will be written to the TIP$JRN or TIP$LOG file.

Default: BEFORE=NO

BLKsiZe=n

The block size (I/O buffer area size) for this file. This parameter is required.

This value must be a multiple of the actual VSEC size of the file and may need to be larger than otherwise obvious because a record may span more than one VSEC.

For MIRAM files created with RCB specified, the BLKSIZE must take into account the presence of an additional byte per record that is needed for the record control byte (RCB).

For ISAM files, the BLKSIZE must include an additional 5 bytes per record and an additional 2 bytes per block.

The value specified must be at least the minimum required by Data Management; a larger value may be specified (at the expense of additional

memory!) to attempt to enhance the performance of certain file operations (sequential reading, for example).

The following computation algorithm is adapted from the formula given in *Consolidated Data Management Macroinstructions* (UP-9979) section 2.3.1, for the keyword parameter BFSZ= for MIRAM files:

```
MOVE    [value in gen for    VSEC=]    TO LSS.
MOVE    [value in gen for RECSIZE=]    TO S.

IF   RCB=YES  AND  RECFORM=FIXED
     COMPUTE  S  =  S + 1.

DIVIDE  S  BY  LSS  GIVING    N  REMAINDER R.

IF R=0
   COMPUTE  BLKSIZE = N * VSEC
ELSE IF  [R evenly divides LSS]
         COMPUTE  BLKSIZE = (N + 1) * VSEC
     ELSE
         COMPUTE  BLKSIZE = (N + 2) * VSEC.
```

Table 4-12. Example BLKSIZE Calculation

RECSIZE	RCB?	VSEC	Min BLKSIZE
256	N	256	256
256	Y	256	512
258	Y	256	768
767	Y	1024	2048

BUFfer=n

The number of the file buffer to which this file is assigned.

This keyword allows the user to assign a file to a specific file buffer (the number of such buffers is specified by the FILEBUFS= keyword in the TIPGEN statement).

This keyword and the specification RESIDENT=YES are mutually exclusive (a file is either buffered or resident).

Default: the generation process assigns files to the available file buffers in a round-robin fashion.

The maximum number that may be specified as a buffer number is the value specified for the keyword FILEBUFS= in the TIPGEN statement.

Note: *File buffers 1, 2 and 3 automatically are used by TIP/30 for files such as the TIP and TIP$LOD libraries, the PRNTR and PUNCH files and for system libraries such as YJCS etc.*

At the end of the printed report generated by the TJ$PARAM job there is a matrix of buffer numbers versus LFD names that may occupy the buffer.

Examine this matrix to look for gaffes such as:

- *only one file in a buffer (might as well make that file resident)*
- *files which are used concurrently in the same buffer (this situation often results in excessive swapping operations).*

CLOSE=YES

Specify CLOSE=YES to instruct TIP/30 to not open the file automatically at system startup.

The file must be explicitly opened by the console operator (via the OPEN command) or a terminal user (via the FOPEN transaction) before it may be used.

Default: CLOSE=NO

COMSToRe=

Establish file as a "common storage file". All of the records in a common storage file are kept in main storage. The records are initially loaded into memory during TIP/30 initialization.

YES Specifies that updates to this common storage file are maintained both in memory and on disk. Updated records are written to the disk file and updated in memory.

NOWRITE Specifies that updates to this common storage file are maintained *ONLY* in memory. Updated records are NOT written to the disk file.

Common storage files are restricted as follows:

- file must be ISAM or *single key* MIRAM
- HOLD=UP, RESident=YES and JOURNAL=NO are forced
- POOL= specification is ignored
- records are NOT written to disk file at system shutdown or file close
- FCS-ADD, FCS-DELETE and FCS-FLUSH operations are not allowed.

DeLeTe=

This keyword controls the way TIP/30 deletes records from this file. The two choices are: logical delete flag and record control byte (RCB). The latter option is available only for MIRAM files.

Default: logical delete flag; X'FF' in the first byte of the record *that is not part of a key.*

DeLeTe=(flag,loc)

Selects logical delete flag scheme and specifies the flag character and the location of the character within the record.

flag Specify the flag character as X'??' or C'?' Default for "flag" is X'FF' (HIGH VALUES).

loc the number of bytes that **precede** the byte of the record that is reserved for the delete flag.

Default for 'loc' is the first byte of the record that is NOT part of a key field.

If the specified byte contains the designated flag character when a record is read from this file, TIP/30 places the record contents in the user's record area but returns PIB-NOT-FOUND status (to indicate that TIP/30 has detected this magic flag value).

This logical delete flagging is merely an agreed convention between TIP/30 and user programs — the record is a perfectly valid record.

DeLeTe=RCB

Specifies that TIP/30 is to use the Record Control Byte (RCB) method of deleting records for this file.

The RCB option must be requested for the file by the job that initially OPENS the file. (// DD RCB=YES)

When this format of the DELETE keyword is used Data Management unconditionally flags deleted records and no program, online or batch, is able to read such records (contrast this with the preceding description of logical delete flag).

The record control byte is not accessible by programs; its presence affects the calculation of the value to be specified for the BLKSIZE= keyword discussed earlier.

By specifying DELETE=RCB, the default value of the RCB= keyword is forced to RCB=YES (for this file).

FILesiZe=n

The maximum number of records in the file. This parameter is required for files defined as type MEM (memory files) and may not be specified for other types of files.

HoLD=

This keyword defines the type of record holding that is desired for this file. The three available techniques are described below.

For a more detailed description of the various techniques, refer also to the section of the TIP/30 manual that describes transaction end (in the PCS section) and the section on "record holding" (in the FCS section).

To read a record and lock the record for update, a transaction program first issues a call to TIPFCS with a function code of FCS-GETUP. To perform the update the program uses an FCS-PUT function code.

The default for the HOLD= keyword is HOLD=YES.

TR Indicates that record update locks for this file are maintained until the end of the transaction.

A transaction program may hold multiple records from this file defined as HOLD=TR.

If the transaction aborts, such records may be "rolled back".

Use of this keyword (on any file) forces the use of a TIP$B4 file (TIPGEN: B4=YES).

UP Indicates that records will be held until they are updated (or the lock is released by FCS-NOUP).

There is no provision for online record roll back.

A transaction program may hold multiple records from a file defined as HOLD=UP.

YES Indicates that a record from this file is held until the record is updated or released OR another GETUP is made to this file.

Refer to the section of the TIP/30 reference manual describing transaction end (in the PCS section) and the section on "record holding" in the FCS section.

A transaction program may hold only ONE record at a time from a file defined as HOLD=YES.

INDsiZe=n

The INDEX BUFFER SIZE for this file.

Default: INDSIZE=256.

This value MUST match the value specified when the file was initially opened (// DD INDS=)

IO= The type of I/O operations that may be performed on this file. The valid choices are:

- INPUT
- OUTPUT
- INOUT

This parameter is **required** for sequential files.

Default: IO=INOUT (for non-sequential files).

INOUT is not supported for sequential files.

JouRNaL=

YES or NO to indicate whether changes to this file are to be journaled (to TIP$JRN file) or logged (to TIP$LOG file).

Default: whatever was specified in either the JOURNAL= or LOG= keyword in the TIPGEN section.

This keyword is not valid for library files.

> **WARNING**
>
> Since the default is taken from the JOURNAL= keyword in the TIPGEN statement, we recommend that this keyword be explicitly stated for each file (if the TIPGEN value was thoughtlessly changed, critical journal information may be accidently turned OFF or ON).

KeyHoLD=n

The number of bytes of the primary key that TIP/30 must keep in the key holding table to enforce TIP/30 record locking.

Default: the full length of the primary key

You may feel that some subset of the full key is sufficient for record locking purposes. Think about this carefully! Choosing too small a number of bytes could cause programs to receive PIB-HELD status when they least expect it.

Refer also to the discussion of record locking in the FCS section of the TIP/30 documentation.

KeYLeN=n

The length of the (only) key for the file.

Omit this keyword for indexed MIRAM files (use KEY1= etc).

KeYLoC=n

The zero relative location of the key in the record (the number of bytes preceding the key).

Default: KEYLOC=0

Omit this keyword for indexed MIRAM files (use KEY1= etc).

KEY1=(length,loc,DUP/NDUP,CHG/NCHG)

Defines MIRAM index 1. See also description of PKEY= keyword.

A choice must be made to allow duplicate key values in this index (DUP) or to not allow duplicate key values (NDUP). A choice must also be made to allow this key to change (CHG) or to prevent key value changes (NCHG).

Note: *Data Management restricts a MIRAM key field to a minimum of 1 byte and a maximum of 80 bytes.*

The "location" specified is zero-relative (it is convenient to think of the location value as the number of bytes that precede the first byte of the key field).

KEY2=(length,loc,DUP/NDUP,CHG/NCHG)

Defines MIRAM index 2. See also description of PKEY= keyword.

KEY3=(length,loc,DUP/NDUP,CHG/NCHG)

Defines MIRAM index 3. See also description of PKEY= keyword.

KEY4=(length,loc,DUP/NDUP,CHG/NCHG)

Defines MIRAM index 4. See also description of PKEY= keyword.

KEY5=(length,loc,DUP/NDUP,CHG/NCHG)

Defines MIRAM index 5. See also description of PKEY= keyword.

MuLTiSeQ=YES

Allow multiple sequential readers of this file.

This specification allows more than one online program to have this MIRAM file in sequential mode at the same time.

TIP/30 does not treat setting this type of file in sequential mode as a use of a serial resource. As a result, programs which have a MULTISEQ=YES file in sequential mode DO NOT need to issue an FCS-ESETL function before requesting terminal input.

Available only for indexed MIRAM files that are accessed using Consolidated Data Management. Requires OS/3 release 11 (or later).

OPEN=

This keyword controls the way TIP/30 opens this file.

Default: OPEN=YES

NO Indicates that TIP/30 must physically CLOSE this file if no online program is using it (the file is NOT to remain open).

Physical file CLOSE operations are relatively time consuming operations.

Library files are always closed when not actually in use; the OPEN= keyword is not relevant for library files.

DEFER TIP/30 is not to open this file during TIP/30 startup processing.

The physical open operation is deferred until the first transaction runs that requests this file.

Once the file is opened, it will remain open (as if OPEN=YES was specified).

This specification may make TIP/30 startup operations faster at the expense of shifting the overhead of opening the file to the first transaction that uses the file.

OPTioNal=

The specification OPTIONAL=YES indicates that this file is optional and may not be defined in the job control stream for TIP/30.

If input is requested from this file (and the file is not present) TIP/30 returns end of file status to the program.

If output is attempted (and the file is not present) the output is ignored, but no indication of an error is returned to the program.

Default: OPTIONAL=NO

PCYLofl=n

The percentage of cylinder overflow for ISAM.

Default: no PCYLOFL value established.

PKEY=n

The number of the key (of a multi-key MIRAM file) that is to be considered the primary key (and the default key) for this file.

If an online program does not specify an explicit index number on calls to the TIP/30 File System (TIPFCS), the PKEY= parameter specifies the index to be used. The TIP/30 catalogue entry for the file may specify KeyREF= to override the PKEY= specification.

Default: PKEY=1

Note: *TIP/30 requires the primary key of an indexed MIRAM file to be defined as NDUP/NCHG (no duplicate keys and no key changes allowed). The primary key of a record (or a portion of it) is the value that TIP/30 enters in the key holding table to enforce record locking conventions.*

POOL=n

Reserves memory for "n" record-size areas where the TIP/30 File Control System maintains a "pool" of the most recently read records for this file. This facility is available for indexed and direct access (or relative record number access) files.

Record pooling does not apply to sequential read operations. Furthermore, only random read requests issued via the primary key for the file are maintained in the pool.

This internal "pool" implements record caching for the file. TIPFCS may be able to satisfy a random read request for a record by retrieving the record from the pool for the file (and thereby avoid a physical read). If a pooled record is updated, the updated record is written to the physical file and the pool is updated.

You can override the number of records to be pooled by using the run-time option FiLePooL=. Even though POOL= is not specified for a file in the generation specifications, record pooling can be enabled through the run-time parameter. See description of the run-time option **FiLePooL=**.

Default: no record pooling for this file.

Note: *Specifying POOL= in the generation parameters forces the file (and the associated record pool) to be "resident" (as if RESIDENT=YES was specified).*

PRINTOV=

For files of type "print", this keyword specifies how form overflow status is handled.

REPORT The default value; specifies that when forms overflow is detected by Data Management, the program is to receive PIB-OVERFLOW status.

SKIP Specifies that when forms overflow is detected by Data Management, an automatic skip to top of form is to be requested *and* the calling programming will receive no error indication.

ReCForM=

The record format.

Choices: FIXBLK, VARBLK, FIXUNB, VARUNB.

Default: RECFORM=FIXBLK

RECsiZe=n

The record size (**NOT including** the RCB byte if the file is a MIRAM file created with RCB).

This keyword is required.

RESident=

YES indicates that the control blocks and I/O areas for this file must permanently reside in memory rather than in a particular file buffer.

Specify this keyword for files that are accessed very frequently (assuming memory is available for this purpose).

Default: file is not resident. Keep in mind that the file may be forced to be resident for other reasons, for example, use of record pooling.

This keyword and BUFFER= are mutually exclusive.

RCB=YES

Include this specification if the file is a MIRAM file that was created with a Record Control Byte (RCB). This keyword applies only to MIRAM files.

If DELETE=RCB is specified for this file, default is RCB=YES otherwise default is RCB=NO.

Note: *This specification is needed only to allow the generation parameter processor to verify the BLKSIZE for this file.*

This keyword has nothing to do with the method that TIP/30 uses to delete records from this file (if requested to do so by an online program).

The DELETE= keyword described earlier controls the method TIP/30 uses to delete records.

USEFiLe=

The keyword parameters that apply to Data Management characteristics of this file are exactly the same as a previously defined buffered file.

All parameters must be repeated and must be exactly the same as the parameters for the other FILE (with the exception of parameters that are not related to OS/3 Data Management: BEFORE=, AFTER=, JOURNAL= etc).

This keyword may be used to circumvent the restriction of 99 buffered files since this file and its "clones" will share a single set of Data Management Control blocks.

Default: the file will have its own control blocks.

Note: *The file specified in the USEFiLe= keyword may not be resident file.*

WARNING

A file that is defined with USEFiLe= cannot be handled by TIP/30 offline (batch) recovery.

VSEC=n

For MIRAM files, defines the variable sector size (see OS/3 Data Management manuals for additional information).

This value must match the value that was established for the file when the file was created.

Default: VSEC=256

. (period)

Place a period at the end of the last keyword specified for each FILE statement to signal the end of that FILE statement.

4.2.1. File Keyword XREF

This section contains a table of all FILE generation keywords related to each type of file that is supported by TIP/30.

Note: This table is provided only as a quick guide because there is a great potential of typographic errors. Consult the description of the FILE statement keywords to be certain that the effect of each keyword is fully understood.

The table indicates whether a particular keyword is either: required (R), optional (O) or undefined (-) for that type of file.

Table 4-13. FILE Keyword Cross Reference

	Indexed			Direct			Sequential		Misc.	
Keyword	MI	IS	IR	MI	DA	MEM	MI	SQ	LIB	PRINT
ACCess=	O	O	O	O	O	-	O	O	-	-
AFTER=	O	O	O	O	O	-	O	O	-	-
AUTOIO=	O	-	O	O	-	-	O	-	-	-
BEFoRe=	O	O	O	O	O	-	O	O	-	-
BLKsiZe=	R	R	R	R	R	-	R	R	-	-
BUFfer=	O	O	O	O	O	-	O	O	O	O
CLOSE=	O	O	O	O	O	O	O	O	O	O
COMSToRe=	O	O	-	-	-	-	-	-	-	-
DeLeTe=	O	O	O	O	O	O	O	-	-	-
FILesiZe=	-	-	-	-	-	R	-	-	-	-
HoLD=	O	O	O	O	O	O	-	-	-	-
INDsiZe=	O	O	O	-	-	-	-	-	-	-
IO=	O	O	O	O	O	O	R	R	-	-
JouRNaL=	O	O	O	O	O	-	O	O	-	-
KeyHoLD=	O	O	O	-	-	-	-	-	-	-
KeYLeN=	O	R	R	-	-	-	-	-	-	-
KeYLoC=	O	R	R	-	-	-	-	-	-	-
KEY1=	O	-	O	-	-	-	-	-	-	-
KEY2=	O	-	-	-	-	-	-	-	-	-
KEY3=	O	-	-	-	-	-	-	-	-	-
KEY4=	O	-	-	-	-	-	-	-	-	-
KEY5=	O	-	-	-	-	-	-	-	-	-

continued ...

Keyword	Indexed			Direct			Sequential		Misc.	
	MI	IS	IR	MI	DA	MEM	MI	SQ	LIB	PRINT
MuLTiSeQ=	O	-	-	-	-	-	-	-	-	-
OPEN=	O	O	O	O	O	O	O	O	-	-
OPTioNal=	O	O	O	O	O	-	O	O	-	-
PCYLofl=	-	O	-	-	-	-	-	-	-	-
PKEY=	O	-	-	-	-	-	-	-	-	-
POOL=	O	O	O	O	O	-	-	-	-	-
PRINTOV=	-	-	-	-	-	-	-	-	-	O
RCB=	O	-	O	O	-	-	O	-	-	-
ReCForM=	O	O	O	O	O	-	O	O	-	-
RECsiZe=	R	R	R	R	R	R	R	R	O	-
RESident=	O	O	O	O	O	O	O	O	-	-
USEFiLe=	O	O	O	O	O	-	O	O	-	-
VSEC=	O	-	O	O	-	-	O	-	-	-

4.3. CLUSTER Definition

The TIPGEN control statement keywords **TermtYPe=** and **termSiZe=** are specified to define the *default* characteristics of terminals in the network.

The CLUSTER control statement enables the user to define *logical* clusters or sets of terminals that have characteristics that differ from the default terminal.

The CLUSTER statement is also used to define terminal bypass printers and some of the characteristics of the printers.

Information stated in these CLUSTER statements is used by TIP/30 to modify internal tables so that the output produced for the different terminal types will be of the correct format and size.

All terminal names that are referenced in CLUSTER statements (master, slaves or bypass) must refer to real TERM names as specified in the ICAM generation used by TIP/30.

Note: *These cluster definitions represent logical groups of terminals with similar characteristics. The clustered terminals need* **NOT** *have any physical interconnections.*

Syntax:

```
CLUSTER  name
         keyword=value  keyword=value
         keyword=value  keyword=value.
```

Table 4-14. CLUSTER Statement

CLUSTER Statement	
Keyword	**Description**
name	Required positional parameter naming master terminal.
BICS=	OFIS Link option: Basic International Character Set.
BYpass=	Name of "bypass" terminal for this cluster.
LFFF=	TIPPRINT precede form feeds (FF) with linefeed.
LoGoN=	TIP/30 LOGON required at this terminal?
NCS=	OFIS Link National Character Set designation.
PRintLF=	Printer needs LF at end of text?
PRintLPP=	Default lines per page for TIPPRINT.
ReaDYmsg=	Send ready message to these terminals.
SLaVes=(,,,)	Up to 8 slave (similar) terminal names.
SP=	Set destructive space bar (U20/U30/U40).
TCP=	Alternative TCP program for cluster.
termSiZe=(r,c)	Size (rows,cols) for terminals in cluster.
TermtYPe=	Type of terminals in cluster.
UNSoL=	Allow receipt of unsolicited messages?
XMIT=	Default control page XMIT value.
. (period)	End of CLUSTER statement.

Where:

name Required positional parameter that specifies the ICAM terminal name of the terminal which is considered the "master" terminal for the cluster.

Programs executing on a terminal in this cluster may refer to this terminal by using the pseudo terminal name *MST.

The designation of a "master" terminal has no mystical significance other than the ability to be referred to by the generic name *MST.

BICS=

OFIS Link option to indicate whether or not terminals in this cluster use BICS. Refer to relevant OFIS Link documentation (OFIS LINK System Administrator Guide).

BYpass=

The name of the bypass terminal for this cluster.

Programs executing on terminals in this cluster may refer to this terminal by using the pseudo terminal name *BYP.

The "bypass" terminal is often a designated terminal that is used for printing purposes. Any terminal in the network may be specified as a bypass terminal for any other terminal.

LFFF=

LFFF=YES indicates that TIPPRINT (when printing to an auxiliary device on a terminal in this cluster) will precede each form feed character (FF) with a linefeed character (LF).

Default: NO.

NO indicates that TIPPRINT will NOT insert a line feed character in front of each form feed character (some printers do NOT automatically supply a LF character when they receive a form feed character).

LoGoN=

NO	Indicates that the terminals in this cluster do not have to LOGON TIP/30.
YES	Indicates that the terminals in this cluster are required and expected to LOGON TIP/30.

Default: LOGON= specification in TIPGEN statement.

Terminals that do not LOGON TIP/30 execute programs as an unknown user (userid is "TP" — transaction program) with membership only in the universal group TIPY and a security level of 32 (or whatever value is specified by the SECUR= run-time keyword).

NCS=

OFIS Link National Character Set designation. See OFIS Link System Administrator guide for information.

PRintLF=

YES or NO indicating whether auxiliary printers used by terminals in this cluster require a line feed (LF) character at the end of each physical message (screenfull of data) transmitted to the auxiliary device.

This entry overrides (for this CLUSTER) the value specified in the corresponding TIPGEN statement.

PRintLPP=n

Specifies the default number of lines per page (for TIPPRINT) when printing to an auxiliary device on a terminal in this cluster.

This entry overrides (for this CLUSTER) the value specified in the corresponding TIPGEN statement.

ReaDYmsg=

YES or NO specifying whether or not the terminals in this cluster are to be sent the TIP/30 ready message.

Default: READYMSG= specification in TIPGEN statement.

SLaVes=(,,,)

A list of up to 8 other terminals in this cluster.

These "slave" terminals are not necessarily physically related to the master; they simply have the same characteristics and are grouped together for convenience.

Default: no slaves.

SP=

For U20/U30/U40 style terminals, setting SP=DS causes TIP/30 to alter the control page to define the space bar as destructive. Specifications other than "DS" (destructive) are not supported by this keyword.

Default: (if omitted) — no alteration of the space bar setting.

TCP=

Name of a non-standard TCP program for this cluster of terminals.

This parameter is used to override the standard TIP/30 command processor and should be specified only on the advice of customer support personnel.

termSiZe=(r,c)

The screen size of terminals in this cluster. The "r" parameter specifies the number of rows; "c" specifies the number of columns.

Default: termSiZe= specification in TIPGEN statement.

TermtYPe=

The terminal type of all terminals in this cluster.

Choices are the same as those listed for the keyword TermtYPe= in the TIPGEN statement previously described.

Default: whatever was specified for TermtYPe= in TIPGEN.

UNSoL=

Establishes for all terminals in the CLUSTER whether or not unsolicited messages may be sent to the terminal.

YES	Terminals are able to receive unsolicited messages.
NO	Terminals are not to receive unsolicited messages.

Default: YES

XMIT=

The control page XMIT value to be set at LOGON time for terminals in this cluster. Choices are the same as those described in the XMIT= keyword in the TIPGEN statement.

Default is the value specified for XMIT= in the TIPGEN statement.

. (period)

Place a period at the end of the last keyword specified for each CLUSTER statement to signal the end of that CLUSTER statement.

Example:

```
CLUSTER ARC1 SLAVES=(ARC2) TYPE=SPC
             BYPASS=ARC3 UNSOL=NO.
```

A CLUSTER statement is often used to define Personal Computers that are operating as UNISCOPE terminals using PEP/STEP software and hardware.

A common technique is to define the PC to be three (or more!) terminals from ICAM's point of view (three or more TERM statements in the ICAM generation parameters — each terminal with a different polling address). Often one display is used as a "bypass" terminal for unattended printing.

Example:

```
CLUSTER PC01 SLAVES=(PC02)
             TYPE=U20 BYPASS=PC03.
```

This example defines two ICAM terminal names as a TIP/30 logical terminal cluster. The BYPASS terminal is stated to be a third terminal: PC03.

TIP/30 transactions that are running on the master or a slave terminal (PC01 or PC02) can route screen output or TIPPRINT output to the generic terminal name ***BYP** to have the information sent to PC03:

```
TIP?▶PRINT(*BYP) JCS/TIP30,S,AUX1
```

This command line executes the utility transaction PRINT on terminal *BYP (TIP/30 resolves this reference by checking the invoking terminal's CLUSTER membership). The PRINT program (which is started on the bypass terminal) routes its output to AUX1 — the printer attached to the bypass terminal.

4.4. Keyword Xref

This section presents a matrix showing all of the keywords that may be specified in the TIPGEN parameters, CLUSTER parameters, and the TIP/30 job control imbedded data set (see description in "TIP/30 Job Control Options" on page 6-1).

The ✓ symbol indicates that the keyword may be specified for the indicated statement.

Table 4-15. Keyword XREF

Keyword	TIPGEN	CLUSTER	JCL
AFT=	✓	-	✓
BaCK=	✓	-	-
BackPRI=	✓	-	✓
Banner1=	-	-	✓
Banner2=	-	-	✓
BICS=	-	✓	-
BYpass=	-	✓	-
B4=	✓	-	-
CATPooL=	✓	-	✓
CICS=	-	-	✓
CLose=	-	-	✓
COMM=	-	-	✓
CONTINUE=	-	-	✓
CURrency=	✓	-	✓
DBMS=	✓	-	✓
DEBUG=	-	-	✓
DECIMAL=	✓	-	✓
DEFOPEN=	-	-	✓
DMname=	-	-	✓
DMSAWT=	✓	-	✓
DMSCAT=	-	-	✓
DMSRWT=	✓	-	✓
EDiTstmp=	✓	-	✓

continued ...

Keyword	TIPGEN	CLUSTER	JCL
ESCape=	✓	-	✓
FaSTLoaD=	✓	-	-
FCSxtent=	✓	-	✓
FiLeBufs=	✓	-	-
FiLePooL=	-	-	✓
FileTab=	✓	-	✓
FREEmem=	✓	-	✓
GDA=	✓	-	✓
ICAM=	-	-	✓
IMS=	✓	-	-
IMSDT=	✓	-	✓
IMSemul=	-	-	✓
IMSROW=	✓	-	✓
IMStranL=	✓	-	✓
IMSUNSDT=	✓	-	✓
JOB=	✓	-	-
JouRNaL=	✓	-	-
KeYTaBLe=	✓	-	✓
LANGuage=	✓	-	✓
LFFF=	-	✓	-
LIBLKSZ=	-	-	✓
LiNEreq=	-	-	✓
LIST=	✓	-	-
LOCAP=	✓	-	✓
LOG=	✓	-	-
LoGoN=	✓	✓	-
MaXCaLLs=	✓	-	✓
MaXPRoG=	✓	-	✓

continued ...

Keyword	TIPGEN	CLUSTER	JCL
MAXTiMe=	✓	-	✓
MCSPooL=	✓	-	✓
McsTab=	-	-	✓
MSGPooL=	-	-	✓
NCS=	✓	✓	✓
NETwork=	✓	-	✓
NumGRPS=	✓	-	✓
OPen=	-	-	✓
OPRQuesz=	-	-	✓
PCXFER=	-	-	✓
PRinT=	-	-	✓
PRintLF=	✓	✓	✓
PRintLPP=	✓	✓	✓
PRintTOF=	✓	-	✓
PRintTTL=	✓	-	✓
PRintUC=	✓	-	✓
PRIority=	✓	-	✓
ProgTab=	-	-	✓
PRoMPT=	-	-	✓
PRSTEN=	✓	-	✓
ReaDYmsg=	✓	✓	✓
RESident=	-	-	✓
RESMEM=	✓	-	✓
RESMOD=	-	-	✓
RESOVLY=	-	-	✓
SCHDname=	-	-	✓
SchdPRI=	✓	-	✓
SECuR=	-	-	✓

continued ...

Keyword	TIPGEN	CLUSTER	JCL
SFSPooL=	✓	-	✓
shutDowN=	✓	-	✓
SITEid=	✓	-	✓
SLaVes=	-	✓	-
SP=	-	✓	-
startUP=	✓	-	✓
STatS=	✓	-	✓
TCP=	-	✓	-
TeRMS=	✓	-	-
termSiZe=	✓	✓	✓
TermtYPe=	✓	✓	✓
TIMeoff=	✓	-	✓
TIMeouT=	✓	-	✓
TIPFILES=	✓	-	-
TIPDUMP=	-	-	✓
UNSoL=	-	✓	-
UpPeR=	✓	-	✓
UserPRI=	✓	-	✓
WARMstrt=	-	-	✓
WORK1=	✓	-	-
WORK2=	✓	-	-
XMIT=	✓	✓	✓
XmitALL=	✓	-	✓
XmitCHan=	✓	-	✓
XmitVAR=	✓	-	✓

4.5. TIP/30 Generation Steps

The following steps are required to generate a TIP/30 system:

1. Create a library element containing the appropriate selection of generation statements as described in the preceding sections
2. Make sure that there is only one TIPGEN statement and that it is the first statement
3. Make certain that the TCAname specified in the TIPGEN statement is a name that does not conflict with an existing TCA.
4. Make sure that there is one FILE statement for each LFD that is to be accessed by online programs through TIP/30.
5. Make sure that all the FILE statements follow the TIPGEN statement.
6. Make sure that (any) CLUSTER statements follow the FILE statement(s).
7. Run the supplied batch job stream TJ$PARAM (described in the following section). Be sure to specify the library and element name of the set of parameters you have prepared.
8. If there are errors in your parameters, correct them and rerun the TJ$PARAM job.
9. If there are no errors in the parameters, the TJ$PARAM job creates (in YJCS or TIP$JCS) the next job (default name is TJ$GEN). This job is automatically scheduled by the TJ$PARAM job (unless you override by using GBL RUN=NO when running the TJ$PARAM job).
10. If the TJ$GEN output is correct you now have a version of TIP/30 that may be used.

Note: *The last step of the constructed job is a LIBS step to delete the constructed job (TJ$GEN) from the library. This action prevents reruns of the TJ$GEN job without first going through the TJ$PARAM process.*

4.6. Generation Parameter Processor

The supplied job stream TJ$PARAM executes the TB$GEN program. Global parameters are provided to specify the element name containing the previously prepared generation parameter stream.

If an LFD named TIP$JCS is defined in the job control stream (this is usually the case), the generated job is written to the library TIP$JCS; otherwise, the job stream is written to YJCS.

The TJ$PARAM job stream accepts some global symbols to determine what processing options are desired:

L= Specify the LFD name of the library where the TIP/30 generation parameters are located. Default is L=SYSGEN.

RUN= Specify RUN=NO if you do NOT wish to have the second step of the generation procedure automatically scheduled.

TCA= Specify the name of the element that contains the TIP/30 generation parameters that are to be processed.

This parameter must be specified.

4.7. Example TIP/30 Generation

The following *example* TIP/30 generation parameter stream illustrates some of the keywords that have been described. This generation stream is intended only as an example — it is not necessarily an endorsement of a particular method of organizing a TIP/30 system.

```
TIPGEN    XX1TCA     AFT=2              BACK=4
                     B4=YES
                     CATPOOL=10         FILEBUFS=5
                     FREEM=5000         JOURNAL=NO
                     MSGPOOL=(5,800)
                     SITEID='ARC'       STATS=(30,300)
                     TERMS=15           READYMSG=YES
                     TERMTYPE=UTS20     TIMEOFF=10
                     XMIT=VAR.
*
*  LIBRARY FILES
*
FILE      SYSGEN,LIB        BUF=4.
*
CLUSTER PC01 SLAVE=(PC02)  BYPASS=PC03  TYPE=SPC.
CLUSTER ARC1 TYPE=U200.
CLUSTER ARC2 TYPE=U20D.
```

4.8. TIP/30 New Release

When a new release of TIP/30 becomes available, customers are sent an order form. Part of the material that is sent with a new release is a set of detailed upgrade instructions.

In some situations, a user may elect to receive a "replacement" TIP/30 release tape (or diskettes). This is sometimes the simplest way to upgrade to a new revision level.

To install such a "replacement" release tape (or diskettes), the following steps should be performed:

- Make sure there is valid backup of TIP and TIP$LOD files
- Run job TJ$LOAD
- Run job TJ$JCS
- Run job TJ$COP
- Regenerate TIP/30.

Note: *This procedure is only valid if you are installing an updated version of the release that you are presently using. The installation of a new release level of TIP/30 is described in the release notice that announces the availability of that release.*

4.9. OS/3 New Release

When a new release of the Unisys Operating system (OS/3) is installed the system programmer must consider the following points that are related to the TIP/30 environment:

- If you are upgrading *major* OS/3 release levels (for example, from release 10 to 11), TIP/30 **must be** generated under the new release before attempting to run TIP/30. Unpredictable results will occur if TIP/30 is not generated under the release of the operating system that is in use.
- If you are simply applying a *minor* upgrade (for example, 11.0.A to 11.0.B), there is usually no need to generate TIP/30 *(although this cannot be absolutely guaranteed).*
- If your site uses the RPG II language to write online TIP/30 programs, you must run the job TJ$UPRPG (see description of supplied batch jobs) and, depending on release level of OS/3, it may be necessary to rerun the installation job TJ$RPGxx (where xx specifies the OS/3 release level: 10, 11, 12, etc.).

Note: *Whenever it is necessary to generate TIP/30, we recommend that a new TCA name be established (remember also that the first 3 characters of the TCA name have to be unique if you use offline batch recovery).*

Section 5
TIP/30 System Files

The TIP/30 system requires a number of files in addition to the site's data files and libraries. These additional files are used by TIP/30 to manage the TIP/30 system.

This section describes each of these files and outlines the use of the file by TIP/30, backup considerations, physical placement information etc.

Since the actual LBL name of a file may vary, this section refers to the files by the actual LFD name. (Some sites run more than one TIP/30 system and therefore use different LBL names for the TIP/30 system files).

Some of the files are always required by TIP/30; several are optional depending on parameters that are specified in the TIP/30 configuration for the site.

Most of these files will be created during the standard installation procedures when TIP/30 is installed. In some situations, optional files may have been left out according to the choices made by the installation personnel.

The individual jobs that are described in the installation procedures may be used to re-create a file or to create a file that was initially not configured.

Files which are referred to as "library" files are standard SAT format OS/3 libraries and may be manipulated by the OS/3 LIBS program (batch librarian).

SAT files which are not identified as "library" files are normally managed only by TIP/30 supplied programs (with exceptions as noted) and must not be indiscriminately manipulated by programs other than those indicated.

Table 5-16. TIP/30 System Files

LFD	Type	Req'd?	Backup	Restore
SYSGEN	SAT library	No	LIBS	
TIP	SAT library	Yes	LIBS	
TIP$BAK	SMIRAM or TAPE	No		
TIP$B4	SAT	No	DMPRST	
TIP$CAT	SAT	Yes	Job TJ$CRBAK	Job TJ$CRRST
TIP$DUMP	SAT	Yes		
TIP$HST	SAT	No	DMPRST	
TIP$JCS	SAT library	Yes	LIBS	
TIP$JRN	SAT	No	DMPRST	
TIP$LOD	SAT library	Yes	LIBS	
TIP$LOG	SAT tape	No		
TIP$MCS	SAT	Yes	Job TJ$CRBAK	Job TJ$CRRST
TIP$MSG	SAT	Yes	DMPRST	
TIP$RNDM	SAT	Yes	Job TJ$CRBAK	Job TJ$CRRST
TIP$SWAP	SAT	Yes		
TIP$TOM	SAT	No	DMPRST	
TIP$TSP	MIRAM (3 keys)	No	DATA	

All of these files *may* be backed up and restored using the standard OS/3 dump/restore utility (DMPRST). Some of the files (as noted) may be backed up or restored **only** with DMPRST.

The job streams named TJ$CRBAK and TJ$CRRST provide backup, restore and reorganization operations for the TIPCAT, TIPMCS, TIP$RNDM files as a group.

Note: *In the following descriptions of the files that are used by the TIP/30 system many LBL names are shown including the string "id". This does not mean that string literally, but the value of the job control symbol &$ID which is presumed to be set to the appropriate value for the individual TIP/30 system (for example: PROD for production system, TEST for test system).*

5.1. SYSGEN — Maintenance Library

LBL name	LFD name	
TIP.id.GEN	SYSGEN	optional

The purpose of the SYSGEN library is to hold the TIP/30 generation parameters. This library is optional, but we recommend that the site create this library and use it. The LBL name illustrated is the default LBL name.

The SYSGEN library is a convenient place to keep all parameters used to generate the operating system, TIP/30, ICAM etc. Other information concerning "system software" can also be stored in this library (for example: patch jobs, copies of JCL procs etc.).

A number of job streams supplied with TIP/30 assume the existence of a library with an LFD name "SYSGEN". If the site chooses to omit this library, some job streams may require modification.

5.2. TIP — Release Library

LBL name	LFD name	
YTIP	TIP	required

The TIP/30 release library is provided on the System 80 Model 7E processor or is supplied on a release tape or release diskettes for other models. If necessary, this library is loaded to disk by the initial installation procedures and is assumed to be accessible by TIP/30 and various batch jobs (compile jobs, link edit jobs, etc).

The release library is *completely* rebuilt when a new TIP/30 release is installed. It is suggested that no user modules be placed in this library because such modules are lost when the release library is reloaded during the installation of a new release!

This file is a standard OS/3 library and is referenced by:

- The TIP/30 HELP utility transaction and the HELP command recognized by many TIP/30 utilities
- Compile and link edit job control streams (the compilers are usually directed to search the TIP library for TIP/30 copy elements, assembler macros etc.; the linkage editor searches the TIP library to resolve calls to TIP/30 subroutines and object modules)
- The TIP/30 generation process uses modules stored in the TIP library during TIP/30 generation.

This library is not referenced extensively by the online system and may be placed on any convenient disk drive.

5.3. TIP$BAK — Backup File

LBL name	LFD name	
TIP.id.BAK	TIP$BAK	optional

The TIP/30 Backup file (TIP$BAK) is an optional file that is used by the supplied job streams TJ$CRBAK/TJ$CRRST (catalogue and random backup and restore) as a backup of the TIP/30 Catalogue (TIP$CAT), the TIP/30 Random File (TIP$RNDM) and the TIP/30 Screen Format File (TIP$MCS). The TIP$BAK file may be a disk or tape file.

The job TJ$BAK will allocate a suitable size TIP$BAK file on disk. (Using the TJ$CRBAK/TJ$CRRST jobs with backup on tape is a function of global parameters for those jobs and is discussed in the documentation of those job streams).

The TIP$BAK file is a MIRAM file that the TIP/30 backup utility uses to hold the contents of the TIPCAT, TIPRNDM and TIP$MCS files.

This file is optional unless use is made of the supplied job streams TJ$CRBAK/TJ$CRRST.

5.4. TIP$B4 — Before Image File

LBL name	LFD name	
TIP.id.B4	TIP$B4	optional

The TIP/30 Before File (TIP$B4) is an optional file that is used by TIP/30 to store "quick before images" of records that are being held for update for the duration of a transaction. This file is used by TIP/30 only if there is one or more generated files specified with HOLD=TR in the TIP/30 generation parameters.

The TIP$B4 file is a SAT file that is managed by TIP/30.

When a transaction program updates a record (for a file that is configured as HOLD=TR), a "before image" of the record is written to the TIP$B4 file.

Similarly, if an online program adds a record to a HOLD=TR file, the record image is written to the TIP$B4 file (it is marked as a "NEW" record).

These "before" images may be used by the TIP/30 system for online roll back (if the transaction aborts for example) or by the TIP/30 system or the batch recovery program for roll back of incomplete transactions after a catastrophic system failure.

The TIP$B4 file may be heavily used at a site which employs HOLD=TR for a large number of files.

The TIP$B4 file is created by using the job stream TJ$B4. The initial disk allocation is not as large as one might expect because the TIP/30 system reuses the space in the TIP$B4 file to minimize the disk space required for this file.

If possible, the TIP$B4 file should be allocated on a disk drive that does not contain a file that is specified as HOLD=TR (this avoids head contention on the drive).

This file is initialized (only) by TIP/30 startup and normally never requires maintenance or backup.

The file may be backed-up/restored by the standard OS/3 system utility DMPRST.

This file is optional unless there are files generated with HOLD=TR.

5.5. TIP$CAT — Catalogue File

LBL name	LFD name	
TIP.id.CAT	TIP$CAT	required

The TIP/30 Catalogue file (TIP$CAT) is used by TIP/30 to maintain critical information concerning the TIP/30 users, online programs and files. The TIP/30 catalogue information controls the manner in which TIP/30 runs online programs and accesses online files.

The TIP$CAT file is a SAT file containing the catalogue records for:

- USER information
- PROGRAM information
- FILE information
- GROUP set information.

There is a TIP/30 batch program (TB$CRB) that is described elsewhere in this section of the manual which is used to reorganize/backup the TIP$CAT file (and the TIP$MCS and TIP$RNDM files).

WARNING

The TIP$CAT file is absolutely crucial to the operation of TIP/30. The utmost care must be taken to ensure that there is always a backup of this file.

Although the information in the TIP/30 catalogue file is referenced often by TIP/30, the actual number of I/O operations for the file is not proportionally high. This is because the TIP/30 system maintains an internal "bit map" indicating which physical records in the file actually contain data. "No find" situations are therefore easily resolved without I/O.

In addition, a TIP/30 generation parameter (CATPOOL=) can be specified to create a "pool" of the most recently used catalogue records in main memory.

To minimize the number of accesses of FILE information from the TIP$CAT file, the FILETAB= TIP/30 generation parameter may be used to specify that FILE information be maintained in main memory.

The TIP$CAT file must be pre-formatted by the TIP/30 batch file format utility (refer to the job TJ$CAT).

The TIP/30 system will only make use of a maximum of 8,191 catalogue records (each record is 256 bytes) — a larger allocation is totally wasted.

5.6. TIP$DUMP — Dump File

LBL name	LFD name	
TIP.id.DUMP	TIP$DUMP	optional

The TIP/30 Dump file (TIP$DUMP) is a required file that is used by TIP/30 in the unlikely event that TIP/30 terminates abnormally. TIP/30 copies the image of memory to this file so that the supplied job TJ$DMP can print or copy the memory dump for offline analysis.

The TIP$DUMP file is a SAT file that is managed by the TIP/30 system.

5.7. TIP$HST — History File

LBL name	LFD name	
TIP.id.HST	TIP$HST	optional

The TIP/30 History File (TIP$HST) is an optional file that is used to accumulate TIP/30 Journal or Log information. The information that is written to the TIP$JRN file (or TIP$LOG) file can be periodically appended to the TIP/30 History File (TIP$HST).

The TIP$HST file is a SAT disk file that is managed by batch programs supplied with TIP/30. The file is not used online by the TIP/30 system.

"Normal" practice is to copy the TIP$JRN (or TIP$LOG) file to the TIP$HST file when TIP/30 shuts down. The TIP$JRN (or TIP$LOG) file is then initialized and ready to receive the journal records for the next session.

Of course, abnormal shutdown situations may require some deviation from this simplistic scheme.

The TIP$HST file may be backed up by the standard OS/3 dump/restore utility (DMPRST).

The following (supplied) job control streams may be used to perform the indicated function related to the TIP$HST file:

- TJ$HST — create and initialize the TIP$HST file
- TJ$JR2HS — copy TIP$JRN to TIP$HST and initialize the TIP$JRN file.

These job streams are documented elsewhere in this section of the manual.

5.8. TIP$JCS — Job Control Library

LBL name	LFD name	
varies	TIP$JCS	optional

The purpose of the TIP$JCS library is to hold the job control streams that are supplied with TIP/30 (along with any other TIP/30 related job control that is created by the site).

The LBL name for this file can be in the "standard" format (TIP.id.JCS) but this has severe drawbacks when the LBL name is needed to execute jobs, since the full LBL name must be provided.

The recommended LBL name is often a short name such as "TJ".

There must be a unique job control library for each unique TIP/30 system on a single processor. This requirement is not onerous since the libraries are relatively small. Experience has shown that having a separate job control library for each TIP/30 system can avoid many pitfalls.

5.9. TIP$JRN — Journal File

LBL name	LFD name	
TIP.id.JRN	TIP$JRN	optional

The TIP/30 Journal File (TIP$JRN) is an optional file that is used by TIP/30 to "journal" information about updated records and the system. The information that is written to the TIP$JRN file depends on various TIP/30 generation parameters.

If a TIP$JRN file is configured, the TIP/30 system will automatically write information to this file concerning user LOGON/LOGOFF, library element reads and writes, and TIP/30 internal statistics at a specified interval.

In addition, "before" and/or "after" images of data file records (for HOLD=TR files) that are modified, deleted or added are written to this file (all according to various TIP/30 generation parameters).

The TIP$JRN file should be allocated on a disk drive that does not contain any application data files that are being journaled — the loss of the drive may mean the loss of both the data files and the only means to reconstruct the information. Ideally, the TIP$JRN file should be allocated on a drive that is the least used disk drive — the TIP$JRN file may be a very heavily used file depending on the TIP/30 system generation options that are selected.

The TIP$JRN file may serve several purposes:

- an audit trail of system activity
- record of images that may be used for (off line, batch) roll back or roll forward (reapplication of updates)
- TIP/30 system statistics collection.

More information about the type and format of records written to the TIP$JRN file can be found in the Journal section of the description of the TIP/30 File Control System (FCS).

"Normal" practice is to copy the TIP$JRN file to a history file (TIP$HST) when TIP/30 shuts down. The TIP$JRN file is then initialized and ready to receive the journal records for the next session. Of course, abnormal shutdown situations may require deviation from this simplistic scheme.

It is also good practice to refresh the TIP$JRN file before installing a new TIP/30 release or patch level and whenever a new TCA (TIP/30 Control Area or generation set) is used — the offline recovery procedures use information contained in the TCA to control recovery — if the file information in the generation parameters is changed, the new TCA may not be appropriate to perform offline recovery using older journal information!

The TIP$JRN file is a SAT file and may only be manipulated by programs supplied with TIP/30 or by the standard OS/3 dump/restore utility DMPRST.

The following (supplied) job control streams may be used to perform the indicated function related to the TIP$JRN file:

- TJ$JRINT — initialize the TIP$JRN file
- TJ$JRN — allocate and initialize the TIP$JRN file
- TJ$JR2HS — copy TIP$JRN file to TIP$HST and initialize the TIP$JRN file.

These job streams are documented elsewhere in this section of the manual.

This file is optional unless JOURNAL=YES was specified in the TIP/30 generation parameters.

5.10. TIP$LOD — Load Library

LBL name	LFD name	
TIP.id.LOD	TIP$LOD	required

The TIP$LOD library is the library containing all online program load modules. This library also contains the resulting load module(s) produced by TIP/30 generations (TCAs). This library is a standard format OS/3 library.

TIP$LOD is initially built by the installation procedures and contains all online program load modules supplied with TIP/30 *and* all user written online program load modules.

Online programs to be used with TIP/30 **must** be linked so that the load module is placed in this library. It is not mandatory to use the OS/3 LIBS "block load" facility for online load modules.

This library must be packed and backed up frequently (especially at sites where program development is a major activity).

5.11. TIP$LOG — Log Tape

LBL name	LFD name	
TIP$LOG	TIP$LOG	optional

The TIP/30 Log Tape (TIP$LOG) may be used as an alternative or a supplement to the TIP/30 Journal File (a disk file). TIP$LOG is used for exactly the same purpose as the TIP$JRN file — the difference is that the information is written to tape.

The TIP$LOG file and the TIP/30 Journal file (TIP$JRN) are mutually exclusive files.

More information about the type and format of records written to the TIP$LOG file can be found in the Journal section of the description of the TIP/30 File Control System (FCS).

"Normal" practice is to copy the TIP$LOG tape to a history file (TIP$HST) when TIP/30 shuts down. A new TIP$LOG tape is then used to receive the journal records for the next session. Of course, abnormal shutdown situations may require some deviation from this simplistic scheme.

This file is optional unless LOG=YES was specified in the TIP/30 generation parameters.

5.12. TIP$MCS — Screen Format File

LBL name	LFD name	
TIP.id.MCS	TIP$MCS	required

The TIP/30 Screen Format File (TIP$MCS) is used by TIP/30 to contain TIP/30 screen formats. The formats are stored in an internal (compressed) format in this file.

The TIP$MCS file is a SAT file with one partition containing records that are 2,560 bytes in length.

There is a TIP/30 batch program (TB$CRB) that is described elsewhere in this section of the manual which is used to reorganize/backup the TIP$MCS file (and the TIP$CAT and TIP$RNDM files).

The TIP$MCS file must be pre-formatted by the TIP/30 batch file format utility (this operation is performed by the TIP/30 installation procedure).

The TIP/30 system will only make use of a maximum of 8,191 screen records (each record is 2,560 bytes) — a larger allocation is totally wasted.

The TIP/30 job control parameter MCSPOOL= defines a pool of memory that is used to keep the most recently used screen formats in memory to eliminate some reads from this file.

5.13. TIP$MSG — TIP/30 Message File

LBL name	LFD name	
TIP.id.MSG	TIP$MSG	required

The TIP/30 canned Message Files (TIP$MSG) is used by TIP/30 to contain many of the messages that are output by TIP/30 or by utility transactions supplied with TIP/30.

The TIP$MSG file is a MIRAM file containing 256 byte records.

The TIP/30 job control parameter FILEPOOL= can be used to establish record pooling for this file to keep the most recently used messages in memory and reduce physical I/O requests for this file.

5.14. TIP$RNDM — Random File

LBL name	LFD name	
TIP.id.RNDM	TIP$RNDM	required

The TIP/30 Random File is a SAT file that is used by TIP/30 to allocate Edit buffers and FCS Dynamic files.

Each FCS dynamic file has an initial allocation of 40 blocks (each block is a fixed size of 512 bytes). The dynamic file may grow (based on the demands made by an online program) in increments of 40 blocks. (In actual fact, 40 is the default initial and secondary block allocation — the actual value used is given by the TIP/30 generation parameter FCSXTENT=).

The TIP$RNDM file is managed internally by TIP/30 and may extend (and frequently does!) depending on the utilization of edit buffers and other dynamic files. The maximum size of the TIP$RNDM file that is supported by TIP/30 is:

(8192 x FCSXTENT) 512-byte blocks

The site administrator should maintain a close watch (via the CAT utility transaction) on the proliferation of edit buffers and dynamic files.

Because the TIP$RNDM file is a SAT file, there is no mechanism available to directly compress the file. Once the file extends, it remains extended (even though much of the space may not actually be in use).

There is a TIP/30 batch program (TB$CRB) — described elsewhere — that can be used to reorganize/backup the TIP$RNDM file (and the TIP$CAT and TIP$MCS files). note> The TIP$RNDM file is directly related to entries in the TIP/30 Catalogue File (TIP$CAT) and must be considered as a logical "extension" of the TIP$CAT file (that is, one file cannot be restored/reorganized independently of the other).

The TIP$RNDM file must be pre-formatted by the TIP/30 batch file format utility.

Since library elements are frequently copied into an Edit buffer the TIP$RNDM file should preferably be placed on a drive that does not contain any of the user's libraries (to reduce head contention).

5.15. TIP$SWAP — Swap File

LBL name	LFD name	
TIP.id.SWAP	TIP$SWAP	required

The TIP/30 Swapping Storage File (TIP$SWAP) is used by TIP/30 to provide high speed physical memory swapping. The file is managed by TIP/30 using SAT Data Management.

This file may be extended (if necessary) by TIP/30 if the space is available on the volume.

The TIP$SWAP file may be used by TIP/30 to:

- temporarily save the memory contents of programs that are "swapped out" (waiting for input for example)
- temporarily save the memory contents of a TIP/30 file buffer if the current buffer occupant (file) is not the desired occupant
- temporarily save the contents of the CDA (in some cases) while TIP/30 is transferring control from one user program to another
- fast loading of certain TIP/30 internal transient functions.

The TIP$SWAP file may be a very heavily used file and is best placed on non-sectored disk drives (if a choice is available); in any case, first consideration should be to place the TIP$SWAP file on a disk drive that is used the least.

The TIP$SWAP file contains NO data of any consequence when TIP/30 is not running. It is not necessary to include this file in backup procedures.

The TIP$SWAP file must be preformatted by the TIP/30 batch file format utility (refer to the job TJ$SWAP).

If possible, the user should allocate the TIP$SWAP file as a contiguous disk extent and, for premium performance, attempt to avoid any alternate track assignments within the TIP$SWAP file space.

5.16. TIP$TOM — Output Message File

LBL name	LFD name	
TIP.id.TOM	TIP$TOM	optional

The TIP$TOM file is an optional file that may be allocated for use by TIP/30 to support the DLMSG (Display Last output Message) transaction that provides compatability with IMS.

The TIP$TOM file is a SAT format file that is managed by TIP/30. This file is optional unless use is made of the DLMSG transaction.

5.17. TIP$TSP — Sample Program File

LBL name	LFD name	
TIP.id.TSP	TIP$TSP	optional

The TIP/30 Sample Program File (TIP$TSP) is an optional file that is used by the TIP/30 sample programs (transactions TSP, TSPUPDT and TSPRNT).

The sample program is supplied with TIP/30 to illustrate how a typical TIP/30 native mode program may be written. This file contains sample data so that the sample program can actually be run and observed.

The TIP$TSP file is a standard indexed MIRAM file with 1 index.

The TIP$TSP file is normally generated into the TIP/30 system by the initial installation procedures.

The file is configured in the TIP/30 generation parameters as follows:

```
FILE TIP$TSP,MIRAM  ACCESS=EXC
                    BLKSIZE=1024
                    HOLD=YES
                    KEY1=(8,0,NDUP,NCHG)
                    KEY2=(25,9,DUP,CHG)
                    KEY3=(10,116,DUP,CHG)
                    RECSIZE=335.
```

Note: *ACCESS=EXC may be used because the file is normally only used online by TIP/30 and is not referenced in batch.*

The TIP$TSP file does not need to be pre-formatted in any way; allocate the file and ensure that it is defined in the TIP/30 generation parameters.

The TIP$TSP file may be backed up using the DATA utility or by using the standard OS/3 dump/restore utility (DMPRST).

The TIP$TSP file may be periodically unloaded/reload (using the OS/3 DATA or MILOAD utility) but the volatility of this file is normally very low and therefore this reorganization is seldom necessary.

This file is optional unless use is made of the TIP/30 sample programs.

Section 6
TIP/30 Job Control Options

There are many options that can be specified when TIP/30 is executed. These options are entered on run control statements that are free format (similar to the generation control statements).

The information specified in these control statements is used to modify internal tables in TIP/30 and override certain parameters that were selected or defined in the TIP/30 system generation.

This provides a certain degree of flexibility in that some generation choices can be changed without having to do a complete TIP/30 generation.

The analysis of the job control parameters validates the parameters carefully. If an error is detected, the statement in error is displayed and the following console message is issued:

```
"CANCEL TIP/30 OR IGNORE ERROR? (C/I)"
```

If the operator responds "I" (ignore) or enters a null reply, the keyword in error is treated as if it had not been specified. If the operator responds "C" (cancel), the remaining statement are checked, but TIP/30 initialization is not started and TIP/30 terminates. After a "C" reply is given, other subsequent errors are logged, but the operator is not prompted for any further responses (since he has already indicated a cancel operation).

WARNING

Errors in the syntax MAY MAKE IT IMPOSSIBLE to run TIP/30. ALWAYS make a backup of the TIP/30 job control stream BEFORE making any "improvements".

The parameters are supplied as an "imbedded data set" (card images bounded by a /$ and /* statement) in the TIP/30 job control. The card images are not scanned beyond column 72.

Example:

```
// EXEC TB$TIP,TIP$LOD,1
/$
 .
 .   imbedded data set containing
 .   TIP/30 run-time options
 .
/*
```

The following run-time control statements may be specified as parameters to TIP/30. Keywords are illustrated in the standard format (upper case letters are required letters; lower case letters may be omitted).

Some of the keywords allow the specification of a list of items (or sub-parameters). In such cases, the keyword may be repeated as many times as needed to specify all desired parameters, for example:

```
RESIDENT=TT$TCP,TT$FCS
RESIDENT=SUBPROG1,SUBPROG2,SUBPROG3
RESIDENT=(PAY020,PAY035)
```

Table 6-17. TIP/30 Run-time Parameters

Run-time Parameters	
Specification	**Description**
tcaname	Name of TIP/30 generation parameters to use.
AFT=	Default size of Active File Table per process.
BackPRI=	Background transaction scheduling priority.
Banner1=	Text for line one of TIP/30 ready message.
Banner2=	Text for line two of TIP/30 ready message.
CATPooL=	Size of CATalogue pool.
CICS=	Load module for CICS program interface.
CLose=	Files (LFDs) not available until OPENed.
COMM=	TIP/30 — ICAM interface load module.
CONTINUE=	Whether or not TIP/30 continues running after detecting memory corruption.
CURrency=	Currency symbol.

continued ...

Run-time Parameters	
Specification	**Description**
DBMS=	DMS configuration.
DEBUG=	System debug option.
DECIMAL=	Define decimal point character.
DEFOPEN=	Override deferred open files.
DMname=	TIP/30 data management interface load module.
DMSAWT=	DMS area wait time.
DMSCAT=	DMS DMCL names controlled by catalogue.
DMSRWT=	DMS record wait time.
EDiTstmp=	Default update stamping for FSE editor.
ESCape=	System escape character.
FCSxtent=	FCS Dynamic file extent size.
FiLePooL=	Override record pool size for one or more files.
FileTab=	Generate in-memory file table.
FREEmem=	Size of TIP/30 Free Memory Pool.
GDA=	Size of Global Data Area.
ICAM=	Restart ICAM queues?
IMSDT=	Delay time for IMS delayed internal succession to self.
IMSemul=	IMS emulator load module.
IMSROW=	Terminal row number for IMS messages.
IMStranL=	IMS transaction code length.
IMSUNSDT=	IMS emulator send messages to down terminals?
KeYTaBLe=	Size of TIP/30 key holding table.
LANGuage=	System default language code.
LIBLKSZ=	Block size for TIP/30 access to library files.
LiNEreq=	List of network lines to open at startup.
LOCAP=	Global ICAM Local Application to connect.
MaXCaLLs=	Specify max calls before program is aborted.
MaXPRoG=	Minimum required paged memory.
MAXTiMe=	Program timeout.

continued ...

Run-time Parameters	
Specification	**Description**
McSgPooL=	Override size of TIP$MCS record pool.
McsTab=	Generate in-memory screen table.
NETwork=	ICAM CCA network name and password.
NumGRPS=	Override number of elective user groups.
OPen=	Specify files (LFDs) to be OPENed at startup.
OPRQuesz=	Size of operator unsolicited message stack.
PCXFER=	PC file transfer load module.
PRinT=	TIPPRINT interface load module.
PRintLF=	Override TIPPRINT line feed option.
PRintLPP=	Override TIPPRINT default lines per page.
PRintTOF=	Override TIPPRINT top of form option.
PRintTTL=	Override TIPPRINT title page option.
PRintUC=	Override TIPPRINT upper case translation.
PRIority=	Number of transaction scheduling priority levels.
ProgTab=	Generate in-memory program table.
PRoMPT=	Define system prompt string.
PRSTEN=	Journal program start & end information.
ReaDYmsg=	Override READYMSG= gen value.
RESident=	Load modules to make memory resident.
RESMEM=	Define resident memory pool for IMS.
RESMOD=	Specify resident TIP internal modules.
RESOVLY=	Specify resident TIP overlays.
SCHDname=	Specify TIP/30 scheduler load module.
SchdPRI=	TIP/30 scheduler execution priority level.
SECuR=	Security level for LOGON=NO terminals.
SFSPooL=	SFS format pool size.
shutDowN=	Background transaction run at TIP/30 EOJ.
SITEid=	Site name.
startUP=	Background transaction run at TIP/30 startup.
STatS=	Define statistics intervals.

continued ...

Run-time Parameters	
Specification	**Description**
termSiZe=	Terminal screen size: (rows,columns).
TermtYPe=	General terminal type.
TIMeoff=	Automatic logoff timeout.
TIMeouT=	Maximum external succession time.
TIPDUMP=	Define external processing if TIP/30 dumps.
UpPeR=	Override TIP/30 uppercase translate table.
UserPRI=	Foreground priority level.
WARMstrt=	Type of warm start (roll back) desired.
XMIT=	UTS400 control page XMIT option.
XmitALL=	UTS400 Fn key to XMIT ALL
XmitCHan=	UTS400 Fn key to XMIT CHAN
XmitVAR=	UTS400 Fn key to XMIT VAR

Where:

tcaname

The tcaname is the only required positional parameter and must appear before any keyword parameters. The name specified is the name of the TIP/30 Control Area (TCA) that represents the particular set of TIP/30 generation parameters to use for this execution run.

This **must be** the first parameter in the imbedded data set (since it is a positional parameter NOT a keyword parameter).

AFT=

Overrides corresponding TIPGEN parameter.

BackPRI=

Overrides corresponding TIPGEN parameter.

Banner1='...'

The text of the first (of two) lines of the TIP/30 ready message that may be sent to terminals when TIP/30 begins execution.

Up to 48 characters may be supplied (enclosed in single quotes).

Banner2='...'

The text of the second (of two) lines of the TIP/30 ready message that is sent to terminals when TIP/30 begins execution.

Up to 48 characters may be supplied (enclosed in single quotes).

CATPooL=

Overrides corresponding TIPGEN parameter.

CICS=

Specifies the load module name of the CICS program interface. IF CICS=YES is specified, the default CICS interface handler is loaded. If this keyword is not specified, the CICS interface is not loaded.

CLose=

A list of LFD names (or prefixes) of online files that are NOT to be opened at startup and are to remain CLOSED and UNAVAILABLE for online use until the operator issues the console OPEN command (or a user runs the FOPEN online utility) for the file.

Example: CLOSE=(TIP$TSP,*PAY)

A maximum of 100 files may be specified by using this keyword.

COMM=

Specifies the load module name of the TIP/30 communications handler. If this keyword is not specified, TIP/30 loads the appropriate communications handler depending on the generations options selected.

This keyword is normally specified only under the direction of customer support personnel.

CONTINUE=

Specifies whether or not TIP/30 is to continue execution if TIP/30 detects that low order memory is corrupted. TIP/30 periodically examines various locations near address 0 (zero) of the TIP/30 job region. These areas are supposed to be binary zeroes. If TIP/30 detects any other value, the assumption is made that some transaction program (or TIP/30 itself) is erroneously modifying memory.

If CONTINUE=YES is specified in the run-time parameters, TIP/30 will take an online dump and will reset the area to binary zeroes and resume processing.

If CONTINUE=YES was not specified, TIP/30 will immediately terminate with error code 9F4.

CURrency=

Overrides corresponding TIPGEN parameter.

DBMS=

Overrides corresponding TIPGEN parameter.

DEBUG=

System debug option. This option controls the use of hardware storage protection by TIP/30. The option controls the *default* situation only; individual transaction programs may be defined in the TIP/30 Catalogue to use or not use storage protection. (See description of same keyword for PROGram entries in the TIP/30 Catalogue Manager utility transaction — CAT).

When DEBUG is set, TIP/30 isolates the transaction program that is running from the rest of the TIP/30 region by establishing a different storage protection key for the program's work areas (PIB, CDA, MCS, WORK, VOL).

If the program attempts to alter memory outside it's allocated area, the hardware will suppress the offending instruction and signal a "Protection Exception" and the transaction program will be be aborted before causing any havoc in the neighbourhood.

There is overhead associated with the use of this option (approximately a millisecond per CALL (on a SYS/80 Model 20) to TIP/30 — to set and reset storage keys). We recommend that *well tested and debugged* systems be specified in the TIP/30 Catalogue with DEBUG=NO to minimize overhead.

New and untried programs should be defined in the TIP/30 Catalogue with the DEBUG=YES specification.

Specifying DEBUG= in a program's Catalogue entry overrides the system specification for that particular program.

Default: DEBUG=YES (run TIP/30 system in debug mode; all transaction programs will run with hardware storage protection unless overriden by the TIP/30 Catalogue entry for the program).

Specify DEBUG=NO to override the default (do not run TIP/30 system in debug mode).

DECIMAL=

Overrides corresponding TIPGEN parameter.

DEFOPEN=

Specify list of file names that are to be treated as if OPEN=DEFER was specified in the generation parameter for the file. This keyword permits the dynamic specification of OPEN=DEFER in the TIP/30 job control stream.

Up to 100 file names (or file name prefixes may be specified).

This keyword overrides the OPEN= specification for the named files.

Example: DEFOPEN=(*PAY,ARMAST)

DMname=

Name of the load module that is to be used by TIP/30 to interface with OS/3 data management.

This keyword should only be specified under the direction of customer support personnel.

DMSAWT=

Overrides corresponding TIPGEN parameter.

DMSCAT=

Specify whether or not DMS database DMCL names must be resolved by referring to the TIP/30 catalogue entries.

Default is NO.

If DMSCAT=YES is specified, all DMS DMCL names must appear in the TIP/30 catalogue (see description of the DMCL= keyword for the FILE command in the CAT utility transaction).

If DMSCAT=NO is set (or is assumed by default), TIP/30 first attempts to locate a definition of the DMCL name in the TIP/30 catalogue. If no catalogue entry is found, the raw DMCL name is used asis.

DMSRWT=

Overrides corresponding TIPGEN parameter.

EDiTstmp=

Overrides corresponding TIPGEN parameter.

ESCape=

Overrides corresponding TIPGEN parameter.

FCSxtent=

Overrides corresponding TIPGEN parameter.

FiLePooL=

Override record pool size for one or more files.

This keyword is specified as a number of pairs of sub-parameters; the first subparameter is the LFD name, the second subparameter is the desired size of the record pool for that LFD.

You may use this keyword to either override record pooling for files that are specified in the generation parameters with the POOL= keyword or to establish record pooling for a file that was not generated with a POOL= value. In the latter case, the file is not forced to be a resident file.

Example: FLPL=(PAYMAST,40,ARMAST,10)

A maximum of 20 pairs of entries may be specified via this keyword.

The number of screen formats in the TIP/30 MCS pool can be overridden by specifying an LFD name of "TIP$MCS". Similarly, the number of records in the TIP/30 Catalogue pool can be overridden by specifying an LFD name of "TIP$CAT".

FileTab=

Overrides corresponding TIPGEN parameter.

FREEmem=

Overrides TIPGEN parameter FREEm=.

GDA=

Overrides corresponding TIPGEN parameter.

ICAM=RESTART

Specifying ICAM=RESTART will cause TIP/30 to specify warm recovery of ICAM disk queues when TIP/30 opens the ICAM network (at startup time).

This specification recovers all outstanding messages in the ICAM disk queues (typically these might be unsolicited messages that were not retrieved by users).

In order for this specification to work, the ICAM disk queues must not be specified with the INIT option on the corresponding LFD statements in Job Control.

Default: ICAM disk queues are not recovered.

IMSDT=

Overrides corresponding TIPGEN parameter.

IMSemul=

Specify the load module name of the desired IMS emulator.

YES	Use the standard IMS emulator (TT$IMS) — no support of Unisys Screen Format Services (SFS).
TT$IS8	Use the IMS emulator that runs on OS/3 release 8 and later releases and supports Screen Format Services (SFS).

IMSROW=

Overrides corresponding TIPGEN parameter.

IMStranL=

Overrides corresponding TIPGEN parameter.

IMSUNSDT=

Overrides corresponding TIPGEN parameter.

KeYTaBLe=

Overrides corresponding TIPGEN parameter.

LANGuage=

Specify language code for TIP/30 messages that are retrieved from the TIP$MSG file.

The default is LANGUAGE=A (American English).

LIBLKSZ=

Specifies the internal blocksize that TIP/30 is to use when accessing OS/3 (SAT) libraries.

The default blocksize is 1280 (5*256). The maximum allowed blocksize is 24576 (96*256). The minimum specification is 256 (no buffering).

LiNEreq=

Specify the type of network open that TIP/30 is to use when connecting with ICAM.

This keyword is valid only with a dedicated ICAM and is ignored if a Global ICAM is being used.

If this keyword is omitted (and a dedicated ICAM is being used), TIP/30 will automatically issue a line request for each line in the network (the user doesn't need to know or specify each line).

LNE=YES indicates that TIP/30 is to open the entire network; if any line fails to open correctly, the network will fail to open (and TIP/30 will not run!).

LNE=(lne1,lne2,...) indicates that only the lines specified are to be opened when the network is opened.

A maximum of 50 lines may be opened via this keyword.

LOCAP=

Overrides corresponding TIPGEN parameter.

MaXCaLLs=

Overrides corresponding TIPGEN parameter.

MaXPRoG=

Overrides corresponding TIPGEN parameter.

MAXTiMe=

Overrides corresponding TIPGEN parameter.

McSgPooL=

Override number of entries in the TIP/30 screen format pool as defined by the corresponding TIPGEN parameter — the size of the pool entries cannot be adjusted by this run-time parameter, only the number of pool entries.

McsTab=

YES indicates that TIP/30 is to maintain an internal memory table of all MCS screen formats. Each table entry occupies 16 bytes (group/name).

Default: NO

Sites with surplus memory may consider using this parameter (along with MCSPOOL=) to improve TIP/30 screen format access times.

NETwork=

Overrides corresponding TIPGEN parameter.

NumGRPS=

Overrides corresponding TIPGEN parameter.

OPen=

A list of LFD names (or prefixes) to be OPENed at startup.

A maximum of 100 files may be opened via this keyword.

This keyword may be used to override the specification of CLOSE=YES in the generation parameters for a FILE.

Example: OPEN=*PAY,*AR,SECFILE

OPRQuesz=

Size of (operator) unsolicited message queue. Specified as a number of commands that may be queued.

Default: OPRQUESZ=10 (up to 10 unsolicited console commands may be queued).

PCXFER=

Load module for file transfer between TIP/30 and Personal Computers (PCs).

Default: if this keyword is not present, the PC file transfer interface is NOT available for user-written programs. Utility transactions supplied with TIP/30 use a built in version of this interface and do not require this keyword specification.

PCXFER=YES directs TIP/30 to include the "standard" PC file transfer interface routine (load module TR$PCX).

Specification of this keyword is only necessary if user-written programs intend to call the PC file transfer subroutines TIPP2H or TIPH2P.

PRinT=

Load module for TIPPRINT support.

Default: the standard TIPPRINT resident interface load module (TR$PRT) is automatically included.

Specification of this keyword is normally not necessary unless directed by customer support personnel.

PRintLF=

Overrides corresponding TIPGEN parameter.

PRintLPP=

Overrides corresponding TIPGEN parameter.

PRintTOF=

Overrides corresponding TIPGEN parameter.

PRintTTL=

Overrides corresponding TIPGEN parameter.

PRintUC=

Overrides corresponding TIPGEN parameter.

PRIority=

Overrides corresponding TIPGEN parameter.

ProgTab=

Specifying ProgTab=YES causes TIP/30 to maintain a table (in memory) of valid catalogue PROG entries. A table entry is made for all valid transaction names that *are not* in the group TIPY. Table entries are approximately 16 bytes (group name and transaction name).

This table is used to quickly verify that a transaction name is valid (before proceeding with a catalogue search).

Default: NO.

PRoMPT=

Define the character string to be used as the TIP/30 command line prompt.

DEFAULT Set the system prompt string to "TIP?▸" Naturally, this is the default value!

LOCAP Set the system prompt string to the 4 character ICAM LOCAP name followed by a question mark and an SOE character (▸). The LOCAP name will be the same LOCAP that TIP/30 has connected.

NETWORK

Set the system prompt string to the 4 character ICAM NETWORK name followed by a question mark and an SOE character (▸).

SOE Set the system prompt string to a question mark followed by an SOE character (▸).

'...' Set the system prompt string to the character string specified (maximum of eight characters) followed by an SOE character (▸).

If the string contains an imbedded space or period the string must be enclosed in single quotes.

Example: PROMPT='A.R.C' results in A.R.C▸

PRSTEN=

Overrides corresponding TIPGEN parameter.

ReaDYmsg=

Overrides corresponding TIPGEN parameter.

RESident=

Specifies the names of online load modules that are to be made permanently memory-resident at TIP/30 initialization.

A maximum of 90 load modules may be made resident — this keyword may be repeated to allow up to the maximum number of load module names.

The load module of resident programs remains permanently in memory. (The RELOAD utility transaction *will* allow you to force the use of a new copy of the load module of a resident program but the previously resident copy is marked "not usable" and the space that it occupied is wasted until TIP/30 shutdown).

IMS or TIP/30 sub-programs must be made resident. TIP/30 initialization routines scan the TIP/30 catalogue to build various internal tables. Any load modules that are identified in a TIP/30 catalogue entry with the specification SUBPROG=YES are automatically made resident and need not be explicitly identified using the RESIDENT= run-time specification.

TIP/30 attempts to keep often used programs in memory. It therefore makes sense to make a program permanently resident ONLY IF the program is used **almost all** the time or the program is a SUBPROG (which must, by definition, be resident).

Example: RESIDENT=TT$TCP,TT$FSE

RESMEM=

Overrides corresponding TIPGEN parameter.

RESMOD=

Specify alternate TIP/30 internal routines to use.

A maximum of 8 modules may be specified via this keyword.

RESOVLY=

Specify resident TIP overlays.

A maximum of 12 modules may be specified via this keyword.

$LIB Make the library open and close routine resident (this routine is approximately X'E00' bytes in size).

$DYN Make the open and close routine for non-library files resident (this routine is approximately X'E00' bytes in size).

For sites with spare memory, making either or both of these routines resident may improve performance of these functions.

SCHDname=

Specify the load module name of the TIP/30 scheduler. This keyword should only be specified under the direction of customer support personnel.

SchdPRI=

Overrides corresponding TIPGEN parameter.

SECuR=

Specify the default security level to assign to users who do not logon the TIP/30 system. Terminals may be designated as LOGON=NO terminals (either in the TIPGEN parameters or individually in CLUSTER statements).

If a terminal does not require a TIP/30 logon, TIP/30 defaults to a fictional user id of "TP" and a default security level.

Since the user id is not real, transactions which are to execute at such terminals must be defined in the TIP/30 Catalogue in the group TIPY.

The SECuR= keyword may be specified in the job control parameters to set the desired security level for such terminals.

Default: SECuR=32

SFSPooL=

Overrides corresponding TIPGEN parameter.

shutDowN=

Overrides corresponding TIPGEN parameter.

SITEid=

Overrides corresponding TIPGEN parameter.

startUP=

Overrides corresponding TIPGEN parameter.

STatS=

Overrides corresponding TIPGEN parameter.

termSiZe=

Overrides corresponding TIPGEN parameter.

TermtYP=

Overrides corresponding TIPGEN parameter.

TIMeoff=

Overrides corresponding TIPGEN parameter.

TIMeouT=

Overrides corresponding TIPGEN parameter.

TIPDUMP=

This keyword allows the specification of the processing that is to occur when (if) TIP/30 terminates abnormally.

TIPDUMP=OS

Specifies that TIP/30 is to issue an operating system CANCEL directive when an abnormal condition is detected (and therefore produce a jobdump or whatever type of dump is specified for the job).

TIPDUMP=NO

Specifies that no special processing is to occur.

TIPDUMP=NOSUBMIT

Specifies that a dump image is to be written to the TIP$DUMP file, but do not schedule a job to process the dump.

TIPDUMP='RV ...'

If the TIPDUMP specification is a quoted string, it is assumed to be an RV command (or SC command) that is to be submitted to the operating system by TIP/30 after the dump image is written to the TIP$DUMP file.

The supplied job stream TJ$DMP is provided to process dump images written to the TIP$DUMP file. If the quoted string is too long to fit on one card image, break the specification into two or more instances of this keyword — be sure to break the string at a multiple of 8 characters!

If this parameter is not specified, the default action is to write a dump image to the TIP$DUMP file and submit the command "RV TJ$DMP" to the operating system just before terminating abnormally.

UpPeR=

Overrides corresponding TIPGEN parameter.

UserPRI=

Overrides corresponding TIPGEN parameter.

WARMstrt=

YES	This is the default value for this keyword — TIP/30 is to examine the TIP$B4 file and roll back changes made to HOLD=TR files by transactions that did not complete normally.
NO	TIP/30 is NOT to examine the TIP$B4 file and roll back changes made to HOLD=TR files by transactions that did not complete normally. The TIP$B4 file is reset by this option.
ONLY	The actions indicated by "YES" are to be performed, BUT TIP/30 is to shutdown immediately after roll back (the ICAM network is NOT opened — users do not have an opportunity to logon to TIP/30).

WARM=NO should only be considered if some sort of error on the TIP$B4 file prevents TIP/30 from properly initializing. It would be prudent to attempt offline batch QUICK roll back before resorting to this option.

WARM=ONLY is implemented to enable sites to perform the roll back and keep users from logging on the system (this may be useful when an abnormal shutdown occurred just before a scheduled shutdown and TIP/30 was not going to be brought back up immediately).

If no errors are detected during TIP/30 warmstart, the TIP$B4 file is initialized **regardless of the setting of WARMstrt=.**

Specifying WARMstrt=NO bypasses the last chance to perform the roll back of any interrupted transactions!

XMIT=

Overrides corresponding TIPGEN parameter.

XmitALL=

Overrides corresponding TIPGEN parameter.

XmitCHan=

Overrides corresponding TIPGEN parameter.

XmitVAR=

Overrides corresponding TIPGEN parameter.

Example:

```
// EXEC TB$TIP,TIP$LOD,1
/$
AR1TCA
RESIDENT=(TT$FSE,TT$TCP)
DEBUG=NO
LOCAP=TIP1
READY=YES
CLOSE=TIP$TSP
/*
```

Additional Considerations:

The tcaname (AR1TCA in the example shown above) must be the first positional parameter in the imbedded data set.

Keywords that specify a list of items (CLOSE=, RESIDENT=, RESMOD= etc.) may appear more than once (continuation of the list is implied in that case).

6.1. TIP/30 UPSI

TIP/30 sets the UPSI switch to indicate the circumstances of TIP/30 termination. The setting of the UPSI can be interrogated by subsequent job steps to determine whether or not any special processing needs to be performed.

A description of the various UPSI settings follows (if the hexadecimal value is shown with the character "x", that nibble is irrelevant in that case):

X'9x' Initialization failure. Abnormal termination.

X'Ax' Initialization failure during online rollback or ICAM initialization. Abnormal termination.

X'Cx' Initialization failure during mainline processing. Abnormal termination.

X'8x' Failure in termination processing. Abnormal termination.

X'1x' Premature termination before TIP$B4 processing complete. Abnormal termination.

X'00' Normal termination — No rollback of interrupted transactions required.

The right (low order) nibble of the UPSI may be set to indicate whether or not transactions were interrupted and whether or not a dump was taken:

X'x1' If this bit is set, the TIP$B4 file has information that must be used to rollback transactions that were in progress. This can be performed by running TIP/30 with the WARMSTRT=ONLY option (for example).

Note: This bit is undefined if the UPSI value is set to X'1x'.

X'x8' If this bit is set, a dump was written to the TIP$DUMP file.

Section 7
Offline (Batch) Recovery

The TB$RCV batch program (as executed by the supplied job stream TJ$RCV) is an offline recovery program that provides a number of important capabilities:

- selectively roll back updates made to online files
- apply updates to a backup version of online files
- roll back updates made to online files by transactions which did not complete because the system crashed.

Transaction roll back or roll forward can be performed only on online files which are configured in the TIP/30 generation as HOLD=TR (hold for transaction). The FCS section of this manual contains a description of record locking and online roll back.

The recovery process is accomplished by referring to information that has been logged in the TIP/30 Journal file (LFD=TIP$JRN), the TIP/30 log tape (LFD=TIP$LOG) or the TIP/30 History file (LFD=TIP$HST). The TB$RCV program is able to use any of these files as input.

It may be necessary to ROLL BACK updates to a file if an error is discovered in a transaction program (for example an erroneous computation that was put into production).

The classic example of ROLL FORWARD is required when a disk failure (crash) occurs and completed transactions have to be reapplied to a backup copy of the online files that were on that disk.

The TB$RCV program reads commands from an imbedded data set in the job stream. The commands must appear within columns 1 through 70 of the card image.

All commands must begin with the word "ROLL" or "QUICK" and end with a semicolon (;). Spaces are normally used to separate items in the command. Commands may be continued on additional card images — the semicolon is used to indicate the end of each command.

There MUST be a // PARAM statement preceding the imbedded data which identifies the TCA used when TIP/30 was executing. The TB$RCV program uses the file definitions from the TIP/30 generation. The TB$RCV program must be executed from the TIP$LOD library.

Syntax:

```
// EXEC TB$RCV,TIP$LOD
// PARAM tcaname
/$
QUICK;
ROLL  direction  lfd   [FOR clause]  [FROM clause]  [TO clause];
ROLL  ...
ROLL  ...
 ...
/*
```

Where:

tcaname The name of the TIP/30 control area (TCA) which was in use while TIP/30 was executing.

The TCA name selected implicitly defines the files that may participate in the recovery process.

Whenever files are removed from the TIP/30 generation parameters, the system programmer must remember that the new TCA (without the file definition) may NOT be appropriate for a subsequent ROLLBACK or ROLL FORWARD.

QUICK;

This keyword indicates that TB$RCV is to perform a roll back of updates that were in progress at the time of a crash (this type of roll back is normally performed using data in the TIP$B4 file).

This activity is normally performed when TIP/30 begins executing — it may, however, sometimes be more convenient to perform this activity in a batch environment.

> **WARNING**
>
> Quick roll back by TB$RCV is NOT journaled to the TIP$JRN or TIP$LOG file (the quick roll back that may be performed by TIP/30 startup is written to the journal or log file and therefore is preferable).

ROLL Each ROLL command (there may be several) specifies the recovery action to be taken for a particular file (or all files). Updates can be rolled forward (that is, re-applied) or rolled backward (undone).

direction

The direction the ROLL (recovery) is to take.

Choose one of "FORWARD" (reapply updates) or "BACKWARD" (roll back updates).

lfd

The LFD name of the file this command applies to.

*ALL may be specified if this command is to apply to all eligible files in the TCA.

FOR clause

the word "FOR" is required if this clause is used.

USER=uid — process updates done by this user.

TRAN=trid — process updates done by the identified transaction.

TERM=tid — process updates done at the identified terminal.

FROM clause

the word "FROM" is required if this clause is used.

FROM YY/MM/DD HH:MM

process recovery records with a timestamp **after** this date and time.

Note: *This time stamp is interpreted literally by the recovery program — we recommend that a time is chosen when TIP/30 was NOT running to avoid attempting to perform recovery from a point in time when a transaction was in progress.*

TO clause

the word "TO" is required if this clause is used.

TO YY/MM/DD HH:MM

process recovery records with a timestamp **before** this date and time.

Note: *This time stamp is interpreted literally by the recovery program — we recommend that a time is chosen when TIP/30 was NOT running to avoid attempting to perform recovery to a point in time when a transaction was in progress.*

; (semi-colon)

Each recovery request must be terminated by a semicolon character (since one request may span several card images).

As many commands as required may be entered.

A quick recovery will do a scan of the input file (TIPJRN, TIPLOG, or TIP$B4) to create recovery requests for all files which TIP/30 had active and was logging at the time of an abnormal shutdown.

A quick recovery will also cause updates done by transactions in progress (at the time of the crash) to be rolled back to the state of the record before the transaction started.

If QUICK recovery is specified, it must be stated first in the TB$RCV control stream; it may be followed by other 'ROLL' requests. QUICK recovery is actually performed last — after all other recovery requests (transactions that were interrupted by a crash were presumably the last online activity!).

If TIP/30 was creating a log tape (TIP$LOG) then the recovery job control should assign the input tape using an LFD name of TIP$LOG.

When doing a QUICK recovery from the before image file, the user should assign the before image file (in place of the journal file) with an LFD name of TIP$B4.

7.1. Quick File Recovery

Prior to release 3.1 of TIP/30, TB$RCV had to be executed as a separate job step (before executing TIP/30) to roll back any partially completed transactions. Starting with release 3.1, TIP/30 automatically performs this function as a normal startup procedure.

As part of TIP/30 initialization, the quick before look file (TIP$B4) is read and any updates which were performed by transactions that did not terminate normally will be rolled back.

There is a TIP/30 run time parameter which may be specified to control whether or not this automatic warm start is to be performed: **WARMstrt=** (refer to the section describing TIP/30 run-time options elsewhere in this manual).

WARNING

Quick roll back by TB$RCV is NOT journaled to the TIP$JRN or TIP$LOG file (the quick roll back that may be performed by TIP/30 startup IS reflected in the journal or log file and therefore is to be preferred).

7.2. Journal File Maintenance

This section describes the files that may be involved in journaling and recovery operations. The job streams that may be used to perform routine maintenance on these files are also described.

Keep in mind that this discussion applies only to online files that are generated in the TIP/30 parameters as HOLD=TR.

TIP$B4

The TIP$B4 file is a disk file that contains BEFORE images that TIP/30 (or TB$RCV) may use to roll back transactions that do not complete.

Normally, this file does not grow very large and the allocated file space is automatically reused by TIP/30.

Unless TIP/30 experiences an abnormal shutdown, the contents of this file are of no interest.

If TIP/30 experiences a crash (HPR, internal TIP/30 failure etc.), there may be BEFORE images in the TIP$B4 file which have to be rewritten to the corresponding files to reverse the effects of updates that were done by transactions which did not complete before the TIP/30 crash.

TIPJRN / TIPLOG

The journal file (TIP$JRN) or the log file (TIP$LOG) are (respectively) disk and tape implementations of a recovery audit file. Either or both of these files may be in use depending on site requirements.

Regardless of the choice of media, the information written to the files is identical.

The TIP$JRN (disk) file grows as the online transaction activity proceeds. Depending on TIP/30 generation options, there may be a large number of records output to the file.

If the TIP$JRN file is to be used simply for audit trail purposes, it can simply be saved (via DMPRST) and initialized whenever it is convenient (for example: after TIP/30 shutdown).

If the TIP$JRN file is to be used for recovery purposes, the initialization must be co-ordinated with other events.

Recovery (by this is meant ROLL FORWARD updates applied to a backup) must have access to a consolidated journal file or files so that all updates can be reapplied to a specific backup set.

The journal file can only be initialized when there is confidence that every update done thus far has been correctly completed and a full backup has been taken and verified.

Since one rarely has that sort of confidence, an alternative mechanism is available for archiving journal information:

TIP$HST

The TIP/30 history file (TIP$HST) is a disk file that may be used to accumulate journal information. Since all records written to the journal file are time stamped, it is quite feasible to append the journal file information for each TIP/30 session to a history file.

Once the journal information has been appended to the TIP$HST file, the journal file may be initialized and logging can begin anew.

If a catastrophe occurs, one should append the current journal file information to the history file and then use the history file (if necessary) as input to the batch recovery program to rebuild data files from a backup etc.

Since the history file is merely a concatenation of a number of journal files, the history file can be periodically archived to tape (or whatever) and initialized. This activity is likely to be included as part of a regular off site backup procedure.

7.3. Recovery Batch Jobs

A number of job control streams are provided with TIP/30 that may be useful for maintenance and backup of the various files that are involved in recovery and journaling procedures. The global parameters for these job streams are described later in a section that describes *all* supplied job streams.

What follows here is a brief description of each of the jobs that is supplied for file recovery subsystems.

TJ$B4 This job is the installation job which creates a TIP$B4 file.

Although the job has a global to allow scratching an existing TIP$B4 file, the user will probably find that there is seldom a need to use this job stream.

The TIP$B4 file space is normally reused by TIP/30; recreating this file would only be required in rare situations where abnormally high transaction activity caused the file to extend more than anticipated.

If the TIP$B4 file is to be moved, this job should be used to scratch the existing file and create a new one on the desired volume. This job recreates the job control PROC that assigns the TIP$B4 file (or updates the YCAT entry) — thereby eliminating some housekeeping.

TJ$JRN This job is the installation job which creates a TIP$JRN file.

Since the journal file can (and usually does!) extend to accommodate online activity, the journal file should be monitored for abnormal extension.

Before the journal file gets out of hand (say weekly?) the user should copy the TIP$JRN file to the history file (if one is in use) and then run the TJ$JRN job to recreate a new journal file.

There is a job (TJ$JR2HS) which copies the journal information from the journal file to the history file and resets the journal file pointers to reuse the space. This procedure would not "shrink" the journal file back to a normal size if it had extended.

TJ$HST This job is the installation job which creates a TIP$HST file.

Since the TIP$HST file is designed to extend to hold the contents of a number of journal files, the history file should be archived and recreated to avoid tying up a large extent all of the time.

There are no supplied job streams to archive the TIP$HST file — DMPRST (to tape) is probably the most popular method of archiving this file.

TJ$JR2HS

This supplied job stream appends the TIP$JRN file (or a TIP$LOG file) to the TIP$HST file.

After copying the TIP$JRN file to TIP$HST, the job simply initializes the TIP$JRN file (beware! this does not shrink the TIP$JRN file if it had extended).

TJ$JRINT

This supplied job stream simply initializes the TIP$JRN file (beware! — it does not shrink the file if it had extended).

Some sites use the TIP$JRN file simply as an audit trail and need a job stream to "reset" the TIP$JRN file after the audit information has been read and printed.

Sites such as those tend to have a small TIP$JRN file and seldom worry about rampant extension of the file.

TJ$RCV This job stream will perform a complete roll forward of updates from a specified input file (usually the TIP$JRN file).

TJ$LST This job stream prints a report from the specified input file (default is TIP$JRN) detailing the journal information in the file.

Global parameters are provided to govern the sort that takes place and to influence the types of records that are selected (ALL, by user, by terminal name, etc.)

Section 8
TIP/30 Batch Jobs

This section describes many of the batch job streams that are supplied to support the online TIP/30 system. Only those job streams that are likely to be run on a regular basis are described — many of the jobs are intended to be run only under the control of the installation programs and are not documented here.

The TIP/30 installation procedure copies the supplied job streams to the private job control library defined for the TIP/30 system to which they apply. The installation procedure also creates (in that library) important job control procs that are used by the job streams. The intention is that these jobs must be run from that library so that they will access the correct job control procs.

The supplied job streams have job names (and element names) which begin with the three characters: **TJ$.**

Before running any of these jobs, examine the job control stream for the job *as it appears in your job control library* and as it is described in this section to make certain that you understand what global parameters (if any) are required and the ramifications of running the job.

The actual JCL statements for these jobs are NOT listed here simply because JCL can contain errors (just like programs) and any listing here might be somewhat different than the latest field release job control.

The supplied job control is intended to execute correctly on all OS/3 releases supported by TIP/30.

WARNING

Modification of the job streams is not recommended. If you experience difficulty running any of the jobs or have suggestions for modifications please contact the support department personnel.

The supplied job stream TJ$TIP is a skeleton job control stream for executing TIP/30. This job stream is normally not modified but is instead "cloned" and localized for the site to take into account any site dependent job control conventions and requirements.

8.1. TIP/30 Job Control Procs

The job control streams supplied with TIP/30 presume the existence of a number of job control procs (also supplied with TIP/30 or dynamically created by supplied jobs). These procs are referenced extensively in the supplied job streams and should not be modified without very careful consideration and study.

The following procs are assumed to exist: (the name is the element name used to file the proc in the job control library):

TIPDATA This proc defines all user online data files (that is, all online files other than libraries).

The TIPDATA proc is used in the batch recovery job stream to make sure all data files referenced by the online system are defined to the TIP/30 offline recovery program.

Avoid specifying the ACCESS= parameter on any job control DD statements if you intend to make use of the ability of the FOPEN transaction (or TIP/30 operator console command) to dynamically alter the ACCESS that TIP/30 has to a file (a job control ACCESS= specification overrides all other file sharing specifications).

Refer also to the discussion of ACCESS= in the description of the FOPEN utility transaction.

TIPENV This proc defines a number of globals that are used by the TJ$MAINT job. This proc is intended to be used internally and is normally not of interest.

TIPFILES An extensively used proc which defines a PRNTR file, the TIP release library (LFD=TIP) and the TIP/30 load library (LFD=TIP$LOD).

The TIPFILES proc accepts a single positional parameter which is used as the first parameter on a // SPL statement in the device assignment that is generated for the print file PRNTR.

The TIPFILES proc also defines a number of global symbols that may be referenced by jobs that use this proc.

TIPICAM This proc defines the disk files required by ICAM. The files are normally defined as job temporary files (although other techniques are possible).

These files are used by ICAM for queueing messages on disk.

In a GLOBAL ICAM environment, the ICAM disk files must be defined in the job control for the GUST job.

TIPLIBS This proc defines all the libraries that are used online with TIP/30.

The TIPLIBS proc is also referenced in the job streams that are supplied to compile and link online programs; for example: TJ$COB74 and TJ$RPG.

TIPSCR This proc is used by various TIP/30 job streams to conditionally scratch a file that is about to be created or re-created. This proc is intended to be used internally and is normally not of interest.

8.2. TIP/30 Supplied Job Control

This section provides a list of the supplied job control streams (in alphabetical order by job name) and gives a brief description of the purpose of the job stream and the globals that may be specified when the job is run.

These jobs must be run from the library where the TIP/30 job control is located for the appropriate TIP/30 system. Although most of the supplied jobs have global parameters that permit the proc names to be overridden (TF=xxxxxxxx), there should seldom be a need to specify alternate proc names. The jobs use the information in the TIPFILES proc (from the private library where they are executed) to identify the environment for the job.

The TIPFILES proc is automatically included by all of these job streams. The TIPFILES proc defines a number of globals that can affect the execution of jobs. An important global is "$PRI".

The $PRI global is used to set the execution priority for the job. The default is priority 6. If a different priority is desired for a particular job, it may easily be overridden:

```
RV  TJ$COB74:JT,,$PRI=1,E=PAY010,L=PAYSRC
```

TJ$COB74 Compile and link online TIP/30 COBOL-74 language program.

TF=	May be used to specify a proc to be used in place of the TIPFILES proc.
TL=	May be used to specify the proc to be used in place of the TIPLIBS proc.
E=	Specify the element name containing the COBOL-74 program source. This global is required.
L=	Specify the LFD name of the *input* library. The default for this global is the SYSGEN library.

TJ$COB85 Compile and link online TIP/30 COBOL-85 language program.

TF=	May be used to specify a proc to be used in place of the TIPFILES proc.
TL=	May be used to specify the proc to be used in place of the TIPLIBS proc.
E=	Specify the element name containing the COBOL-85 program source. This global is required.
L=	Specify the LFD name of the *input* library. The default for this global is the SYSGEN library.

TJ$COP Copy the load modules for all online transaction programs supplied with TIP/30 from the TIP (release) library to the TIP$LOD library. This job is normally used to refresh the TIP$LOD library with the load modules from a new TIP/30 release library.

WARNING: this job automatically performs a pac operation on the TIP$LOD library — do not pac the TIP$LOD library while the related TIP/30 system is running!

TF= May be used to specify a proc to be used in place of the TIPFILES proc.

DEL= Default: N.

This global controls whether or not existing TIP/30 load modules will first be deleted from TIP$LOD. DEL=Y need only be specified to update an existing TIP$LOD library.

RPG= Specify "N" to suppress the step to copy TIP/30 RPG II modules to the YOBJ library. Default is RPG=Y (copy RPG II modules to YOBJ).

TJ$COR Skeleton jcl for installing TIP/30 patches. This job should only be used under the direction of support department personnel.

TJ$CRBAK Backup TIP$CAT, TIP$RNDM, TIP$MCS files to TIP$BAK file (tape or disk).

TF= May be used to specify a proc to be used in place of the TIPFILES proc.

M= The M= global controls the type of output media (M=T indicates tape output; M=D indicates disc output). If the output media is tape, the job will prep the tape.

Default: M=T

V= The V= global controls the volume serial number of the output media.

Default: V=TIPBAK

TJ$CRRST Restore TIP$CAT, TIP$RNDM and TIP$MCS files from TIP$BAK file (tape or disk).

TF= May be used to specify a proc to be used in place of the TIPFILES proc.

M= The M= global controls the type of input media (M=T indicates tape input; M=D indicates disc input).

Default: M=T

V= The V= global controls the volume serial number of the input media.

Default: V=TIPBAK

FMT= This global controls whether or not the TIPCAT, TIPMCS and TIP$RNDM files are to be pre-formatted before the restore procedure begins. The default is FMT=Y.

Specify FMT=N to avoid the pre-format step (if, for example, you are making use of the ability of the backup/restore program to selectively restore from a backup — see the description of the TB$CRB batch program).

Before formatting proceeds, a password is solicited from the console. Refer to the description of the console message for a discussion of the password that is required (see reference to console message "TI105").

TJ$DEL Delete a specified element from any library that is defined in the TIPFILES or TIPLIBS job control procs.

TF= May be used to specify a proc to be used in place of the TIPFILES proc.

TL= May be used to specify the proc to be used in place of the TIPLIBS proc.

L= Global to indicate the LFD name of the library containing the element to be deleted.

Default: L=TIP$LOD

E= Specify the name of the element to delete. This global is required.

TY= The TY= global controls the type of library element that is to be deleted. Possible type codes are the same as those codes accepted by the LIBS utility (S=source, L=load, O=object, etc).

Default: TY=L

TJ$DMP TIP/30 Dump Processor. This job stream processes a dump of TIP/30 that has been written to the TIP$DUMP file.

TF= May be used to specify a proc to be used in place of the TIPFILES proc.

ASK= This global controls whether or not the dump program is to issue a prompt before printing the dump. Specify ASK=N to suppress the prompt.

Default: ASK=Y.

MODE= This global controls the recording density of the output tape that the TJ$DMP program creates when tape output is selected. Specify MODE=1600 or MODE=6250 or the actual job control mode code (such as NMC0).

Default: MODE=1600.

ST= This global controls the disposition of the spool file output of this job. Specify ST=HOLD or ST=RETAIN to hold or retain (respectively) the spool file for this job.

Default: ST= (spool file is not held).

The TY= global controls the desired processing of the TIP/30 dump file. Choices are:

TY=PD Print from disk. This specification prints the entire dump in the TIP$DUMP file.

TY=SD Show from disk. This specification prints a one page summary of the dump in the TIP$DUMP file.

TY=RD Reset dump file. This specification marks the TIP$DUMP file as "processed" — this is typically used to ignore a TIP$DUMP that was produced and is not wanted.

TY=CDT Copy from disk to tape. This specification copies the TIP$DUMP file contents to an output tape.

TY=CTD Copy from tape to disk. This specification creates the TIP$DUMP file contents from an input tape.

TY=PT Print from tape. This specification prints the entire dump from a previously created tape copy of the TIP$DUMP file.

TJ$GUST Canned job stream to run GUST (Global ICAM User Services Task) — handy if you wish to run Global ICAM but do not have the JCL figured out). The ICAM network name that is used by this job is defined in the TIPFILES job control proc.

TF= May be used to specify a proc to be used in place of the TIPFILES proc.

TI= Used to specify the proc to be used in place of the TIPICAM proc.

Default: TI=TIPICAM

ICAM= This global allows the specification of the ICAM symbiont name that is to be loaded by this job. If ICAM is already running, the attempt to start another ICAM symbiont is graciously ignored by OS/3.

Default: ICAM=C1

TJ$INS This job is the main TIP/30 installation job. The documentation for this job can be found in the TIP/30 Installation Guide.

TJ$JCS Copy JCL from the TIP/30 library into your JCL library.

This job stream is described in the TIP/30 Release Notice.

TJ$JRINT Initializes (pre-formats) the TIPJRN, TIPB4 or TIP$HST file.

TF= May be used to specify a proc to be used in place of the TIPFILES proc.

F= This global controls the file to initialize. The default value is the Journal file that is identified by the TIPFILES proc that is used.

Specify F=B4 to initialize the TIP$B4 file.

Specify F=HST to initialize the TIP$HST file.

TJ$JR2HS This job copies the TIPJRN, TIPLOG or TIP$B4 file to the TIP$HST file. Typically, this job is used to copy the contents of the TIP/30 journal file to the journal history file (TIP$HST). This job automatically initializes the input file after it is copied to the output file.

TF= May be used to specify a proc to be used in place of the TIPFILES proc.

VI= VSN of the input file. This global is required if the input is tape.

MI= This global specifies the type of media of the input file. Specify MI=T for tape input (a TIP$LOG tape), or M=ID for disk input (TIP$JRN or TIP$B4 file).

Default: MI=D.

VO= VSN of the output file. This global is required if the output is tape.

MO= This global specifies the type of media of the output file. Specify MO=T for tape output (a TIP$LOG tape that is to be extended), or MO=D for disk output (TIP$HST file).

Default: MO=D.

B4= This global is used to indicate the special case that the TIP$B4 file is the desired input file. Specify B4=Y to indicate that the TIP$B4 file is the input file (MI=D must also be specified to indicate the input is disk).

Default: B4=N.

TJ$LC Sort and list the TIP$CAT file. Also summarize the TIP/30 screen formats that are defined in the TIP$MCS file.

TF= May be used to specify a proc to be used in place of the TIPFILES proc.

MCS= This global controls whether or not information about the TIP$MCS (screen format file) is to be listed. The default is "Y".

XREF= This global controls whether or not an XREF is desired for all catalogue information that is listed. The default is "Y".

USER= This global permits the specification of a particular TIP/30 userid. If a specific user is specified via this global, the job displays only the TIP/30 catalogue information that is accessible by that user.

If this specification is omitted (the default case) the entire TIP/30 catalogue is listed.

TJ$LCOS3 Sort and list the OS/3 YCAT file.

TF= May be used to specify a proc to be used in place of the TIPFILES proc.

TJ$LOAD Load the TIP/30 release tape.

This job stream is described in the TIP/30 Release Notice.

TJ$LOG Prep tape for TIP$LOG.

TF= May be used to specify a proc to be used in place of the TIPFILES proc.

V= This global is used to specify the desired tape volume serial number. This global is required — the job attempts to initialize (PREP) a tape with the volume serial specified.

The tape is prepped with LBL name "TIP.id.LOG".

TJ$LST Sort and list information from TIPJRN, TIPLOG, TIP$B4 or TIP$HST file.

TF= May be used to specify a proc to be used in place of the TIPFILES proc.

F= This global specifies the input file to be listed. The default is the TIP$JRN file that is identified by the TIPFILES proc that is in use.

The value of this global is used to construct the suffix of the input file LBL name (for example: TIP.id.xxx).

Other specifications are: F=LOG, F=B4 and F=HST.

WK= This global specifies the number of blocks to allocate to each of the three work files that are assigned by this job (WORK1, WORK2 and WORK3).

Default: WK=4000.

LIST= This global identifies the type of listing that is desired: If this global symbol is empty (or omitted), the job stream assumes that one of the globals (A=, L=, P=, T=, Y=, U= — see later) will be specified.

ALL Specifies that the entire input file is to be listed. Choosing this value causes the other globals (A=, L=, P=, T=, Y=, U=) to be ignored.

SUMMARY

Specifies that a *summary* listing of the input file is desired. Choosing this value causes the other globals (A=, L=, P=, T=, Y=, U=) to be ignored.

A=(,,,) This global permits the specification of up to 7 TIP/30 account numbers. Input file information that is related to any of the specified accounts will be listed.

L=(,,,) This global permits the specification of up to 7 TIP/30 LFD names. Input information that is related to any of the specified LFD names will be listed.

P=(,,,) This global permits the specification of up to 7 TIP/30 transaction ids. Input file information that is related to any of the specified transactions will be listed.

T=(,,,) This global permits the specification of up to 7 terminal ids. Input file information that is related to any of the specified terminals will be listed.

U=(,,,) This global permits the specification of up to 7 TIP/30 userids. Input file information that is related to any of the specified users will be listed.

Y=(,,,) This global permits the specification of up to 7 journal record types. Input file information from the specified record types will be listed.

TJ$PAC Job to PAC a library using the LIBS utility.

TF= May be used to specify a proc to be used in place of the TIPFILES proc.

TL= May be used to specify the proc to be used in place of the TIPLIBS proc.

L= This global specifies the LFD name of the library to be packed. The default value is L=SYSGEN.

Note: *Some libraries (TIP$LOD for example) must not be packed while TIP/30 is running!*

TJ$PARAM TIP/30 generation parameter validation and generation.

This job also builds (and optionally schedules) the second and final job in the generation procedure).

This job stream is described in the TIP/30 generation procedures in "4.6. Generation Parameter Processor" on page 4-52.

TF= May be used to specify a proc to be used in place of the TIPFILES proc.

TL= May be used to specify the proc to be used in place of the TIPLIBS proc.

RUN= This global controls whether or not the second job stream is to be automatically scheduled.

Specify RUN=NO to inhibit scheduling the TJ$GEN (generated) job.

Default: RUN=AUTO (schedule TJ$GEN if no parameter errors).

L= Specify the LFD name of the library where the TIP/30 generation parameters are located. Default is L=SYSGEN.

TCA= This global is used to specify the element name of the set of TIP/30 generation parameters that are to be scrutinized by the TIP/30 parameter processor.

TJ$RCV Off line batch recovery using the TIP/30 Journal Files.

TF= May be used to specify a proc to be used in place of the TIPFILES proc.

TD= May be used to specify the proc to be used in place of the TIPDATA proc.

F= This global specifies the input file to the recovery program. The default is F=JRN (the TIP$JRN file that is identified by the TIPFILES proc that is in use).

The value of this global is used to construct the suffix of the input file LBL name (for example: TIP.id.xxx).

L= Specify a *single* LFD name that is to be recovered.

Default: *ALL (all files are eligible for recovery).

Note: *This job stream is designed to recover either a single LFD or all eligible LFD names. If you wish to recovery some subset of all of the LFD names, you must either run this job stream multiple times or make a copy of this job stream and manually insert appropriate ROLL commands for the LFD names involved.*

MIN= This global controls the amount of memory allocated to the job. The value is used directly in a // OPTION MIN= statement and must be coded appropriately for that purpose.

Default: MIN=50000 (hexadecimal).

ROLL= Specify the type of offline recovery desired.

Default: ROLL=FORWARD

Other choice is ROLL=BACKWARD

FORWARD recovery re-applies updates for specified LFD names by using information in the journal file.

BACKWARD recovery rolls back updates for specified LFD names by using information in the journal file.

TCA= Specify the name of the TIP/30 generation parameters (TIP/30 Control Area — TCA) that is to be used to determine the configuration parameters for the files that are to be used.

If this parameter is not specified, a QGBL prompt is issued to solicit a value.

It is important to use the tcaname that was in use when the input journal file was created (because the recovery program obtains file information from the tca information).

TJ$RENAM Rename a file.

FI=	This keyword specifies the LBL name of the input (old) file. This global is required.
FO=	This keyword specifies the LBL name of the output (new) file. This global is required.
L=	This keyword specifies the LFD name of the new file. The LFD name is used to catalogue the file in the operating system catalogue (YCAT). This global is required.
V=	This keyword specifies the VSN of the existing (old) file. This global is required.
CAT=	This keyword specifies whether or not the existing (old) file is defined in the system catalogue (YCAT). Specify CAT=N if the old file was not in the catalogue. Default: CAT=Y.
A=	This keyword permits the job account number of the job to be specified.

TJ$RPG Compile and link an online TIP/30 program written in the RPG II language.

TF=	May be used to specify a proc to be used in place of the TIPFILES proc.
TL=	May be used to specify the proc to be used in place of the TIPLIBS proc.
E=	Specify the element name of the RPG program source. This global is required.
L=	Specify the LFD name of the *input* library. The default for this global is the SYSGEN library.

TJ$SCR Scratch (and decatalogue) a file.

V=	This global specifies the volume serial number of the volume containing the file to be scratched. This global is required.
F=	This global specifies the LBL name of the file to be scratched. This global is required.
CAT=	This global may be specified as CAT=N if the file to be scratched is not catalogued in the OS/3 system catalogue (YCAT). Default CAT=Y (scratch file and decatalogue file.
D=	The device number to use to assign the file. Default: D=50.

WARNING

TJ$SCRTP is an extremely dangerous job — it scratches entire TIP/30 systems!

TJ$SCRTP Scratch and decatalogue an entire set of TIP/30 system files. This job schedules a number of instances of the subordinate job TJ$SCR1.

ID= This global specifies the TIP/30 system identifier of the file set to be scratched.

This global is required. The value of the global is used to construct the middle portion of the LBL names to scratch and decatalogue. For example, ID=TEST results in job names of the form: TIP.TEST.xxxxx

Be VERY careful when specifying this parameter!

J= This global specifies the LBL name of job control library to be used to execute the subordinate job TJ$SCR1. This global is required. The TJ$SCRTP job issues a QGBL prompt at run time to ask whether or not this job control library is to be scratched.

TJ$TIP Skeleton job control for TIP/30. Used to execute the bootstrap version of TIP/30 with the BOOTCA generation parameters.

This job stream is normally "cloned" and tailored to any specific site job control requirements (such as running any pre-TIP or post-TIP job steps).

TJ$UPRPG Modify OS/3 YOBJ library to supoprt TIP/30 RPG II programs. A number of replacement object modules are copied from the TIP library into YOBJ and then renamed. This job is required for sites which run or intend to run TIP/30 programs written in the RPG II language. This procedure is also imbedded in the TJ$COP job.

TF= May be used to specify a proc to be used in place of the TIPFILES proc.

Note: *This job PACs the YOBJ library after the replacement modules are copied to that library.*

8.3. Batch Program TB$CRB

The batch program TB$CRB is a specialized dump/restore program for the TIP/30 files TIPCAT, TIPRNDM and TIP$MCS. These files are crucial to the operation of the TIP/30 system. The files are implemented as System Access Technique (SAT) files but they are NOT libraries and cannot be manipulated with OS/3 utilities other than DMPRST.

This utility is provided to manipulate the files as a related set of files.

The program is most often used as a simple dump/restore program for the files; however, there are run-time control cards that may be specified to control the behaviour of the program during a restore operation.

During a restore operation, data from the CATALOGUE (TIP$CAT), RANDOM FILE (TIP$RNDM) or the screen format file (TIP$MCS) may be selected or omitted as desired.

Parameter cards may be submitted to the TB$CRB program via an imbedded data set (/$ through /*). These parameters are free-format and may be submitted in any order (since all parameter cards are read before the program begins processing).

DUMP=TAPE

Indicates that TB$CRB is to dump all of the information from TIPCAT, TIPRNDM and TIP$MCS to tape (TIP$BAK).

This is the default behaviour of the TB$CRB program.

DUMP=DISC

Indicates that TB$CRB is to dump all of the information from TIPCAT, TIPRNDM and TIP$MCS to disc (TIP$BAK).

RESTore=TAPE

Indicates that TB$CRB is to restore information from an input tape (TIP$BAK) to TIP$CAT, TIP$RNDM and TIP$MCS depending on any further SEL or DEL information.

RESTore=DISC

Indicates that TB$CRB is to restore information from an input disc file (TIP$BAK) to TIP$CAT, TIP$RNDM and TIP$MCS depending on any further SEL or DEL information.

Only one of DUMP= or RESTORE= should be specified; they are mutually exclusive operations!

VERIfy=NO

Indicates that TB$CRB is NOT to verify the contents of the tape/disc produced by the DUMP= option.

Default is VERIFY=YES

The verification procedure should *always* be specified for DUMP operations.

This option is ignored on a RESTore operation.

FCSXTENT=nn

Indicates the size of the FCS extent and must match the value specified in the corresponding TIP/30 generation parameter.

Default is 40 (same default as TIPGEN).

This value is validated to be between 10 and 256.

On a DUMP operation this value must match the FCSXTENT value for the TIP$RNDM file that is being dumped — otherwise DATA WILL PROBABLY BE LOST!.

On a RESTORE operation this value must match the FCSXTENT value for the TIP$RNDM file that is being reloaded. To be able to access the reloaded TIP$RNDM, the TCA must specify the same FCSXTENT= value.

If the value of FCSEXTENT is being changed, the TIP$RNDM file must be initialized before running the restore job (or at least specify FMT=Y as a global for the TJ$CRRST job stream).

The following types of SEL and DEL cards may be specified to control the data that is to be restored from a tape/disc created by TB$CRB.

For the sake of safety, SEL and DEL parameters are only allowed on a RESTORE operation; ALL information is processed during a DUMP operation.

TB$CRB allows approximately 100 DEL/SEL statements (some statements are treated as more than one internal table entry). Since multiple specifications may be mutually exclusive the rule is that SELections are performed first (in the order specified); DELetions are processed after all SELections.

Each of the subparameters on the SEL and DEL statement may be specified using standard TIP/30 prefix notation (*ABC means "begins with ABC").

The subparameters in the CAT SEL command correspond to the four parameters required by the list command in the on-line catalogue manager program (CAT).

The subparameters in the MCS SEL command correspond to the group and screen format name.

CAT SEL=?/?/?/?

Select catalogue (or TIP$RNDM) entries which match the four subparameters specified (the CAT key).

CAT DEL=?/?/?/?

Discard catalogue (or TIP$RNDM) entries which match the four subparameters specified (the CAT key).

MCS SEL=GRP/name

Select screen formats which match the group and screen name specified.

MCS DEL=GRP/name

Discard screen formats which match the group and screen name specified.

WARNING

The default operation of the TB$CRB program is to restore all of the information for the CATalogue and for the MCS files.
The appearance of the first SEL or DEL card causes TB$CRB to abandon the default operation and perform ONLY what is specified by the SEL and DEL cards that appear in the input stream.
For example, if a "CAT SEL=EDP,*,*,*" statement (select all catalogue entries for the group EDP) was the only SEL or DEL statement in the input stream — TB$CRB would not restore any of the MCS screen formats.

Example of use of SEL and DEL cards:

MCS SEL=*,!TEST

Restore screen formats from ALL groups provided the screen name does not begin with the characters "TEST".

CAT DEL=*,*DUMP,,D

Discard (during the restore operation) all catalogue entries (and TIP$RNDM entries!) in any group for dynamic files with names beginning with the characters "DUMP".

For examples of job control streams for this program, refer to the supplied job streams "TJ$CRBAK" and "TJ$CRRST".

Additional Considerations:

The TB$CRB program sets a non-zero UPSI value if the program detects any errors in the parameter statements or during the operation of the program.

Section 9
Operations Guide

This section describes the operational aspects of TIP/30 from the point of view of the system operator. Included is a description of the various TIP/30 console commands that are available to the operator. The console commands allow the operator to monitor and control the operation of TIP/30.

9.1. Console Operation

The system operator should never change the system date or time while TIP/30 is running. If the date or time is incorrect, critical journal information may be incorrectly written.

Furthermore, user programs may be dependent on the date and time for scheduling activity etc.

> **WARNING**
>
> DO NOT CHANGE THE TIME OR DATE WHILE TIP/30 IS RUNNING!

TIP/30 is critically dependent on the timer services provided by the operating system. If the time is changed while TIP/30 is running, TIP/30 may go into a continuous wait state.

If TIP/30 is (inadvertently) executed when the TIME or DATE is not correct, we recommend that you shutdown TIP/30 as soon as possible using the STOP TIP/30 console command. The STOP command does not perform transaction roll back and is suggested because transaction roll back may be dangerous when the time or date is incorrect. Correct the date and time and restart TIP/30. When TIP/30 restarts, the normal start up procedures examine the TIP/30 before image file and perform any necessary roll back operations.

The following section describes commands that may be presented to TIP/30 as unsolicited operator console commands.

An unsolicited command is submitted by prefixing the command text with the string "UNS" followed by a space and the current TIP/30 **jobname**:

```
UNS xxxxxxx WHOSON
```

9.2. TIP/30 Operator Access Control

TIP/30 provides a mechanism whereby console operator access can be controlled through the TIP/30 Catalogue. The reserved TIP/30 userid "CONSOLE" may be created to control which programs and files may be accessed by the TIP/30 console operator.

Some of the operator console commands that are described in the following sections are implemented as transaction programs that are executed in background; the other commands are executed internally by TIP/30. The operator functions that are executed as transaction programs are affected by the presence or absence of a userid named "CONSOLE".

The following console commands are "external" (implemented as transactions):

- APB
- CLOSE (FCLOSE)
- EXEC (xxxxxxxx)
- MSG
- OPEN (FOPEN)
- SET.

External commands execute the corresponding TIP/30 transaction. Exceptions are:

- OPEN and CLOSE commands execute the transactions "FOPEN" and "FCLOSE" (respectively)
- the EXEC command executes the transaction named as the parameter to the EXEC command.

When an external command is executed, the default action of the TIP/30 system is to set the background userid to the string "CONSOLE", set the security level to 1 and set no elective group membership.

If the reserved userid "CONSOLE" is defined in the TIP/30 catalogue, the security and group membership is set to the values specified in that user catalogue record.

Example:

```
USER  CONSOLE  SECUR=29  GROUPS=(EDP,MANUFACT).
```

In this example, the system console operator can execute transaction programs (via the EXEC command) as if he has programmer level security and membership only in the elective groups "EDP" and "MANUFACT".

9.3. TIP/30 Operator Commands

The following commands are all available as unsolicited console key ins to the TIP/30 job.

Note: *Although many of the operator commands have identically named utility transactions, this section documents the behaviour of commands assuming that the command is submitted as an unsolicited console command.*

Some commands are not implemented as utility transactions.

APB ...text...

Send the specified text as a one line unsolicited message to all users logged on TIP/30. The message text does not have to be enclosed in quotes but is restricted to a maximum of 60 characters.

APB/ALL ...text...

Send the specified text as a one line unsolicited message to all terminals currently connected to TIP/30 (whether or not logged on TIP/30).

The message text does not have to be enclosed in quotes but is restricted to a maximum of 60 characters.

CLOSE lfd,lfd ...

Mark the specified file or files as "not available for online use" and issue a Data Management CLOSE operation for each file.

If there are online transactions currently using the file a message is sent to the operator's console indicating the number of current users accessing the file.

No new program is allowed to access the file from this point until a subsequent OPEN command is issued.

Any online program which attempts to access a CLOSED file receives an error status in the PIB-STATUS field.

Online programs currently using the file are allowed to continue using the file. Once no online program is using the file, it is CLOSED and a message is sent to the console operator.

This feature is useful when the operator wants to run a batch program against a file which is being used by the online system and later return the use of the file to the online system.

A number of lfd names may be specified (separated by commas).

A filename may be specified using standard prefix notation. For example:
CLOSE *AP,*PAY

CRASH

Same as STOP command except that a dump is produced. See description of STOP command which follows.

DATE

Display the current date and time.

Example result: TUESDAY FEBRUARY 23 1988 at 11:23

DIE/xxxx

Cancel the program running for the specified user-id (xxxx) or at the specified terminal (xxxx) with a "Process Cancel" error code.

It may be necessary to cause some sort of input (for example by pressing the (MSG WAIT) key) at the specified terminal to enable TIP/30 to cancel the transaction program.

DOwn/line

TIP/30 issues a request to ICAM to set down the specified line.

This command is ignored in a GLOBAL ICAM environment.

If the specified line is a workstation, the terminal is made available to interactive services.

DOwn/term

TIP/30 requests ICAM to set down the line which is implied by the *terminal name* specified *(xxxx)*. (Terminal names are often more readily known than line names).

This command is ignored in a GLOBAL ICAM.

If this is a workstation then the terminal is made available to interactive services.

DUMP

Invoke TIP/30 online dump. TIP/30 writes a dump image to the TIP$DUMP file and resumes normal processing. If the TIP$DUMP file contains a previous dump that has not been processed, this request for a dump is rejected. If an error occurs during the dump process, an operating system jobdump is enabled in case of subsequent failures.

If a dump is successfully taken, this command invokes the processing defined by the run-time TIP/30 job control parameter TIPDUMP= (see description of that option in "TIP/30 Job Control Options" on page 6-1.

DUMPF

This command is identical to the "DUMP" command described above, with the exception that there is implied permission to overwrite any existing dump information in the TIP$DUMP file.

If a dump is successfully taken, this command invokes the processing defined by the run-time TIP/30 job control parameter TIPDUMP= (see description of that option in "TIP/30 Job Control Options" on page 6-1.

EOJ [timeout]

Request a TIP/30 orderly shutdown.

TIP/30 immediately inhibits any further logon requests and waits for all currently logged on users to logoff.

If the timeout parameter is specified (representing a number of *minutes*), programs which are waiting for a specific period of time wait no longer than "timeout" minutes and then are reactivated. There is no guarantee, however, that a program is properly checking for an orderly TIP/30 system shutdown.

It may be necessary to issue a STOP command sometime after the EOJ command is issued to force users off the TIP/30 system.

The operator should follow the procedure that is established by the site administrator.

EOJ OFF

Rescind a previously issued delayed EOJ command.

This command has to be entered relatively quickly after an unintended EOJ command!

EXEC cmdline

Start a transaction program in background.

The supplied text (cmdline) is processed as a normal command line; the indicated transaction is started in background by a TIPFORK operation.

Refer to "9.2. TIP/30 Operator Access Control" on page 9-2 for a description of the mechanism whereby such transactions are executed.

EXEC SET NOLOGONS

Execute the SET program to inhibit TIP/30 logons.

EXEC SET LOGONS

Execute the SET program to allow TIP/30 logons.

FILES/— prefix

Produce an I/O summary report of active OS/3 files assigned to TIP/30. The option field may contain an "O" (FILES/O) or "C" (FILES/C) to restrict the display to files which are currently OPENED or CLOSED (respectively).

Parameter 1 may be specified to filter the resulting output to include only LFD names which match the prefix that is specified. For example, entering FILES *PAY displays information about files with an LFD name that begins with "PAY".

FLAG

Display the current status of the 32 TIP/30 flags.

GO term

Restart a previously PAUSEd process

LMOFF term

Turn off software line monitor on specified terminal.

To obtain the line monitor printout:

1. Breakpoint the print queue for the active TIP/30 job:
   BR ACT,PR,JOB=xxxxxxxx
2. Start a burst mode output writer for the active TIP/30 job:
   PR BX,JOB=xxxxxxxx

LMON term

Turn on the software line monitor for a specific terminal.

A display of all input and output messages (including delivery notification from ICAM) for the specified terminal is printed (in SNAP dump format) on the PRNTR.

Refer to the previous description of the LMOFF command to find out how to obtain the printout.

MSG/term ...text...

Send a one line unsolicited message to the specified terminal.

The text does **not** have to be enclosed in quotes and is restricted to approximately 60 characters.

MSG/user ...text...

Send a one line unsolicited message to the specified user (if logged on).

The text does **not** have to be enclosed in quotes and is restricted to approximately 60 characters.

OFF flag#,...,flag#

Cause the named TIPFLAGS to be placed in the *OFF* state.

The values of flag# may be 0 through 31 inclusive.

This command should not be used indiscriminately; online programs may be critically dependent on the setting of flags.

ON flag#,...,flag#

Cause the named TIPFLAGS to be placed in the *ON* state.

The values of flag# may be 0 through 31 inclusive.

This command should not be used indiscriminately; online programs may be critically dependent on the setting of flags.

OPEN lfd,lfd,...

Mark the file or files specified as "available for online use" and issue a Data Management OPEN for the file(s).

This is the inverse of the CLOSE command.

Several lfd names may be specified (separated by commas).

LFD names may be specified using standard prefix notation (for example: OPEN *AP).

OPEN/SRD lfd,lfd,...

Open the specified file or files and change TIP/30's ACCESS to those files to ACCESS=SRD.

This option opens the specified files so that the online system may read the files and batch jobs may update the files.

OPEN/EXCR lfd,lfd,...

Open the specified files and change TIP/30's ACCESS to those files to ACCESS=EXCR.

This option opens the specified files so that the online system may update the files and batch jobs may only read the files.

PAuse xxxx

PAUSE the specified process (by specifying either the terminal name or the userid).

PURGE/xxxx

Purge the specified user-id (xxxx) or the specified terminal (xxxx) from the system. This command may be needed in situations where a DIE command is unable to cancel a running process.

QCLEAR term,queue

TIP/30 issues a request to ICAM to flush a message queue for the specified terminal.

Only one queue may be specified at a time; the value specified must be "H" (high), "M" (medium) or "L" (low).

SET ...

Invoke the SET utility transaction to alter the attributes of a process or some aspect of the system.

See separate documentation of the SET utility transaction.

STAT

Display TIP/30 statistics report on the console.

STOP

Terminate TIP/30 immediately.

TIP/30 immediately closes all files and terminates.

If user transactions were in progress, they are prematurely stopped (console message TI097 may be generated too — see the description of that console message).

The system shutdown program (if one is specified in the generation parameters or in the TIP/30 job control) is NOT scheduled.

It may be necessary to use this command to shutdown TIP/30 if user programs do not properly recognize a prior EOJ command.

The STOP command does NOT perform transaction roll back.

TERM

List all terminals in the ICAM network, showing status (up/down). If a TIP/30 user is logged on at a terminal, the userid is also shown.

UP xxxx

TIP/30 issues a request to ICAM to set up the specified line.

This command is ignored in a GLOBAL ICAM environment.

If the TIP/30 ready message is configured, it is sent to the first terminal on the line.

UP xxxx

TIP/30 requests ICAM to UP the line associated with the terminal named xxxx (terminal names are often more readily known than line names).

This command is ignored in a GLOBAL ICAM environment.

If the TIP/30 ready message is configured, it is sent to the first terminal on the line.

WHOSON

List users that are currently logged on TIP/30.

9.4. Console Messages

Following is a list of TIP/30 console messages that may occur, along with a description of the specific situation and any suggested course of action. In the examples of message text, underscores represent data in the message that will be supplied by TIP/30.

Gaps in the numbering of messages are the result of either:

- the elimination of messages that are not relevant to the version of TIP/30 to which this documentation applies, or
- messages which are reserved for future use.

Messages from TIP/30 are prefixed by the string "TInnn" where "nnn" represents the internal message number or identifier.

TI001 TIP/30 Initialization Allinson-Ross Corporation

Informational message; TIP/30 initialization has been started.

TI002 __/__/__ - __:__:__ TIP/30 ready for _____

Informational message; TIP/30 initialization completed at the date and time specified. The SITE-ID is shown at the end of the message text.

TI004 Unknown operator request - consult manual

The operator has entered an unsolicited console command that is not recognized by TIP/30 — please refer to the section of the TIP/30 reference manual titled "OS/3 CONSOLE OPERATION" for the correct spelling and syntax of operator commands.

TI005 Error attaching schd TCB

An error occurred when the main TIP/30 task tried to "ATTACH" the subtask that performs all program scheduling. This error may occur if an insufficient number of TCBs was requested on the JOB card (number of TCBs must be greater than 3). TIP/30 will terminate when this error occurs.

TI006 Error attaching comm TCB

An error occurred when the main TIP/30 task tried to "ATTACH" the subtask that handles all network communications (via ICAM). This error usually indicates that an insufficient number of TCBs was specified on the JOB card.

TI007 Error opening ICAM (____/_____), code=_____

An error occurred when TIP/30 tried to open the communications network (via ICAM MOPEN macro call). This error usually indicates that the CCA name or network password is incorrect.

This error may also occur if any ICAM disk queue files are not allocated as a "contiguous" extent or are not large enough, or a dedicated ICAM network has more terminals defined than was generated into TIP/30 (see TERMS= parameter of TIPGEN macro). TIP/30 terminates when this error occurs.

The error codes are listed in *OS/3 System Messages* (UP-8076 Table A-1, Category "AA").

TI008 Unable to load control module _____

TIP/30 is unable to load the TCA module which is produced by the TIPGEN procedure. TIP/30 will terminate.

TI010 TIP/30 terminated

This message appears when TIP/30 has encountered an unrecoverable error.

TI012 Insufficient memory to execute TIP/30

The amount of paged memory that is available after TIP/30 has completed initialization is less than the minimum amount (specified by the MAXPROG= keyword in the TIP/30 generation parameters).

This error usually indicates that an insufficient amount of memory was specified on the JOB card for the TIP/30 execution. TIP/30 will terminate when this error occurs.

TI013 Fatal initialization error -- TIP/30 aborted

A previous error has occurred and the execution of TIP/30 cannot continue.

TI014 Invalid option selected at _____

An invalid option was selected in the run control statements provided in the imbedded data set in the TIP/30 JCL.

TI015 _____ is now available for online use

This message is confirmation that a closed file has been successfully reopened.

TI016 _____ has _____ users. Close held pending

Warning that the requested close of a file is being deferred until all current users of the file have relinquished control of the file.

While a file close operation is pending, no users may access the file (other than those users who were already granted access before the CLOSE was issued).

When all current users have finished with the file, the system will issue the following message (TI017) to confirm the file has (finally) been closed.

TI017 _____ is closed & not available for online use

This message is confirmation that an online file has been closed (and is now available for use by other jobs).

TI018 File _____ does not exist

This message is a warning that the requested file was not found (possible spelling error).

TI019 Unable to load PMDA : _____ _____

There is insufficient memory available to load the TIP/30 Post Mortem Dump Analysis (PMDA) program for the indicated user and transaction. TIP/30 continues running, but the indicated user will not receive a dump.

TI020 Fatal: TIP$SWAP file not assigned

The required TIP$SWAP file has not been assigned to the TIP/30 job. The job control stream should be updated to include the required file (LFD name is TIP$SWAP).

This message represents a fatal error.

TI021 Unable to load resident program: _____

The indicated program, which is named in a RESIDENT= job control statement, cannot be loaded. Processing continues, but the program will not be made resident.

TI024 TIP$SWAP - space lost due to fragmentation

TI025 TIP$SWAP - extending file by _____ blocks

The TIP/30 Swap file (TIP$SWAP) is being automatically extended by TIP/30. The number of blocks that are being added to the extent size is shown in the text of the message.

TI026 TIP$SWAP - I/O error - DM__

An unrecoverable hardware I/O error has occurred on the TIP$SWAP file. As a temporary solution, try moving the file to another location or disk drive (refer to supplied job stream TJ$SWAP).

The hardware customer engineer should by made aware of this error so that the hardware error log may be checked (ONUERL). TIP/30 will terminate when this error occurs.

TI027 Memory management error - job cancelled

An error has occurred which has corrupted TIP/30 internal memory management. Possibly a rogue user program has destroyed part of the TIP/30 region. TIP/30 terminates when this error occurs.

TI028 Program exception - PSW= _____ __ __ _____

An program check exception has occurred within TIP/30. If the error cannot be traced to rogue user programs, then the memory dump produced by this condition should be forwarded to Customer Support with as much supporting information as possible. TIP/30 terminates if this error occurs.

TI029 TIP/30 internal software failure

TIP/30 has detected an unrecoverable error (internal tables have been modified in error). If the error cannot be traced to rogue user programs, the memory dump produced by this condition should be forwarded to Customer Support with as much supporting information as possible. TIP/30 terminates if this error occurs.

TI030 Unable to attach user task

TIP/30 was unable to ATTACH a new user task.

TIP/30 terminates when this error condition occurs.

TI031 ICAM note on ____ - _____

TIP/30 was informed by ICAM that an error, as noted, occurred on the line indicated. This may be a warning; consult the appropriate ICAM error message description.

TI032 ICAM error _____, from ____ = _____

An ICAM error occurred when TIP/30 issued a request to the specified terminal. This may be a warning; consult the appropriate ICAM error message description.

TI033 Truncated input from = ____ program = _____

TIP/30 is reporting that input has been truncated. The input message was from the indicated terminal and was read by the indicated program.

This can occur (for example) if the terminal user pressed XMIT from a location BEYOND the designated cursor resting location in a TIP/30 screen format.

Essentially, more data than the program was expecting has arrived and has been appropriately truncated by ICAM.

This is a warning message and normally may be ignored.

TI034 TIP/30 version _____ - statistics

TI035 _____ at __:__:__

TI036 Msg in _____ average length: _____

TI037 Msg out _____ average length: _____

TI038 TIP$SWAP I/O: _____

TI039 _____ program loads for _____ requests

TI040 Average response time is _____ seconds

TI041 TIP/30 began execution at __:__:__ on __/__/__

This set of information and statistics is displayed as part of the TIP/30 shutdown procedure, in response to the STATS unsolicited command, or on a regular basis (at a frequency specified by the TIP/30 generation keyword STATS=).

TI042 _____ is not logged on

An invalid userid has been specified as a parameter in an unsolicited command to TIP/30.

TI043 ____ is an invalid terminal name

An invalid terminal name has been specified as a parameter in an unsolicited command to TIP/30.

TI046 Flag - State Flag - State Flag - State Flag - State

TI047 __ - ___ __ - ___ __ - ___ __ - ___

This heading line and detail lines are displayed in response to the FLAGS console operator command.

TI048 File #I/O's Output Pooled Users Open

TI049 _____ _____ _____ _____ ____ -

This heading line and details lines are displayed in response to the FILES operator command. The information displayed includes:

- the file name
- the total number of I/O requests issued for this file
- the number of I/O requests which were outputs
- the number of reads which were satisfied from the file's record pool
- the current number of users of the file
- whether the file is open.

TI050 TIP/30 requires at least 4 task control blocks

This message is displayed when TIP/30 determines that an insufficient number of task control blocks (TCBs) have been specified on the TIP/30 JOB card.

TIP/30 terminates if this error occurs.

TI051 TIP/30 will not use more than 50 tasks

This warning message is displayed if more than 50 task control blocks were requested on the TIP/30 JOB card.

TI052 No memory to logon ____

The memory manager in TIP/30 is unable to acquire enough free memory (from the pool of free memory specified via the TIP/30 generation parameter FREEM=) to LOGON the user at the named terminal.

TI053 Unable to load _____ for ____

TIP/30 was unable to load the specified program. The program does exist in the load library but an I/O error occurred during the load. The program should be re-linked and placed back into the TIP$LOD library. (The TIP$LOD library may be compromised!)

TI054 No memory to ____ for ____

The memory manager in TIP/30 is unable to acquire enough free memory to complete the indicated function for the user.

TI055 _____(___):_____

This is a message to the operator from the user and terminal indicated.

TI056 Key holding table full (___)_____

This message is displayed whenever TIP/30 detects that the TIP/30 key holding table is FULL.

Any requests to hold a record when the table is full, will receive a "record held" status.

This error should be brought to the attention of the systems programmer — who should consider making the key holding table larger and investigate why so many records were being held at the same time (a program may have been looping).

TI057 Command queue full - Command rejected

This message indicates that TIP/30 is still processing the previous unsolicited console command and that the most recent unsolicited command has therefore been ignored.

TI058 Unauthorized user attempted logon at ____

The operator is being informed that an unsuccessful attempt to logon has been detected at the indicated terminal.

TI059 DLL: load of ____, size = _____ bytes

The named terminal has been down line loaded with a module of the size indicated.

TI060 DLL: status from ____ = _____

The status of the down line load to the named terminal is given.

TI061 TIP$MCS I/O: _____ reads; _____ writes

Informational; the number of physical read and write operations to the TIP$MCS file.

TI062 TIP$CAT I/O: _____ reads; _____ writes

Informational; the number of physical read and write operations to the TIP$CAT file.

TI063 TIP$RNDM I/O: _____ reads; _____ writes

Informational; the number of I/Os to the TIP/30 Random File (TIP$RNDM).

TI064 All tasks were busy _____ times.

The number of times all tasks were busy. If this number is very high then the number of Task Control Blocks on the TIP/30 JOB card should be increased (minimum number of TCBs is 4; maximum is 50).

TI065 TIP$RNDM error DM__ for _____/_____/_____

An I/O error has occurred on this file, check the System Message handbook for description of the DMxx error.

TI066 _____(___):_____ Abended:_____

Reports the USER-ID(terminal):program name, which has aborted. If TIP/30 is able to determine information about the cause of the abnormal termination of the program, the reason will appear after the word "Abended".

TI076 _____(___):____ issued to _____ file

Informational: the userid at the indicated terminal has issued a file OPEN or CLOSE request (as indicated in the text of the message) for the file. This message makes it easier for the console operator to keep track of who has opened or closed files.

TI077 _____ is __% full

This message indicates the percentage that the indicated file is to being full. This is informational.

TI078 No background tables for ____

TIP/30 is unable to honour a request to start a background process from the specified terminal due to insufficient background table entries. The maximum number of concurrent background tasks is a TIPGEN option (BACK=) and may need to be modified.

TI079 _____ - _____ blocks in use

Informational; the number of blocks of the indicated file which have been used to date.

TI080 _____ file not assigned

The specified file was not found in the TIP/30 job control. The file is not used for a critical function — execution of TIP/30 will continue without the associated function. This condition should be brought to the attention of the system programmer.

TI097 Outstanding record locks - quick recovery recommended

This message is displayed when the TIP/30 system has been terminated with a STOP or CRASH command and records for HOLD=TR files were locked by transaction programs that were in progress.

The operator is advised that quick recovery must be performed before running any batch processing that uses the online files.

A reply is required — any reply will suffice; this is a very important message and we want be sure that it was NOT overlooked!

If you do not intend to restart TIP/30 before proceeding with batch processing, we recommend that you quickly restart TIP/30 with the run-time parameter WARM=ONLY — this will perform any necessary transaction roll back and then shutdown without allowing users to log on TIP/30.

TI098 TIP/30 has expired - contact A.R.C.

This message is reserved for future use and may be ignored.

TI099 _____(____):_____ rolled back _____

The USER-ID(TERM-ID) using the named program has caused a record roll back on the file named.

TI100 Gen parameters validated - Start _____? (Y/N)

The TIP/30 parameter processor has successfully completed the analysis of the TIP/30 generation parameters and needs to know whether the named generation job should be scheduled.

TI101 TIP/30 file formatter version _____

This informational message indicates the version of the TIP/30 file formatting program that is beginning execution.

TI102 _____ formatted, blocks = _____

The named file has been formatted to the specified number of blocks.

TI103 _____ successfully copied to _____

The LFD name first specified has been copied to the indicated LFD name.

TI104 About to initialize _____ OK? (Y/N)

The TIP/30 file initialization program is requesting confirmation to initialize the specified file (do you really want to do this?) Answering "N" will prevent it; "Y" will initialize the named file.

TI105 Enter password to initialize this file

The TIP/30 file initialization program is requesting password authorization before initializing the named file. Be absolutely sure you know what you are doing, or be prepared to test your site's recovery procedures.

This message represents the point of no return.

The password is the name of a popular teleprocessing monitor package that is available for Unisys computer users.

Enter the three character reply **TIP** to initialize the file; any other response will terminate the job STEP immediately. If you are not sure exactly what is happening, it might be prudent to cancel the job and investigate!

TI106 _____ will use _____ records

The named file has been initialized and has a record capacity as specified. The allocated number of blocks has been rounded down to the nearest prime number of blocks (this is to ensure that the key hashing algorithm will function properly).

TI107 _____ unused _____ records

The indicated file has been initialized. The specified number of records (blocks) represent the unused difference between the number of allocated blocks and the blocks that will actually be used. See also message number TI106.

TI108 Invalid user-id/password - Catalogue not processed

The operator has not specified the correct password for the initialization of the file; the function will not be performed.

TI109 Enter catalogue list options?

The batch catalogue listing program has found that there is insufficient data in the job control stream and is prompting the user at the OS/3 console for display options. Refer to batch job documentation for replies.

A console response that ends with a period (".") discontinues prompting.

TI110 I/O error on WORK1, DM__

The system scratch file (WORK1) has suffered a Data Management error as specified. See the OS/3 System Messages handbook for a description of the DMxx error.

TI111 No files assigned in recovery JCL

The TIP/30 Recovery module being executed has not been able to find required LFD information from the job control stream. Check the spelling of the LFDs with the file names in the TIP/30 generation.

TI112 Insufficient memory to load TCA

The TCA specified in the TIP/30 job stream will not fit in the memory space specified on the JOB card. More memory must be allocated for this job.

TI113 Recovery module (TCA) not found

The TCA specified does not exist, or did not specify journaling. Check generation options and recovery job JCL to resolve problem. The tcaname supplied to the TIP/30 batch recovery program must match the tcaname that was in use when TIP/30 created the journal or log file that is input to the batch recovery program.

TI114 TIP recovery in progress version _____

Informational; The TIP/30 Recovery module is being executed.

TI115 Syntax error - correct and try again

The parameters specified are incorrect for this job.

TI116 Missing semi-colon - Continue? (Y/N)

The recovery program has detected a syntax error in the control stream and is prompting the user for permission to continue in spite of the error.

This error may be a symptom of more serious errors or omissions in the recovery job control run-time parameters! The most prudent approach is to cancel the job and investigate why the run-time parameters do not include the required semicolon after each recovery statement.

TI117 _____ not journalled or DVC-LFD sequence missing

The JCL for the file named is missing or incorrectly spelled or the specified file name is not defined as a journaled file in the TIP/30 generation parameters that the recovery program has been told to use.

TI118 Too many errors - recovery terminated

The recovery program has detected too many syntax errors and is terminating abnormally.

TI119 _____ journal records read & _____ processed

Informational; statement of the number of journal records read and processed during forward file recovery.

TI120 _____ journal records read for backward recovery

Informational; statement of the number of journal records read during backward file recovery (roll back).

TI121 Beginning scan for quick recovery

Informational; recovery program is running and is beginning the scan to see if any quick roll back operations need to be performed.

TI122 TIP/30 file recovery completed

Informational; normal job termination of the batch off line recovery program.

TI123 _____ rcvd _____ on _____ (__/__/__ __:__)

Informational; number of records recovered (either forward or backward as indicated) for the named file as of the indicated date and time.

TI124 No dump. Previous dump not processed. _____ _____

An unsolicited console DUMP command was issued, but a previous dump has not been processed. Issue a DUMPF command to overwrite dump.

TI125 Retry,Cancel,Overwrite,None. _____ ____? (R/C/O/N)

This message indicates that TIP/30 is attempting to write a dump, but the dump file still has a dump from a previous time which has not been processed (printed or copied). The date/time is that of the existing dump in the file. This message is only seen if the dump analysis program (TB$DMP) has not completed processing the file.

The following replies are recognized:

R RETRY. If the dump job has not been run or has not completed, you may wish to wait for that job to complete before answering R. The dump file will be examined again to see if it has been processed.

C CANCEL. For some reason, you do not wish to overwrite the dump file but you want a dump. CANCEL will create an OS/3 job dump. TIP/30 will force an OS/3 job dump. There will be no normal termination processing.

O OVERWRITE. The dump in the file is not desired. You wish to have the current contents of TIP/30 memory written to the file, overwriting the older dump. TIP/30 will overwrite the dump and continue with normal termination processing.

N NONE. The current contents of TIP memory are not to be dumped. TIP will just continue with normal termination processing.

TI126 Error in resume all tasks. For online dump.

Internal error while trying to take an online dump.

TI127 Error in tpause all tasks. For online dump.

Internal error while trying to take an online dump.

TI128 Begin TIP/30 termination processing.

TIP/30 has begun normal termination processing.

TI129 Cannot schedule TIP dump job.

Attempt to schedule job to analyze TIP/30 dump failed.

TI130 Console request failed: no free memory!

Unsolicited operator command ignored due to no memory in free memory pool.

TI131 OS/3 job dump enabled for possible fatal error.

Informational. An OS/3 style job dump has been enabled if TIP/30 encounters a fatal error.

TI132 Begin write dump to TIP$DUMP file.

TIP/30 is beginning the process of writing dump information to the TIP$DUMP file.

TI133 Complete write to TIP$DUMP file.

The process of writing dump information to the TIP$DUMP file is complete.

TI134 Online dump not taken. Dump set for OS/3 job dump.

An online dump has not been taken. An OS/3 style job dump is taken instead.

TI135 Reset B4 file. No B4 records to rollback.

Informational. At termination, the TIP$B4 file has no records to roll back. The TIP$B4 file has been reset to an initialized state.

TI136 All console input (except STOP/EOJ) has been ignored

During TIP/30 initialization, operator input has been ignored (except STOP and EOJ commands).

TI137 Schedule TIP dump job.

Informational message on job log.

TI138 All tasks suspended during online dump.

Informational message on job log.

TI139 All tasks resumed. Online dump complete.

Informational message on job log.

TI140 TIP lower memory has been corrupted. Taking online dump.

Periodic validation of lower TIP/30 job memory indicates some program has overwritten lower memory. This is an indication of a program bug. This message is displayed only if CONTINUE=YES is specified in TIP/30 run-time job control options.

TI141 Abend during online dump.

Informational message on job log.

TI142 File _____ may have outstanding record locks

Informational. During normal termination processing (last step), the listed files may have outstanding record locks.

TI143 _____ file is not initialized

Fatal error. The specified file is not initialized or cannot be processed.

TI144 Unable to load file DTF for _____

The control blocks for the specified file cannot be created or accessed.

TI145 _____ ____% full, ____% records not at home position

Informational. The specified file is the indicated percent full. The percentage of records not at the "home" position is also shown. Records not at the "home" position are a natural result of the hashed file organization, but a large percentage not at home position may result in poor performance when accessing the file.

TI146 _____k of memory assigned to free memory pool

Informational.

TI147 _____k of memory available in pool, before page memory allocation

Informational.

TI148 _____k of free memory required from TIPGEN & JCS

Informational.

TI149 _____k allocated for paged memory (programs)

The specified amount of memory (in 1024 byte blocks) has been allocated by TIP/30 for transaction program execution.

TI150 Not enough memory. Function __, Amount _____

JCL error, region size inadequate. "Function" is memory request which failed. Problem is with this parameter or some prior parameter. "Amount" is amount of memory required for current function.

Function	Description
AF	AFT buffer pool
FT	File Table
MP	MAXPROG for Paged Memory
MT	MCS table
PM	Paged Memory Map
PT	Program table
RC	Reentrant Control Table
RE	Load RESIDENT or SUBPROG module
RQ	Free Memory Buffer Pool
TK	Task Control Table

TI151 Error in RESMOD TP$COMM list module: _____ _____

Internal TIP/30 error.

TI152 TIP/30 Version: _____ TCA module: _____

Informational. This message displays the current version of TIP/30 and the name of the TIP/30 generation parameter set (TCA) that is in use.

TI153 Replace RESMOD _____ is missing new name

Development & debugging use only.

TI154 Replace RESMOD _____ with _____

Informational message. The indicated module name that is referenced in the job control is no longer valid; the default module name (as shown) is assumed.

TI155 Warning: Replace RESMOD _____ is not defined

Development & debugging use only.

TI156 Start of buffer pool: _____

Development & debugging use only.

TI157 JCS _____ Parameter = _____ in error.

This message indicates that the noted run-time keyword or associated parameter is invalid.

TI158 Module _____ is not a TCA module name (1st JCS)

This message indicates that the TCA name found in the run-time options is not valid. A possible cause is a missing TCA name — the TCA name must be supplied on the first card image of the imbedded data set in the TIP/30 job control stream.

TI159 JCS unrecognized keyword: _____

The specified keyword was specified in the run-time options for TIP/30, but is not recognized as a valid keyword. Check the spelling of the keyword.

TI160 Warning: No floating point defined in OS/3 gen.

The OS/3 operating system was generated without floating point arithmetic support (FLOAT=YES).

TI161 Warning: No CDM support defined in OS/3 gen.

The OS/3 operating system was generated without support for Consolidate Data Management (CDM). TIP/30 requires CDM support in OS/3.

TI162 Warning: No FILELOCK=SHARE defined in OS/3 gen

The OS/3 operating system was generated without the proper level of file locking. TIP/30 requires FILELOCK=SHARE.

TI163 Fatal error: Only OS/3 release 8.1 or better supported

The OS/3 operating system must be at least level 8.1 or later to support TIP/30.

TI164 Warning: Remove DMNAME=xxxxx from JCS; ignored.

This message is a warning. An obsolete specification for the DMname= keyword was detected in the run-time options and is ignored. To avoid confusion, the obsolete specification should be removed from the TIP/30 job control parameters.

TI165 _____ JCS parameter error; partially processed.

JCS error. The named JCS parameter has been processed and stored up to the point the error was detected.

TI166 _____ parameter error: File _____ not found.

A file name specified as a run-time parameter is not found.

TI167 Termination of TIP/30, taking dump.

TIP/30 terminating due to some event. Further console key ins are ignored.

TI168 CANCEL TIP/30 or Ignore error? (C/I)

JCS error. One or more JCS errors have been detected. A reply of "C" terminates TIP/30 immediately after processing the rest of the JCS. A reply of "I" ignores the error and continues with TIP/30 initialization.

TI169 _____ Program records in CAT file

Informational. The number of "Program" definition records were detected in the TIP$CAT file during initialization.

TI170 _____ File records in CAT file

Informational. The number of "File" definition records were detected in the TIP$CAT file during initialization.

TI171 Internal error: invalid parameter list to TR$INITB

Internal TIP/30 error.

TI172 Internal error: TASK open

Internal TIP/30 error.

TI173 TIP$DUMP FILE ERROR CODE: DM__

An error has been reported to TIP/30 (by OS/3 Data Management) for the TIP$DUMP file. The DMxx error code is shown.

TI174 All files are closed

Informational message. TIP/30 has successfully closed all files.

TI175 Boot unable to load overlay: _____

Internal TIP/30 error.

TI176 TB$TIP Link Edit _____ _____

Informational. The date and time that TIP/30 was linked is shown.

TI177 ICAM note on ____ - line down, disconnected

ICAM returned the specified status to TIP/30.

TI178 ICAM note on ____ - line down, not disconnected

ICAM returned the specified status to TIP/30.

TI179 ICAM note on ____ - disk error during output

ICAM returned the specified status to TIP/30.

TI180 ICAM note on ____ - line is now up

ICAM returned the specified status to TIP/30.

TI181 ICAM note on ____ - line already active

ICAM returned the specified status to TIP/30.

TI182 ICAM note on ____ - unidentified terminal

ICAM returned the specified status to TIP/30.

TI183 ICAM note on ____ - ARP buffers depleted

ICAM returned the specified status to TIP/30.

TI184 ICAM note on ____ - network buffers depleted

ICAM returned the specified status to TIP/30.

TI190 ICAM error _____, from ____ = Boundary error; table alignment

ICAM returned the specified status to TIP/30.

TI191 ICAM error _____, from ____ = Limits error; address not in job region

ICAM returned the specified status to TIP/30.

TI192 ICAM error _____, from ____ = Decode error; invalid function

ICAM returned the specified status to TIP/30.

TI193 ICAM error _____, from ____ = No message available

ICAM returned the specified status to TIP/30.

TI194 ICAM error _____, from ____ = Missing or invalid destination

ICAM returned the specified status to TIP/30.

TI195 ICAM error _____, from ____ = No network buffer

ICAM returned the specified status to TIP/30.

TI196 ICAM error _____, from ____ = Disk error

ICAM returned the specified status to TIP/30.

TI197 ICAM error _____, from ____ = Invalid length

ICAM returned the specified status to TIP/30.

TI199 Completed statistics journal

Informational message. Statistical records written to TIP$JRN or TIP$LOG file.

TI201 User ____, TID ____, error in access _____, TRID _____

The user at the terminal name shown is attempting to access a product that has not been correctly installed. The product name and transaction name is also shown.

TI202 User ____ at ____ is not authorized to access _____ via _____

The user at the terminal name shown does not have proper TIP/30 Catalogue security specified to access the item.

TI204 _____ is an invalid destination name

The specified character string is not recognized as a valid destination name.

TI205 Product _____ in DEMO mode, Expire __/___

The identified product has expired; users accessing the product are using it in demo mode. After the date shown the product cannot be accessed.

TI206 Product _____ will expire __/___

The identified product will expire on the date shown. This message is displayed when the product is within 30 days of expiry.

TI207 JCS _____ reference to _____ has changed, default _____ used.

Informational message. The indicated module name that is referenced in the job control is no longer valid; the default module name (as shown) is assumed. The job control parameters should be modified to remove the obsolete name reference.

Section 10
TIP/30 Installation Guide

This section is the installation guide for TIP/30 (Transaction Interface Processor), a software product developed by Allinson-Ross Corporation.

10.0.1. TIP/30 Pre-installation Setup

Before proceeding with the installation of TIP/30, there are a few procedures which should be followed to make the installation as smooth as possible. This installation guide outlines these procedures.

> **WARNING**
>
> It is very important that you read all of this installation guide before attempting to perform any of the steps that are described.

10.0.2. OS/3 Supervisor Generation

The following items should be checked in the generation parameters for the OS/3 supervisor. Some keywords apply only to older releases of OS/3 and may not be applicable to your release.

SYMBPRI=5

OS/3 symbionts (run processor, output writer, etc) and batch jobs should not execute at priority levels that conflict with the priority levels used by TIP/30.

PRIORITY=5+#job classes

There should be enough priority levels to run TIP/30 (at levels 1 through 4), symbionts (at level 5), and each "class" of batch job at a different priority level (if required). We recommend that you not have excessive priority levels (no more than needed).

Assuming that TIP/30 is executed at priority level 1, TIP/30 makes use of priority levels 1 through 4 and may make use of priority levels beyond 4 depending on the TIP/30 generation keyword parameter PRIority=.

If you have batch jobs which specify an explicit execution priority on job control EXEC statements, it is wise to review those job control streams to make sure that batch jobs do not run at a priority that conflicts with TIP/30.

A reasonable rule of thumb is to run I/O bound jobs at higher priority (numerically lower value) than CPU bound jobs.

ISINTPRI=6

If you intend to use OS/3 interactive services, add one more level to the PRIORITY keyword and set this keyword to 6. This setting will help prevent interactive services users from monopolizing the system.

RESHARE=

TIP/30 does not require any resident shared code. However, you should make resident those shared code modules that are used by your system (batch and online). The OS/3 job SCLIST may be run to produce a listing of OS/3 shared code modules. The comment that appears next to each module might help you determine whether or not that module is important enough to be made resident via the RESHARE= specification.

RESHARE=DM$MIAQ (for OS/3 release 11 and later) is *suggested* if TIP/30 is to support multiple concurrent sequential readers of MIRAM files.

RESMOD=SM$TASK,SM$ASCKE,SM$LOCK,SM$STXIT,SM$GTPUT

These are the only transients that are recommended or required to be made resident for use by TIP/30. Other transients may be declared resident according to the requirements of other systems.

Note: *SM$ASCKE must be made resident. If this resmod is not specified, the system may enter a permanent RUN state when TIP/30 is executed.*

COMM=#CAs,#ports

This parameter is required to support the communications hardware.

SPOOLING=INPUT

This is not mandatory, however we recommend this since it will allow you to use facilities in TIP/30 to run batch jobs without writing the job control statements to a library.

If the type of spooling is changed, the first use of the new supervisor requires initialization of the spool file. Be sure to clear out the spool file before IPLing.

FILELOCK=

FILELOCK=SHARE is recommended for use with TIP/30.

The specification FILELOCK=YES can lead to library files being compromised and is not recommended. A warning message is issued at TIP/30 startup if FILELOCK=SHARE is not specified.

The specification FILELOCK=NO is not supported.

JOBACCT=YES

This is not mandatory but including job accounting allows various TIP/30 utilities to monitor the performance of the OS/3 system.

SELACC=34

If you are using selector type disk subsystems (8430/8433) specify SELACC=34 on each selector disk declaration for maximum performance.

FLOATPT=YES

This is required if you intend to use TIP/30 BASIC, OS/3 FORTRAN, or COMP-1 or COMP-2 (floating point) fields in online COBOL programs.

This is an example of an OS/3 Supervisor generation for use with TIP/30 on a SYS/80 Model 6.

```
SUPGEN
          SUPVRNAM=TIPSUP
          COMM=1,8
          FILELOCK=SHARE
          FLOATPT=YES
          ISLOCAPID=ISSN
          ISINTPRI=6
          ISNETNAME=TIPC
          JOBACCT=YES
          JOBSLOTS=7
          PRIORITY=12
          RESMOD=SM$TASK,SM$ASCKE,SM$LOCK,SM$STXIT,SM$GTPUT
          ROLLOUT=YES
          RUNVSN=SYSRUN
          SPOOLBUFR=16
          SPOOLOWBUFR=8
          SPOOLCYL=100
          SPOOLHDR=NO
          SPOOLING=INPUT
          SPOOLMAP=75
          SPOOLVSN=SYSSPL
          SYMBPRI=5
          SYSLOG=YES
          TIMER=MAX
          TRANS=15
END
I/OGEN
PRINTER    TYPE=0770 LCB=OWNLC1 PRINTPOS=132
           CHARSET=48 VFB=STAND1 ATTNRE=YES
READER     ATTNRE=YES
DISC       TYPE=8418 ADDR=00-03
DISC       TYPE=8430 ADDR=00-05 CHAN=4 SELACC=34
TAPE       TYPE=6    ADDR=00-03
PRINTER    VIRTUAL=5
END
```

This is an example of OS/3 Supervisor generation parameters for use with TIP/30 (on a SYS/80 Model 20):

```
SUPGEN
          SUPVRNAM=SY$STD
          CACHESEGSIZE=12
          CHAN=13
          COMM=8
          COMM1=NO
          CONALARM=NO
          CONPRINT=YES
          CONSOLOG=MAX
          DAYCHANGE=YES
          DDPSC=NO
          DLOADBUFR=4096
          DLOADTABLE=8
          DMGTMODE=MIXED
          DMRECV=YES
          ERRLOGBUF=6
          EXECPRI=12
          EXPREGION=4096
          FILELOCK=SHARE
          FLOATPT=YES
          IGNORESFT=YES
          IORB=50
          ISBATCHLMT=4
          ISINTLMT=10
          ISINTPRI=7
          ISLOCAPID=INTS
          ISLOGONSC=NO
          ISNETNAME=INET
          JCREADWKS=NO
          JOBACCT=YES
          JOBQUEREC=HOLD
          JOBSLOTS=18
          MAXTIME=0
          MAXTYPE=NONE
          PASSWORD=NO
          PRIORITY=15
          RECOVERDS=YES
          RESHARE=DD$T1110,DM$CFM00,DM$FSP00,DM$MIAQ0,D3$M1110
          RESHARE=PR$IOE00
          RESMGT=NO
          RESMOD=SM$ASCKE,SM$ATCH,SM$GTPUT,SM$LOCK,SM$LOD
          RESMOD=SM$STXIT,SM$TASK
          RETAINLOG=NO
          ROLLOUT=YES
          RUNVSN=SYSRUN
          SAM=YES
          SCDINDEX=YES
          SHAREDMGT=32
```

```
            SPOOLBUFR=16
            SPOOLBURST=NO
            SPOOLCOMP=YES
            SPOOLCYL=1
            SPOOLVSN=SYSRUN
            SPOOLCYL2=80
            SPOOLVSN2=SYSSPL
            SPOOLFARSI=NO
            SPOOLHDR=NO
            SPOOLICAM=C1
            SPOOLING=REMOTE
            SPOOLMAP=300
            SPOOLMAXLINE=20
            SPOOLMODE=PRI
            SPOOLNOINPUT=NO
            SPOOLOWBUFR=16
            SPOOLPRT=ALL
            SPOOLRECV=CLOSED
            SPOOLTEST=YES
            SPOOLUPDATE=YES
            SYMBPRI=6
            SYSLOG=NO
            SYSTEMDATE=YMD
            TAPEBLKNO=NO
            TRANS=8
            TRNWKAREA=64
            UNATCONSOLE=0
            VOLTABLE=YES
            VVAVR=YES
END
I/OGEN
DISC        CHAN=1
            ADDR=A0-A7
            CACHE=YES
            COADDR=A0-A7
            COCHAN=2
            TYPE=8494
DISC        CHAN=1
            ADDR=90-97
            CACHE=YES
            TYPE=8470
DISC        CHAN=2
            ADDR=B0-B2
            CACHE=YES
            TYPE=8417
DISC        CHAN=2
            ADDR=B3
            CACHE=YES
            TYPE=8419
TAPE        CHAN=3
```

```
          ADDR=80-81
          DENSITY=DUAL
          MODE=C3
          TRANSLAT=NO
          TYPE=28
PRINTER   CHAN=3
          ADDR=E0
          ATTNRE=YES
          CHARSET=STD
          LCB=OWNLC1
          PRINTPOS=160
          REMOTE=NO
          TYPE=0770
          VFB=STAND1
DISKETTE  CHAN=C
          ADDR=20
          AUTOLOAD=NO
          TYPE=8420
WORKSTATION CHAN=C
          ADDR=11,12
          DESPACE=YES
          TYPE=3560
REMWORKSTATION
          AMOUNT=10
          SCRENMEM=1
PRINTER   VIRTUAL=24
PUNCH     VIRTUAL=3
READER    VIRTUAL=3
END
```

10.0.3. OS/3 ICAM Generation

The following items should be checked in the OS/3 ICAM generation parameters:

CCA statement

The CCA statement controls the type of ICAM that is to be generated (a dedicated or stand-alone ICAM, or a GLOBAL ICAM). Although TIP/30 will function correctly in either environment, a GLOBAL ICAM offers a more flexible environment and is recommended.

TYPE=(TCI)

A dedicated ICAM must be generated as a TCI network for TIP/30.

TYPE=(GBL,,node)

If you intend to use a global ICAM then specify the GBL parameter; it will also be necessary to define a LOCAP for use by TIP/30.

The node name can be up to four characters that identifies your site in a multi-node ICAM network. This name is not important for TIP/30 use but must not conflict with any TERM or LOCAP names that are used.

FEATURES=(OPCOM,OUTDELV)

These are the only ICAM features which are required by TIP/30. Other features may be included if needed for other communication applications.

PASSWORD=

The password feature of ICAM is not required by TIP/30. If a password is assigned, it, along with the NETWORK name, must be supplied as run-time parameters to TIP/30.

BUFFERS statement

As a general guide, choose a buffer size which is large enough to hold the average output message for your network and mix of applications. It is often better to have a large number of medium size buffers rather than a small number of very large buffers. ICAM chains buffers together to hold a large output (or input) message, but the buffer size should be chosen according to the most common traffic conditions.

The maximum size of a single message is specified in sub-parameter 2 of the DEVICE= keyword parameter on the LINE statement.

STAT=YES

The keyword STAT= (note the spelling) on the BUFFER statement will allow use of the TIP/30 utility transaction CCA to display ICAM buffer and ARP utilization statistics.

The overhead involved in maintaining these statistics is negligible. We recommend that this option be specified on the BUFFER statement.

LINE statement

LBL=

The line buffer length for local workstations must be large enough to accommodate the largest output message that is anticipated, otherwise output messages may be truncated by ICAM. For local workstations, LBL=900 is usually sufficient.

QUEUES

The LOW queue is used by TIP/30 for the TIP/30 ready message and for auxiliary I/O (eg: printing).

The MEDIUM queue is used for all normal terminal traffic.

The HIGH queue is used for unsolicited messages to the terminal.

If remote batch devices are used with TIP/30, all queues for those devices must be disk queues.

Note: if disk queueing is specified, ICAM may use the disk queue file to buffer messages. This overhead can be avoided by using main memory queues. If sufficient main storage is available, assign the queues for *interactive* terminals to "MAIN".

RECONECT=

This keyword (note the spelling) is applicable only to switched lines (dialed lines). **DO NOT** use this keyword on any other type of line!

STATS=YES

The keyword STATS= (note the spelling) on LINE statements will allow use of the TIP/30 utility transaction CCA to display line and terminal statistics.

The overhead involved in maintaining these statistics is negligible. We recommend that this option be specified on all LINE statements.

TCIDTF statement

MSGSIZE=n

The recommended value is 3584. Specify the size of the largest message that can be output. Too small a number may result in output message truncation.

10.0.4. Example ICAM Generations

This is an example of the generation parameters for a dedicated (non-Global) ICAM.

```
COMMCT
TIPC      CCA        TYPE=(TCI),FEATURES=(OPCOM,OUTDELV),PASSWORD=PWD
          BUFFERS    30,64,5,ARP=40,STAT=YES
L311      LINE       DEVICE=(LWS),LBL=900,STATS=YES
T311      TERM       ADDR=(311),FEATURES=(LWS),                          X
                     LOW=DQFILE1,MEDIUM=MAIN,HIGH=MAIN
L312      LINE       DEVICE=(LWS),LBL=900,STATS=YES
T312      TERM       ADDR=(312),FEATURES=(LWS),                          X
                     LOW=DQFILE1,MEDIUM=MAIN,HIGH=MAIN
L313      LINE       DEVICE=(LWS),LBL=900,STATS=YES
T313      TERM       ADDR=(313),FEATURES=(LWS),                          X
                     LOW=DQFILE1,MEDIUM=MAIN,HIGH=MAIN
LN08      LINE       DEVICE=(UNISCOPE),TYPE=(9600,SYNC),ID=08,STATS=YES
ARC1      TERM       ADDR=(21,51),FEATURES=(U400,1920),AUX1=(COP,73),    X
                     LOW=DQFILE1,MEDIUM=MAIN,HIGH=MAIN
ARC2      TERM       ADDR=(21,52),FEATURES=(U400,1920),AUX1=(COP,73),    X
                     LOW=DQFILE1,MEDIUM=MAIN,HIGH=MAIN
SBP1      TERM       ADDR=(22,51),FEATURES=(U40,1920),AUX1=(COP,73),     X
                     LOW=DQFILE2,MEDIUM=DQFILE2,HIGH=DQFILE2
LN09      LINE       DEVICE=(UNISCOPE),TYPE=(2400,SYNC,SWCH,UNAT),ID=09, X
                     RECONECT=YES
TRM1      TERM       ADDR=(22,51),FEATURES=(U200,1920),AUX1=(COP,73),    X
                     LOW=DQFILE1,MEDIUM=MAIN,HIGH=MAIN
DQFILE1   DISCFILE   FILEDIV=10
DQFILE2   DISCFILE   FILEDIV=10
TCIDTF    DISCFILE   MSGSIZE=3584
          ENDCCA
          MCP        MCPNAME=M5
                     CACH=(08,9600,SYNC)
                     CACH=(09,2400,SWITCHED,SYNC)
END
```

This is an example of a global ICAM.

```
COMMCT
NET1     CCA       TYPE=(GBL,,ARC),FEATURES=(OPCOM,OUTDELV),GAWAKE=YES
*
         BUFFERS   30,128,5,ARP=50,STAT=YES
*
*        TIP/30 TCI LOCAP
*
TIPC     LOCAP     TYPE=(TCI)
*
*        INTERACTIVE SERVICES LOCAP
*
ISSN     LOCAP     TYPE=(DMI),IAS=(YES,OFF),MT=YES,                    X
                   LOW=DQFILE,MEDIUM=MAIN,HIGH=MAIN
*
L311     LINE      DEVICE=(LWS),LBL=900,STATS=YES
T311     TERM      ADDR=(311),FEATURES=(LWS),TCTUPD=YES,               X
                   LOW=DQFILE,MEDIUM=MAIN,HIGH=MAIN,INPUT=(YES)
L312     LINE      DEVICE=(LWS),LBL=900,STATS=YES
T312     TERM      ADDR=(312),FEATURES=(LWS),TCTUPD=YES,               X
                   LOW=DQFILE,MEDIUM=MAIN,HIGH=MAIN,INPUT=(YES)
LN08     LINE      DEVICE=(UNISCOPE),TYPE=(9600,SYNC),ID=08,STATS=YES, X
                   LBL=64
ARC1     TERM      ADDR=(21,51),FEATURES=(U400,1920),TCTUPD=YES,       X
                   LOW=DQFILE,MEDIUM=MAIN,HIGH=MAIN,INPUT=(YES),       X
                   AUX1=(COP,73)
ARC2     TERM      ADDR=(21,52),FEATURES=(U400,1920),TCTUPD=YES,       X
                   LOW=DQFILE,MEDIUM=MAIN,HIGH=MAIN,INPUT=(YES),       X
                   AUX1=(COP,73)
*
*        FORCE SOME TERMINALS ONTO TIP/30 LOCAP
*
         SESSION   EU1=(ARC1),EU2=(TIPC)
         SESSION   EU1=(ARC2),EU2=(TIPC)
*
DQFILE   DISCFILE  FILEDIV=5
TCIFLE   DISCFILE  MSGSIZE=3584
         ENDCCA
         MCP
                MCPNAME=C3
                CACH=(08,9600,SYNC)
END
```

10.1. Installation — PART I

The TIP/30 system is released on tape or diskette in LIBS format. For the System 80 Model 7E, the TIP/30 release library is named YTIP and is pre-loaded on the RES volume.

The installation of TIP/30 includes the creation of a number of files that are used by the TIP/30 system. The following tables show the files that may be created (some are optional files). Included in the tables is the default allocation for the files and the approximate size of each file (in cylinders) on several popular disk types.

The installation procedure allocates these files by using block allocation (BLK) or track block allocation (TBLK) to permit the job control to run regardless of the actual destination disk type.

The following table gives approximate *cylinder* allocations for the default sizes of the various files used by TIP/30. LBL names enclosed in parentheses are default LBL names that may be overridden during the installation procedure; the string "id" represents the selected system identifier.

Table 10-18. Disk Space Requirements

LFDname	LBLname	Default BLKs (size)	8417	8419	8433	8470	8494
SYSGEN	(TIP.id.GEN)	1400 (256)	2	4	3	1	1
TIP	(YTIP)	21000 (256)	25	60	34	7	12
TIP$BAK	TIP.id.BAK	16000 (512)	39	92	52	11	18
TIP$B4	TIP.id.B4	1400 (256)	2	4	3	1	1
TIP$CAT	TIP.id.CAT	4096 (256)	5	12	7	2	3
TIP$DUMP	TIP.id.DUMP	15000 (256)	18	43	24	5	9
TIP$HST	TIP.id.HST	8000 (512)	20	46	26	6	9
TIP$JCS	(JT)	500 (256)	1	2	1	1	1
TIP$JRN	TIP.id.JRN	8000 (512)	20	46	26	6	9
TIP$LOD	TIP.id.LOD	10000 (256)	12	29	16	4	6
TIP$MCS	TIP.id.MCS	1400 (2560)	17	40	23	5	8
TIP$MSG	TIP.id.MSG	500 (256)	1	2	1	1	1
TIP$RNDM	TIP.id.RNDM	10000 (512)	24	58	32	7	12
TIP$SWAP	TIP.id.SWAP	12000 (256)	15	35	20	4	7
TIP$TOM	TIP.id.TOM	200 (4096)	4	10	6	2	2
TIP$TSP	TIP.id.TSP	200 (256)	1	1	1	1	1
Totals			206	484	275	64	100

Table 10-19. TIP/30 Disk File Allocation

LFD	Volume	Blocks	Type	Req'd?	Default BLKs (size)
SYSGEN			SATlib	Yes	1400 (256)
TIP			SATlib	Yes	21000 (256)
TIP$BAK			Miram	Opt	16000 (512)
TIP$B4			SAT	Opt	1400 (256)
TIP$CAT			SAT	Yes	4096 (256)
TIP$DUMP			SAT	Yes	15000 (256)
TIP$HST			SAT	Opt	8000 (512)
TIP$JCS			SATlib	Yes	500 (256)
TIP$JRN			SAT	Opt	8000 (512)
TIP$LOD			SATlib	Yes	10000 (256)
TIP$MCS			SAT	Yes	1400 (2560)
TIP$MSG			Miram	Yes	500 (256)
TIP$RNDM			SAT	Yes	10000 (512)
TIP$SWAP			SAT	Yes	12000 (256)
TIP$TOM			SAT	Opt	200 (4096)
TIP$TSP			Miram	Opt	200 (256)

10.1.1. Step 1 — Getting Started

If you are using OS/3 Release 13 or later, this step can be skipped — proceed directly to "10.1.2. Step 2A — Quick Install" on page 10-16.

Before executing the installation job, three modules must be copied from the TIP/30 release media to the appropriate OS/3 libraries. To do this, enter the appropriate Interactive Services commands from the choices shown below:

① Copy modules from release TAPE:

```
COPY TJ$INS,TIP30$VER$0400,TIP$30,DEV=xxx      TO TJ$INS,$Y$JCS

COPY TJ$LOAD,TIP30$VER$0400,TIP$30,DEV=xxx     TO TJ$LOAD,$Y$JCS

COPY SMCUTP00,TIP30$VER$0400,TIP$30,L,DEV=xxx TO SMCUTP00,$Y$LOD,,L
```

Note: *The DEV=xxx parameter specifies the device address of the tape drive where the TIP/30 release tape is mounted. Specify the correct device address for your configuration.*

② Copy modules from release DISKETTE:

```
COPY TJ$INS,DATA,TIP$30       TO  TJ$INS,$Y$JCS

COPY TJ$LOAD,DATA,TIP$30      TO  TJ$LOAD,$Y$JCS

COPY SMCUTP00,DATA,TIP$30,L   TO  SMCUTP00,$Y$LOD,,L
```

10.1.2. Step 2A — Quick Install

The job TJ$INS is provided to guide you through the installation of TIP/30. The job presents a menu of possible activities including the initial installation of a TIP/30 system.

To execute the TJ$INS job, issue the following command from a workstation or the system console (a workstation is recommended because running this job from the console while other jobs are executing can make reading the prompts difficult).

```
RV TJ$INS:$Y$JCS,,A=xxxx
```

Where:

A=xxxx This job global can be used to specify the account number for the job. The default job account number is "TIP".

The TJ$INS job may be restarted by using the global RESTART=name. The name specified is the job name where the restart is to begin. For example, if an unrecoverable error occurred during the execution of the job TJ$CAT, the TJ$INS job can be rerun this way:

```
▶RV TJ$INS,,A=xxxx,RESTART=TJ$CAT
```

If it is necessary to abandon the installation procedure and start over from scratch, first run the job TJ$SCRTP to erase all files for a specified system identifier, then restart the TJ$INS job from the beginning. See the description of the job TJ$SCRTP.

The TJ$SCRTP job may optionally scratch the job control library associated with the TIP/30 system that is being scratched. The TIP release library associated with the TIP/30 system being scratched is not scratched by the TJ$SCRTP job — this action eliminates the need to reload the TIP library when the installation procedure is started over.

When the installation job begins processing, the main menu appears:

```
 11 LOGON003 IS19 LOGON ACCEPTED AT 13:29:26  ON  89/12/11, REV 13.0.0S4
▶rv tj$ins
 12 RV2039   TJ$INS: TIP/30 INSTALLATION/MAINTENANCE JOB
 13 RV2039   R03 RUN PROCESSOR SUCCESSFULLY PROCESSED TJ$INS
 14 TJ$INS   JC01  JOB TJ$INS   EXECUTING JOB STEP SMCUTP00 #001 14:09:28
 15 TJ$INS   TPS TRANSACTION PLATFORM SYSTEM
 16 TJ$INS   ------------------------------------------------
 17 TJ$INS   ENTRY LEVEL TIP/30 INSTALLATION/MAINTENANCE UTILITY
 18 TJ$INS   COMMAND DESCRIPTION             ==> INSTALL MENU <==
 19 TJ$INS   END     END INSTALLATION PROGRAM
 20 TJ$INS   1       QUICK INSTALLATION OF TIP/30
 21 TJ$INS   2       STANDARD INSTALLATION OF TIP/30
 22 TJ$INS   3       UPDATE TIP/30 RELEASE 3.2 TO 4.0
 23 TJ$INS   8       DISPLAY "@" COMMAND LIST
 24 TJ$INS   9       HELP - SUMMARY OF OPTIONS FROM THIS MENU
 25?TJ$INS   ENTER COMMAND
```

Items ⑧ and ⑨ provide help information and are not described here.

This section describes selection ① from the main installation menu, "Quick Installation of TIP/30". A "Quick installation" can be selected only if all of the following conditions apply:

- You accept the default size allocations for the various TIP/30 system files that are required
- The TIP/30 system files are to be installed on no more than 2 disk volumes
- You do not (initially) require any of the optional TIP/30 files such as the TIP/30 Journal file, TIP/30 History file or TIP/30 Log file.
- You accept the default assignment of the TIP/30 system identifier (this identifier is used to construct unique file LBL names). The system identifier defaults to the LOCAP name (if a GLOBAL ICAM is used) or the NETWORK name (if a dedicated ICAM is used).

If all of the above conditions are not met, you can not select the "Quick Installation". You must select item ② and proceed to "Step 2B — Detailed Install" on page 10-22.

When the "quick installation" option is chosen, the TJ$INS program prompts you for a few crucial items of information. The prompts are shown below (as an illustration) and are described in detail following the example.

```
▶25 1
 26 TJ$INS   FUNCTION: ENTRY LEVEL TIP/30 (TPS) INSTALLATION
 27 TJ$INS   THIS FUNCTION WILL RESULT IN A NEW TPS SYSTEM
 28 TJ$INS   BEING INSTALLED.
 29?TJ$INS   ENTER LBL NAME OF TIP/30 RELEASE LIBRARY ($Y$TIP)
```

ENTER LBL NAME OF TIP RELEASE LIBRARY (DEFAULT IS YTIP)

This prompt is requesting the desired LBL name of the TIP/30 release library. The default LBL name is YTIP. System 80 Model 7E users have the TIP/30 release library pre-loaded on the sysres volume with the LBL name "YTIP" and should take the default response to this prompt.

If you have received TIP/30 on a release tape or diskettes, you must enter the LBL name you wish to use for the TIP/30 release library.

The recommended LBL name is constructed like this:

```
TIP.id.RELEASE
```

Where "id" represents the system identifier for this TIP/30 system (the identifier defaults to the LOCAP name for a GLOBAL ICAM, or the ICAM NETWORK name if a dedicated ICAM is used).

```
 30 TJ$INS   DOES TIP/30 RELEASE LIBRARY ALREADY EXIST? (Y/N)
▶30 y
 31 TJ$INS   RE-LOAD TIP RELEASE LIBRARY (NOW)? Y/N
▶31 n
 32 TJ$INS   ENTER PRIMARY DEFAULT VOLUME NAME FOR NEW TIP/30 FILES
▶32 a941a6
 33?TJ$INS   ENTER SECONDARY DEFAULT VSN FOR NEW TIP/30 FILES (DEFAULT:A941A6)
▶33
```

DOES THE RELEASE LIBRARY ALREADY EXIST? (Y/N)

This prompt is asking whether or not the TIP/30 release library already exists. System 80 Model 7E users should reply "Y" (because the library YTIP on RES is shipped on the RES volume for that hardware system). Other users reply "N" to have the TJ$INS job automatically load the release media or reply "Y" to use a previously loaded release library.

RE-LOAD TIP RELEASE LIBRARY (NOW) Y/N

This prompt is asking whether or not you want to reload the contents of the TIP/30 release library right now. In most cases, the response is "N", but the option exists to load tape or diskettes to an existing TIP/30 release library (in case you have received a more recent version on tape or diskettes).

If a "Y" response is given to reload the TIP release library, the following prompt is issued:

```
MEDIA OF TIP/30 RELEASE LIB IS TAPE? (Y/N)
```

Reply "Y" to this prompt to reload the TIP release library from a tape, otherwise reply "N" to reload the library from diskettes.

ENTER PRIMARY DEFAULT VOLUME NAME FOR NEW TIP/30 FILES

This prompt is requesting the volume serial number of a disk that is to be primary disk of choice for allocating TIP/30 system files. Reply with the volume serial number of a disk drive that is currently mounted and will always be mounted when TIP/30 is to run.

This prompt requires a response.

ENTER SECONDARY DEFAULT VSN FOR NEW TIP/30 FILES
(DEFAULT: xxxxxxx)

This prompt is requesting the volume serial number of a disk that is to be an alternate disk of choice for allocating TIP/30 system files. Reply with the volume serial number of a disk drive that is currently mounted and will always be mounted when TIP/30 is to run.

If a null response is given to this prompt, the TJ$INS job assumes that all of the TIP/30 system files are to be allocated on the primary disk drive (shown as the default value in the prompt).

The TJ$INS job allocates the TIP/30 system files so that the files are intelligently allocated on the one or two disk volumes available.

```
 34 TJ$INS    ENTER ICAM NETWORK NAME
▶34 net2
 35 TJ$INS    ENTER ICAM NETWORK PASSWORD (NULL,1-8 CHARS)
▶35
 36 TJ$INS    ENTER ICAM LOCAP NAME (4 CHARS,NULL=DEDICATED)
▶36 tst2
 37 TJ$INS    ENTER LBL NAME FOR JCS LIBRARY (DEFAULT=JT)
▶37
 38 TJ$INS    ENTER VSN FOR FILE JT (DEFAULT:A941A6)
▶38
```

ENTER ICAM NETWORK NAME

This prompt is requesting the name of the ICAM network that is to be used in conjunction with TIP/30.

This prompt requires a response.

Enter the network name as it appears in the ICAM generation for your site (this is the name on the CCA statement in the ICAM generation parameters).

ENTER ICAM NETWORK PASSWORD (NULL, 1-8 CHARS)

If your ICAM network was generated with a "password", supply that password as the response to this prompt. If your ICAM does not have a "password", enter a null response.

ENTER ICAM LOCAP NAME (4 CHARS,NULL=DEDICATED)

This prompt is requesting the name of the ICAM LOCAP that this TIP/30 system is to use.

If you are using a dedicated ICAM, enter a null response.

If you are using a GLOBAL ICAM enter the desired LOCAP name as it appears in the ICAM generation for your site (this is the name on a LOCAP statement that specifies TYPE=(TCI) in the ICAM generation parameters).

ENTER LBL NAME FOR JCS LIBRARY (DEFAULT=JT)

Each TIP/30 system requires a dedicated job control library for the TIP/30 job control streams. This prompt is requesting the desired LBL name for that library. Since the LBL name may need to be specified when running TIP/30 related jobs, a short LBL name is recommended to minimize typing effort.

ENTER VSN FOR FILE xxxxxxxx (DEFAULT: xxxxxxxx)

This prompt is issued to request the volume serial number for the file name identified in the prompt. In this context, the prompt is issued for the private job control library identified in the previous prompt.

Enter the volume serial number for the disk where the job control library for this TIP/30 system is to be located.

After answering the above prompts, the TJ$INS job submits and monitors a number of jobs to:

- Allocate and initialize the various TIP/30 system files
- Catalogue the TIP/30 system files in the operating system catalogue (YCAT)
- Create (if necessary) a job control library for the TIP/30 system and create in that library a number of job control PROCS that are referenced by the various supplied TIP/30 batch jobs.

The following prompt appears at the point where the SYSGEN library is to be allocated:

DOES SYSGEN LIBRARY ALREADY EXIST? Y/N

The answer to this prompt determines whether or not the installation procedure creates a new SYSGEN library or uses an existing SYSGEN library.

If the reply is "Y", you are prompted for the LBL name of the existing SYSGEN library to be used. If the existing SYSGEN library is not catalogued, you are prompted for the VSN for that library so that it can be catalogued.

If the reply to this prompt is "N", the installation procedure creates a SYSGEN library with the default file name (TIP.id.GEN).

When all of the files are allocated and prepared, a bootstrap version of TIP/30 is assembled and the final job (TJ$TIP) is executed to run this bootstrap version of TIP/30. At this point, the quick installation of TIP/30 is complete — although the TJ$INS job remains running until the bootstrap TIP/30 job (TJ$TIP) terminates.

The TJ$TIP job stream includes a PAUSE statement that is issued to permit you to wait until the necessary job streams have finished running:

REPLY WHEN TIPGEN COMPLETED (xxxx$GEN)

Reply to this message when the job xxxx$GEN has completed — "xxxx" represents the system identifier (LOCAP or NETWORK name).

The last portion of the workstation or console output of the TJ$TIP job is similar to the following:

```
58 TJ$INS   JOB TJ$TIP STARTED.
59 TJ$TIP   ****************************************
60 TJ$TIP   *                                      *
61 TJ$TIP   *  T I P / 3 0   V E R S I O N   4 . 0  *
62 TJ$TIP   *                                      *
63 TJ$TIP   ****************************************
64 TJ$TIP   TI01  USING   TCA=BOOT40
65 TJ$TIP   TI03  USING   LOCAP=TST2
66 TJ$TIP   TI04  USING   NET=NET2
67 TJ$TIP   JC01  JOB TJ$TIP   EXECUTING JOB STEP TB$TIP00 #001 11:20:23
68 TJ$TIP   TI001 TIP/30 INITIALIZATION   ALLINSON-ROSS CORPORATION
69 TJ$TIP   TI152 TIP/30 VERSION: 4.0 C40R0-000   TCA MODULE: BOOT4000
70 TJ$TIP   TI145 TIP$CAT    5% FULL,    0% RECORDS NOT AT HOME POSITION
71 TJ$TIP   TI145 TIP$MCS    0% FULL,    0% RECORDS NOT AT HOME POSITION
72 TJ$TIP   TI149        736K ALLOCATED FOR PAGED MEMORY (PROGRAMS)
73 TJ$TIP   TI079 TIP$RNDM -       0 BLOCKS IN USE
74 TJ$TIP   TI002 89/09/27 - 11:21:02 TIP/30 READY FOR TST2 TIP30
75 TJ$TIP   TI035 WEDNESDAY SEPTEMBER 27 1989 AT 11:21:02
```

Of course, some of the details may vary depending on your choice of ICAM network name, LOCAP name and so on.

At this point, two jobs are running:

1. TJ$INS (the installation job) which executed the bootstrap TIP/30 system
2. TJ$TIP — the bootstrap TIP/30 system.

The TJ$INS job waits for the job TJ$TIP (TIP/30) to terminate.

The next step is to LOGON the bootstrap version of TIP/30 to complete Part II of the TIP/30 installation procedure (the online portion). Proceed to "10.2. Installation — PART II" on page 10-24.

10.1.3. Step 2B — Detailed Install

This section describes the installation procedures that are followed when a "quick installation" cannot be used.

When a "Standard Installation" is selected from the main menu of the TJ$INS job, the following prompts appear:

```
 12 TJ$INS   FUNCTION: ENTRY LEVEL TIP/30 (TPS) INSTALLATION
 13 TJ$INS   THIS FUNCTION WILL RESULT IN A NEW TPS SYSTEM
 14 TJ$INS   BEING INSTALLED.
 15 TJ$INS   PERFORM THIS FUNCTION? Y/N
▶15 y
 16 TJ$INS   ENTER LBL NAME OF TIP/30 RELEASE LIBRARY ($Y$TIP)
▶16
 17 TJ$INS   DOES TIP/30 RELEASE LIBRARY ALREADY EXIST? (Y/N)
▶17 y
 18 TJ$INS   RE-LOAD TIP RELEASE LIBRARY (NOW)? Y/N
▶18 n
 19?TJ$INS   USE DEFAULT SIZES FOR TIP/30 SYSTEM FILES? Y/N
▶19 n
```

In general, the prompts are similar to the prompts issued for the quick install, except that you may be prompted for the sizes for individual files and you may be prompted for the desired VSN for individual files depending on your response to the last question shown above. Another difference is that some of the TIP/30 files are optional (for example, the TIP/30 Journal file — TIP$JRN); you will be prompted to decide whether or not such optional files are to be allocated at this time.

Remember, at any prompt you may issue any of the global "@" commands that are available, such as: @VOL, @HELP, etc.

As the prompts are answered, the TJ$INS job submits and monitors jobs to:

- Allocate and prepare the various TIP/30 system files
- Catalogue the TIP/30 system files in the system catalogue (YCAT)
- Create a job control library for the TIP/30 system and create in that library a number of job control PROCS that are referenced by the various supplied TIP/30 batch jobs.

When all of the files are allocated and prepared, a bootstrap version of TIP/30 is assembled and the final job (TJ$TIP) is executed to run this bootstrap version of TIP/30. At this point, the batch portion of the installation of TIP/30 is complete — although the TJ$INS job does not terminate until the job TJ$TIP terminates.

The workstation or console output of the TJ$TIP job stream will look something like this:

```
54 RV3732   TJ$TIP: LOCAP=TST2, NET=NET2, ID=TST2
55 RV3732   TJ$TIP: ASSOCIATED  ALT JCS LIB:JT
56 TJ$INS   JOB TJ$TIP COMPLETED BY RUN PROCESSOR.
57 RV3732   R03 RUN PROCESSOR SUCCESSFULLY PROCESSED TJ$TIP
58 TJ$INS   JOB TJ$TIP STARTED.
59 TJ$TIP   ****************************************
60 TJ$TIP   *                                      *
61 TJ$TIP   *  T I P / 3 0   V E R S I O N   4 . 0  *
62 TJ$TIP   *                                      *
63 TJ$TIP   ****************************************
64 TJ$TIP   TI01  USING   TCA=BOOT40
65 TJ$TIP   TI03  USING   LOCAP=TST2
66 TJ$TIP   TI04  USING   NET=NET2
67 TJ$TIP   JC01  JOB TJ$TIP   EXECUTING JOB STEP TB$TIP00 #001 11:20:23
68 TJ$TIP   TI001 TIP/30 INITIALIZATION   ALLINSON-ROSS CORPORATION
69 TJ$TIP   TI152 TIP/30 VERSION: 4.0 C40R0-000   TCA MODULE: BOOT4000
70 TJ$TIP   TI145 TIP$CAT    5% FULL,    0% RECORDS NOT AT HOME POSITION
71 TJ$TIP   TI145 TIP$MCS    0% FULL,    0% RECORDS NOT AT HOME POSITION
72 TJ$TIP   TI149        736K ALLOCATED FOR PAGED MEMORY (PROGRAMS)
73 TJ$TIP   TI079 TIP$RNDM -       0 BLOCKS IN USE
74 TJ$TIP   TI002 89/09/27 - 11:21:02 TIP/30 READY FOR TST2 TIP30
75 TJ$TIP   TI035 WEDNESDAY SEPTEMBER 27 1989 AT 11:21:02
```

Of course, some of the details may vary depending on your choice of ICAM network name, LOCAP name and so on. The important line to look for is the highlighted one shown in the above display — the TIP/30 ready message.

At this point, two jobs are running:

1. TJ$INS (the installation job) which executed the bootstrap TIP/30 system
2. TJ$TIP — the bootstrap TIP/30 system.

The TJ$INS job waits for the job TJ$TIP (TIP/30) to terminate.

The next step is to LOGON the bootstrap version of TIP/30 to complete Part II of the TIP/30 installation procedure (the online portion). Proceed to "10.2. Installation — PART II" on page 10-24.

10.2. Installation — PART II

Once the bootstrap version of TIP/30 is generated, TIP/30 is executed to provide a platform for the final installation steps that are performed using online programs and utilities.

10.2.1. Accessing the TIP/30 System

This section describes how to logon the bootstrap TIP/30 system. The assumption is made that the bootstrap TIP/30 system (job name TJ$TIP) is executing.

To logon the TIP/30 system, you must use a terminal that is able to access the TIP/30 system. Access to the TIP/30 system may vary depending on terminal type, type of ICAM, and whether or not a DCP (Distributed Communications Processor) is involved.

The following points provide guidelines for accessing the TIP/30 system:

- WORKSTATION — Before attempting to connect to TIP/30, you must first logon OS/3 Interactive Services. Once that task is complete, enter system mode and enter a $$SON command to connect to the TIP/30 LOCAP. When the connection is made with TIP/30, the workstation side of the terminal is connected to TIP/30. Proceed to "10.2.2. Logon TIP/30" on page 10-25.
- Dedicated ICAM — In a dedicated ICAM environment, the terminals that are defined in the network that TIP/30 is using should be polling and directly available for use. Proceed to "10.2.2. Logon TIP/30" on page 10-25.
- GLOBAL ICAM — In a GLOBAL ICAM, it may be necessary to make a connection from the terminal chosen to the TIP/30 LOCAP by issuing a $$SON command. Terminals may optionally be hard assigned to the TIP/30 LOCAP by using a SESSION statement in the ICAM generation parameters. In that case, those terminals are considered to be permanently connected to the TIP/30 LOCAP and a $$SON command is not needed. Issue a $$SON to the LOCAP name that TIP/30 is using and proceed to "10.2.2. Logon TIP/30" on page 10-25.

10.2.2. Logon TIP/30

Once a connection is made to the TIP/30 system, a TIP/30 ready message is normally displayed (the ready message may not appear in a dedicated ICAM environment):

```
<<< TIP/30 Version 4.0 C40R0-000  Ready for xxxxxxxxxxx yy/mm/dd  hh:mm >>>

             *  BOOT STRAP SYSTEM  *
             *        BOOT40       *
```

To logon the TIP/30 system, press any function key, the XMIT key, or the MSG WAIT key. The TIP/30 system responds with a prompt like the following to permit you to logon the TIP/30 system:

```
WEDNESDAY AUGUST 27 1989           Please logon

Allinson-Ross Corporation      TIP/30     Logon
Enter:User-Id/Password/Account-No     Site = xxxxxxxxxxx
 ▶
```

The cursor is positioned immediately to the right of the start of entry character on the bottom line. The prompt instructs you to enter a userid followed by a password.

The TIP/30 system initially contains a single defined userid and password. Enter the following information and press the XMIT key:

```
WEDNESDAY AUGUST 27 1989           Please logon

Allinson-Ross Corporation      TIP/30     Logon
Enter:User-Id/Password/Account-No     Site = xxxxxxxxxxx
 ▶TIP30/ADMIN
```

Note: *If your response is not accepted (spelling problem?), the LOGON prompt appears again. If your response takes more than 60 seconds, the screen is erased. If this happens, start this logon procedure over again (by pressing a function key, the MSG WAIT key, or the XMIT key).*

Once the logon userid and password have been correctly entered, the standard TIP/30 system prompt is displayed on the terminal:

```
TIP?▶
```

The logon to the TIP/30 system is now complete.

10.2.3. Step 3 — Online Install

Once the bootstrap version of TIP/30 has been initially generated, and you have logged on TIP/30 according to the information provided in the previous section, the installation procedure is continued using an online program.

To continue the installation procedure, enter the following on the TIP/30 command line and press the XMIT key:

```
TIP?▶INSTALL
```

The INSTALL program displays the following main menu:

```
========== Allinson-Ross Corporation ==========
           TIP/30 Online Installation
================================================

You may use Function keys instead of numbers
to select functions.  (ie F1 = 1)

Main menu

1. Entry Level TIP/30 (TPS) initial install menu
2. IMS Conversion menu
3. Product menu
4. TIP/30 Update install menu
8. Help
9. Quit program            (or MSG-WAIT)

Enter number for corresponding function. ▶
```

The program expects a choice to be entered. You may enter the desired numeric value and press the XMIT key, or press the function key to select the corresponding item. The MSG WAIT key corresponds to the quit function for this and all subordinate menus.

The main menu selections offer the following choices:

1. Entry Level TIP/30 (TPS) initial install menu

Selection 1 displays a subordinate menu of functions that are available for the initial installation of the Transaction Platform System (TPS).

2. IMS Conversion menu

Selection 2 displays a subordinate menu of functions that are available to assist in the conversion of generation parameters from an existing IMS system.

3. Product menu

Selection 3 displays a subordinate menu of functions that are available to install and/or maintain other products related to TIP/30

4. TIP/30 Update install menu

Selection 4 displays a subordinate menu of functions that are available for miscellaneous post-installation activities.

8. Help

Selection 8 displays help information about this menu.

9. Quit program

Selection 9 terminates the online INSTALL program.

10.2.4. Step 4 — Load Screen Formats

Many of the utilities supplied with the TIP/30 system use TIP/30 screen formats to interact with the terminal. These screen formats are supplied in the TIP library and must now be loaded into the TIP/30 screen format file (LFD=TIP$MCS).

To perform this function, select item number 1 from the main install menu:

```
========== Allinson-Ross Corporation ==========
           TIP/30 Online Installation
================================================

You may use Function keys instead of numbers
to select functions.  (ie F1 = 1)


Main menu

1. Entry Level TIP/30 (TPS) initial install menu
2. IMS Conversion menu
3. Product menu
4. TIP/30 Update install menu
8. Help
9. Quit program           (or MSG-WAIT)


Enter number for corresponding function. ▶1
```

As a result, the following subordinate menu appears:

```
========== Allinson-Ross Corporation ==========
           TIP/30 Online Installation
================================================


Initial Install TIP/30 menu

1. Initial install of TIP/30      (Define screens)
2. Install TIP/30 Sample Program file/data  (TSP)
8. Help
9. Quit menu & return     (or MSG-WAIT)


Enter number for corresponding function. ▶1
```

Item number 1 is the action required to load the TIP/30 screen formats.

After selecting item 1, the display will output one line for each screen format that is restored from the TIP library. When all screens have been restored, control returns to the menu shown above.

10.2.5. Step 5 — Load Sample File Data

The sample program supplied with the TIP/30 system uses a small MIRAM file (LFD=TIP$TSP) to contain sample data to demonstrate how the program operates.

To load data into the TIP$TSP file, select item number 2 from the Initial Install TIP/30 menu:

```
========== Allinson-Ross Corporation ==========
           TIP/30 Online Installation
================================================


Initial Install TIP/30 menu

1. Initial install of TIP/30      (Define screens)
2. Install TIP/30 Sample Program file/data  (TSP)
8. Help
9. Quit menu & return     (or MSG-WAIT)


Enter number for corresponding function. ▶2
```

After selecting item 2, a procedure is run to load a small number of records into the TIP$TSP file. When the file has been loaded, control returns to the menu shown above.

10.2.6. Step 6 — Convert IMS Parameters

This step is required if you currently run the Information Management System (IMS) product from Unisys and you wish to migrate the IMS transactions to the TIP/30 system.

Before attempting this, you must copy the IMS configuration parameters to the TIP/30 SYSGEN library. The default LBL name of the SYSGEN library is "TIP.id.GEN" ("id" is the system identifier) unless an alternate name was specified during a detail install.

The install main menu indicates that there is a subordinate menu for the conversion of IMS based systems (item 2):

```
========== Allinson-Ross Corporation ==========
           TIP/30 Online Installation
================================================

You may use Function keys instead of numbers
to select functions.  (ie F1 = 1)


Main menu

1. Entry Level TIP/30 (TPS) initial install menu
2. IMS Conversion menu
3. Product menu
4. TIP/30 Update install menu
8. Help
9. Quit program           (or MSG-WAIT)


Enter number for corresponding function. ▶2
```

The following subordinate menu is displayed:

```
========== Allinson-Ross Corporation ==========
           TIP/30 Online Installation
================================================


IMS Conversion menu

For IMS conversion, do these steps in order.
1  Run IMSCNV transaction to begin IMS conversion
2. RV job to copy IMS load modules to TIP$LOD
3. Run CAT to process IMS generated statements
8. Help --- Please read - note new TIPGEN.
9. Quit menu & return    (or MSG-WAIT)


Enter number for corresponding function. ▶
```

To convert the IMS parameters to the corresponding TIP/30 parameters, perform steps 1 through 3 in sequence.

Note: *Conversion of IMS parameters does not create a loadable TIP/30 for these parameters. TIP/30 must be generated after the conversion process is complete. For a description, please select option 8 (HELP) of the IMS Conversion Menu.*

Note: *Please see the IMS to TIP/30 Conversion Restrictions in the System Release Description (SRD) for your operating system.*

10.2.7. Step 7 — Customize TJ$TIP Job Stream

The job stream TJ$TIP that is executed to run the bootstrap TIP/30 system must be cloned to create the job stream that will be regularly used to execute TIP/30. Although the TJ$TIP jobstream has a number of global symbols that can be overridden, we recommend that a customized version of the job be created — this job is often called "TIP" or "TIP30", but you are free to choose any name you wish.

At some sites, various housekeeping activities are required before and possibly after TIP/30 is run. This cloned job stream is an ideal place to execute such job steps.

Until such time as a customized version of this job stream is created, you may start TIP/30 by running the TJ$TIP job stream directly:

```
RV  TJ$TIP:JT
```

Note: *This RV command assumes that the default TIP/30 job control library LBL name was chosen (LBL=JT). If this is not the case, change ":JT" to the LBL name that was chosen.*

The TJ$TIP job contains a number of globals variables that may be overridden on the RV command. The default value for all of the global symbols are established by the initial installation dialogue with the job TJ$INS; the initial global values are set in a job control proc named "TIPFILES" located in the same library as the TJ$TIP job stream.

An important global defined in the TJ$TIP job is TCA=. This global specifies the name assigned to a set of TIP/30 generation parameters and controls which of many potential "generations" of TIP/30 that is to be executed. For further information, see the documentation of TIP/30 System Generation.

10.2.8. Step 8 — Shutdown TIP/30

When you have finished performing the installation steps, TIP/30 may be shutdown by issuing the following command at the TIP/30 command line:

```
TIP?▶EOJ  1
```

The EOJ transaction begins an orderly TIP/30 system shutdown. If TIP/30 does not begin shutdown within approximately one minute, there may be terminals connected to the TIP/30 system that are in middle of a transaction. To force those users off the TIP/30 system, issue the following operator command from the system console:

```
UNS TJ$TIP STOP
```

The STOP command terminates TIP/30 immediately.

10.3. TIP/30 Impact on Users

The installation of TIP/30 may have some observable impact for terminal operators. Although all existing IMS action programs will perform as they did under IMS, other aspects of TIP/30 may mean slight changes for user departments.

The security system is an important feature of TIP/30. You may choose not to implement TIP/30 security at this time. If you do want TIP/30 security, a userid and logon password must be established for every user of the system. Both the user-id and password are from one to eight characters in length, and must start with a letter.

If you assign user-ids, be sure to make an accurate list of each user-id and the associated password.

10.4. Multiple TIP/30 Systems

This section provides information for sites which plan to run more than one TIP/30 system on a single processor.

To be able to run more than one TIP/30 system (or a combination of Unisys IMS and TIP/30 systems), it is necessary to run a Global ICAM. A Global ICAM permits the definition of a number of LOCAPs — each of which represents an application that uses ICAM services.

For an example of the generation parameters for a Global ICAM, please refer to "Example ICAM Generations" on page 10-10.

Each TIP/30 system must be assigned a unique LOCAP name that it will use (the LOCAP name that a TIP/30 system uses is normally specified in the job control statements that execute that TIP/30 system).

Each TIP/30 system **MUST** be a free-standing system — it must have unique TIP/30 system files for its operation. The following TIP/30 system files — if they are used — *must be unique for each TIP/30 system* and absolutely cannot be shared between TIP/30 systems under any circumstances:

- TIP$BAK — Backup File
- TIP$B4 — Quick Before Image File
- TIP$CAT — Catalogue
- TIP$DUMP — TIP/30 Dump File
- TIP$HST — Journal History File
- TIP$JCS — Job control library
- TIP$JRN — Journal File
- TIP$LOG — Log Tape
- TIP$MCS — Screen Format File
- TIP$RNDM — Random File
- TIP$SWAP— Swap File

- TIP$TOM — IMS Trace Output File

The TIP/30 files that are OS/3 SAT Libraries (namely: TIP and TIP$LOD) may be shared across multiple TIP/30 systems, but care must be taken to avoid conflicting load module name usage in the TIP$LOD library. Of course, this also assumes that both TIP/30 systems are at identical revision levels.

Glossary

This section supplies working definitions of some of the common terms used in the TIP/30 documentation. The definitions are not intended to be rigorous; they are explanations within the context of the TIP/30 system.

A

ACK Acknowledge(ment). A signal indicating that error detection logic has failed.

ASCII American Standard Code for Information Interchange. A set of character representations that associates single byte binary values with external graphic characters. See also "byte" and "EBCDIC". The ASCII character set is typically used by communications hardware for data transmission.

asynchronous

Happening simultaneously but independently.

auxiliary device

A peripheral unit (such as a printer, diskette, or cassette) attached to a terminal.

B

Background

As in ... process. A background process is a TIP/30 online transaction that is running but not associated with a physical terminal.

Background processes are non-interactive programs.

batch Not interactive.

bi-synch Bi-synchronous; a communications protocol which implies that traffic is synchronized in both directions by acknowledgement messages.

bit bucket A mythical and cavernous receptacle which is provided by hardware manufacturers to hold any data which is deliberately or accidentally mislaid during data manipulation.

For example, digits to the right of the decimal place that are truncated by a move operation fall into the bit bucket.

bypass A terminal that has an identifiable polling address but typically has no keyboard or display screen. Bypass terminals are often utilized to perform printing operations since they do have memory and auxiliary device capability.

byte The smallest addressable unit of storage in memory. A byte is composed of 8 bits (binary digits). The value of a byte ranges from zero to 255 (decimal) or 0 to FF (hexadecimal). Each of the characters in the computer's character set (either ASCII or EBCDIC) may be stored in a byte using a unique representation from the 256 possible binary values that may be stored in a byte.

C

catalogue (OS/3)

A directory of file names and corresponding volume label location information (stored in file YCAT).

catalogue (TIP/30)

A directory of information about users, transaction programs, and online files.

CRT Literally, Cathode Ray Tube. Often used to refer to the display screen of a computer terminal.

cursor A current position marker on a CRT. Usually a blinking rectangle or underline character that reminds the user where the next character will appear on the screen.

D

Direct Access

A file organization technique that numbers fixed size records using integers from 1 to the highest record number.

Doubleword

On OS/3 hardware a doubleword is 8 consecutive bytes beginning on an address that is evenly divisible by 8 (the right most 3 bits of the address are zero).

An area is said to be "doubleword aligned" if it begins at an address that is evenly divisible by 8.

COBOL aligns all WORKING-STORAGE level 01 items on a doubleword boundary.

TIP/30 aligns all of the external work areas for a transaction program (PIB, CDA, MCS, WORKAREA) on a doubleword boundary.

dynamic file

A TIP/30 pseudo-file that has the characteristics of direct access.

May be created, manipulated and erased (scratched) on demand by TIP/30 transaction programs.

E

EBCDIC Extended Binary-Coded Decimal Interchange Code. A set of character representations that associates single byte binary values with external graphic characters. See also "byte" and "ASCII". The EBCDIC character set is typically used by the CPU for internal data representation.

edit buffer A particular type of TIP/30 dynamic file that is used by the TIP/30 text editors as a work space for editing.

element The name of a library member or module.

F

FCS File Control System. TIP/30 interface between programs and on-line files.

Foreground

As in ... process. A foreground process is a TIP/30 online transaction that is running at a physical terminal.

Fullword

On OS/3 hardware a fullword is 4 consecutive bytes beginning on an address that is evenly divisible by 4 (the right most 2 bits of the address are zero).

An area is said to be "fullword aligned" if it begins at an address that is evenly divisible by 4.

A fullword can be defined in COBOL by specifying a PICTURE of 9(6) through 9(9) COMP SYNC.

Function Key

A key on a UNISCOPE terminal keyboard (numbered F1 through F22) which signals the host computer when pressed. NO data is sent from the terminal.

H

Halfword

On OS/3 hardware a halfword is 2 consecutive bytes beginning on an address that is evenly divisible by 2 (the right most bit of the address is zero).

An area is said to be "halfword aligned" if it begins at an address that is evenly divisible by 2.

A halfword can be defined in COBOL by specifying a PICTURE of 9(1) through 9(4) COMP SYNC.

hardware The physical computer equipment.

hashing A technique of computing a key from a value. Typically used to map a large number of values onto a smaller set of values.

Host computer

The main computer; the computer which is running TIP/30.

I

IMS A Unisys software product that provides an execution environment for transaction programs.

IMS emulation

A facility of TIP/30 which enables a transaction program written to use the facilities of IMS to run under control of TIP/30 without change or recompilation.

index A collection of keys and associated location information that can be searched to locate an item with a given key.

interactive

Operating in "question and answer" mode.

An interactive program presents decisions for a user to make and acts according to the response.

ISAM Indexed Sequential Access Method. A file organization method that allows access to records either randomly by a single key or sequentially by a single key.

Records may be fixed length or variable length (in the Unisys OS/3 implementation).

K

key A portion of the data in a record which is used to index the record.

L

LFD The name of a file as stated in the Job Control information for the job which refers to the file.

LFN Logical File Name. The name by which a TIP/30 program refers to a file. The logical file name is associated with the LFD name of the file by TIP/30 catalogue information.

M

MIRAM Multiple Indexed Random Access Method. File organization method that is similar to ISAM with the exception that there may be from one to five keys.

MSG-WAIT

Key on UNISCOPE terminals that signals the host computer when pressed (NO data is sent from the terminal).

multi thread

A number of transactions concurrently sharing resources.

N

native mode

A program that uses TIP/30 facilities that is NOT running under the control of the TIP/30 IMS/90 emulator is said to be running in this mode.

NAK Negative acknowledgement.

O

OS/3 Operating System 3. The control software supplied by Unisys for use on Series 90 and System 80 machines.

P

prefix notation

A notation convention adopted by most TIP/30 utilities to allow selection by prefix.

Eg: "*ABC" means all names with prefix "ABC"

Eg: "!XYZ" means all names NOT with prefix "XYZ"

An imbedded "?" or "." character implies that the corresponding position may be occupied by any character (for example: *A??B matches A12B or AXYB).

S

single thread

A method of transaction processing which allows one transaction to monopolize resources until completion of the transaction.

SOE (character). Start Of Entry character. On UNISCOPE terminals a character (shaped like a pennant blowing from left to right) which marks the left most boundary of data to be transmitted to the host computer.

Example: ▸

software The programs which control the operation of the hardware or other (application) programs.

T

TPS Transaction Platform System. A significant subset of the TIP/30 system that is included with System 80 Model 7E processors and available as a priced item for other OS/3 hardware platforms.

transaction

A program that executes under the control of TIP/30.

TIP See TIP/30.

TIP/30 Transaction Interface Processor — a system software product of Allinson-Ross Corporation.

U

unsolicited

As in ... message. A message sent to a terminal that is not necessarily a response to a previous input message.

In effect, a message sent gratuitously by another process in the system which arrives unexpectedly.

An unsolicited message is queued by ICAM until such time as the terminal operator presses the MSG-WAIT key (at which time ICAM will display the message on the terminal).

X

XMIT Transmit. A key on UNISCOPE terminals that sends data from the CRT to the host computer.

Appendix A
Programming Reference

The following table may be used to convert decimal numbers (base 10) to and from hexadecimal numbers (base 16).

Table A-1. 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	0	0	0	0	0	0
1	1 048 576	1	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	14 680 064	E	917 504	E	57 344	E	3 584	E	224	E	14
F	15 728 640	F	983 040	F	61 440	F	3 840	F	240	F	15

Table A-2. 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	1 024	21	2 097 152	32	4 294 967 296

Table A-3. 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
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

The following table is the character code table for the American Standard Code for Information Interchange (ASCII).

Table A-4. ASCII Code Chart

	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	sp	!	"	#	$	%	&	'	(	)	*	+	,	_	.	/
9x	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
Ax	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
Bx	P	Q	R	S	T	U	V	W	X	Y	Z	[	\	]	^	_
Cx	'	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
Dx	p	q	r	s	t	u	v	w	x	y	z	{	|	}	~	del
Ex	— not defined —															
Fx	— not defined —															

The following table is the character code table for the Extended Binary-Coded Decimal Interchange Code (EBCDIC).

Table A-5. EBCDIC Code Chart

	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						

Index

A

B

C

D

E

F

G

H

I

J

K

L

M

N

O

P

Q

R

S

T

U

V

W

X