TIP/30 Reference Manual

Version 2.5 (82/08/01)

ARP-600

A Product of:

Allinson-Ross Corporation
First Rexdale Place,
155 Rexdale Boulevard, Suite 906
REXDALE, Ontario
Canada M9W 5Z8
TEL: (416) 746-3388
TWX: (610) 491-1772

```
**********************************************************************
**********************************************************************
**                                                                  **
**    A    L       L       IIIII  N   N    SSSS   OOO   N   N       **
**   A A   L       L         I    NN  N   S      O   O  NN  N       **
**  AAAAA  L       L         I    N N N    SSS   O   O  N N N  =====  **
**  A   A  L       L         I    N  NN       S  O   O  N  NN       **
**  A   A  LLLLL   LLLLL   IIIII  N   N   SSSS    OOO   N   N       **
**                                                                  **
**  RRRR    OOO    SSSS   SSSS       CCC    OOO   RRRR   PPPP      **
**  R   R  O   O  S      S          C      O   O  R   R  P   P      **
**  RRRR   O   O   SSS    SSS       C      O   O  RRRR   PPPP       **
**  R  R   O   O      S      S      C      O   O  R  R   P      ..   **
**  R   R   OOO   SSSS   SSSS       CCC    OOO   R   R  P      ..   **
**                                                                  **
**   CCC    OOO   PPPP   Y   Y  RRRR   IIIII   GGG   H   H  TTTTT   **
**  C      O   O  P   P   Y Y   R   R    I    G   G  H   H    T     **
**  C      O   O  PPPP     Y    RRRR     I    G      HHHHH    T     **
**  C      O   O  P        Y    R  R     I    G  GG  H   H    T     **
**   CCC    OOO   P        Y    R   R  IIIII   GGGG  H   H    T     **
**                                                                  **
**********************************************************************
** (C) 1975,1976,1977,1978,1979,1980,1981,1982                      **
** Allinson-Ross Corporation  reserves the right to modify or revise **
** the  content  of this  document.  Except  where a  Software  Usage **
** Agreement  has been executed, no  contractual obligation  between **
** Allinson-Ross  Corporation and the recipient is  either expressed **
** or implied. It is agreed and understood that the information con- **
** tained herein is proprietary and confidential and that the recip- **
** ient shall take all necessary precautions to ensure the confiden- **
** tiality thereof. This  document, in whole or in  part, may not be **
** copied  or transmitted, in  any form or by any means, electronic, **
** mechanical, photocopying, or otherwise, without the prior written **
** permission of:                                                   **
**                                  Allinson-Ross Corporation,      **
**                                  155 Rexdale Blvd, Suite 906,    **
**                                  Rexdale, Ontario,               **
**                                  Canada            M9W 5Z8       **
**                                  Tel: (416) 746-3388             **
**                                                                  **
**                                                                  **
**********************************************************************
**           THIS  DOCUMENT  WAS PRODUCED USING THE                 **
**         ALLINSON-ROSS CORPORATION DOCUMENT GENERATOR.             **
**********************************************************************
**********************************************************************
```

1. CHAPTER I - INTRODUCTION

1.1 PREFACE

This document is the reference manual for TIP/30 (Transaction Interface Processor), a software product developed by Allinson-Ross Corporation.

The names TIP and TIP/30 are used interchangeably in this manual.

Please direct any inquiries or requests for further information to:

Allinson-Ross Corporation
First Rexdale Place
155 Rexdale Blvd., Suite 906
Rexdale, Ontario
Canada M9W 5Z8
Tel. (416) 746-3388
TWX. (610) 491-1772

1.2 HOW TO USE THIS REFERENCE MANUAL HOW TO USE

The TIP/30 reference manual is organized into volumes, chapters, sections, and sub-sections.

The division into volumes is purely a matter of convenience for physical handling and distribution.

The manual is a hierarchy of information. The information in the manual is presented in a "top down" fashion. By this we mean that the most general information is presented first, proceeding on to more specific information at a later point in the documentation.

The page numbers at the bottom of each page are relative to each individual sub-section.

Each sub-section is terminated by a sentinal such as: -+*+-

Following is a brief description of the contents of the chapters in the reference manual.

HOW TO USE THIS REFERENCE MANUAL

Chapter I	An overview of TIP/30 and the design philosophy of the product. This section on how to effectively use the manual. A generalized table of contents. A glossary of terms used in the manual.
Chapter II	A presentation of some of the fundamental concepts necessary to understand the use of TIP/30.
Chapter III	The detailed documentation for all supplied on-line utility programs.
Chapter IV	Documentation for the Allinson-Ross Application generator utility (Tip Query Language).
Chapter V	The program control system (how to create TIP native mode programs).
Chapter VI	The file control system (interface with user data files).
Chapter VII	The message control system (terminal interface).
Chapter VIII	System Maintenance information. (The care and feeding of TIP/30).
Chapter IX	Appendices.
Chapter X	Key Word In Context (KWIC) index. A KWIC index is produced by indexing every non-trivial word in title information. Users are then able to find appropriate information even if they only remember one conceptual key word. Although the KWIC index is admittedly voluminous it is invaluable.

HOW TO USE THIS REFERENCE MANUAL

This is a reference manual for the product TIP/30. It is not intended as a tutorial introduction to (on-line) data processing. There is no substitute for a proper grasp of the fundamentals of data processing terms and ideas.

Readers of this manual are assumed to have a reasonable understanding of general data processing principles. Descriptions of how terminals work, how the operating system works and the like are far beyond the scope and intent of this manual. The reader is urged to consult the appropriate publications from the manufacturer of the hardware for this type of information.

It is recommended that ALL users read Chapters I and II thoroughly before attempting to use the features of TIP/30.

Programmers (or users) who will need to know how to run (some of) the supplied utilities will find Chapter III indispensible as a reference.

Chapter IV (Applications development systems) describes an on-line query language. We strongly recommend (as a matter of fundamental philosophy) that the Data Processing personnel use this (tool!) as a convenient means to generate applications. Very few user department personnel will have the capability of making effective use of this product without extensive training AND support from the Data Processing Department.

Programmers must read and understand much of Chapters V through VII before attempting to create new TIP/30 programs of their own.

Chapter VIII (System maintenance) is of primary interest to those individuals who are responsible for the ongoing maintenance and support of TIP/30 at the site. The system programmer must have a thorough knowledge of this chapter to be able to understand fully the ramifications of any changes he may envision for the TIP/30 system at the site.

Chapter VIII also contains all the necessary documentation (including console messages and commands) for the machine operator who must run TIP/30 and be aware of the status of the on-line system.

Chapter IX contains appendices which provide supplementary documentation for various topics.

Chapter X is the Key Word In Context (KWIC) index. This index may be consulted to find information when the user is unsure of its location in the manual.

1.3 REFERENCE MANUAL STRUCTURE STRUCTURE

==========

TIP/30 REFERENCE MANUAL - STRUCTURE

==========

VOLUME I.

CHAPTER I: INTRODUCTION

- Preface
- How to use this document
- Structure of this document
- Table of Contents
- Glossary of terms and concepts
- Introduction to TIP/30 (overview)

CHAPTER II: FUNDAMENTAL CONCEPTS

- User identification and Password
- Logon and Logoff procedures
- Command line / options / parameters
- System Security / Catalogue

CHAPTER III: EXECUTING TIP/30 UTILITY PROGRAMS

- Alphabetically ordered documentation of transactions supplied by Allinson-Ross that may be executed at a terminal.

- ACCESS Assign a dynamic file
- AFT Active File Table
- APB All Points Bulletin
- ASG Assign a dynamic file

- BASIC BASIC language interpreter
- BCP Batch terminal Control Program

- CAT TIP/30 Catalogue Manager
- CC COBOL Converter (Reformatter)
- CCA ICAM Statistics Display
- CPAGE Change UTS400 control page
- CRASH Terminate TIP/30 with a dump
- CREATE Create dynamic file

- DD, DDU Dynamic file display
- DEBUG Set file in debug mode
- DEFKEY Define function keys
- DIE Abort TIP/30 transaction
- DLL Down Line Load Utility
- DOC Document Processor

- EOJ TIP/30 normal end of job

- FCLOSE Physically Close OS/3 file
- FIN Logoff
- FOPEN Physically Open OS/3 file
- FREE terminate Access to a file

- HELP display HELP information

- IDA interactive debug aid

- JBQ display OS/3 job queue
- JCL interactive JCL submittor

- LOGOFF Log off TIP/30
- LOGON Log on TIP/30

- MAIL interactive Mail
- MEM display OS/3 memory usage
- MODE set terminal mode
- MSG send a message to terminal

- MSGAR message (screen format) archiver
- MSGDEF message definition
- MSGSHOW message test

- NEWUSER logoff and logon in one step
- NOTE display note on terminal

- ODD Online Data Display

- PMDA Post Mortem Dump Analysis

- QED TIP/30 text editor

- RELOAD refresh load module from library
- RPG editor for RPG source programs
- RV run an OS/3 batch job

- SCRATCH scratch dynamic file
- SET set terminal characteristics
- SPL spool file processor
- STATUS TIP/30 statistics
- STOP immediate TIP/30 eoj
- SYM OS/3 symbiont submittor
- SYS OS/3 system status

- TCB OS/3 task control block display
- TIPFLG Set TIP/30 flag(s)
- TLIB TIP/30 Librarian

- UTSASM UTS400 assembly language processor

- VTOC online disc VTOC

- WHOSON display users of TIP/30 system
- WMI display user logged on

CHAPTER IV: APPLICATIONS DEVELOPMENT SYSTEMS

- TIP QUERY LANGUAGE (TQL)

VOLUME II.

CHAPTER V: PROGRAM CONTROL SYSTEM

- structure of on-line programs
- linkage section areas
- TIPABRT establish island code linkage
- TIPBITS convert 32 bytes to fullword
- TIPBYTES convert fullword to 32 bytes
- TIPDATE get date including day of week
- TIPDUMP force deliberate program check
- TIPFCER expanded FCS error code
- TIPFLAG flag services
- TIPFORK start asynchronous process
- TIPRTN return to caller
- TIPSNAP storage dump for selected areas
- TIPSUB 'perform' other program
- TIPSUBP 'perform' resident subprogram
- TIPTIMER timer services
- TIPXCTL transfer control to program (GO)

CHAPTER VI: FILE CONTROL SYSTEM

- TIPFCS and the TIP/30 catalogue
- Record and file locking
- Summary of FCS calls
- File types supported
- Common FCS parameters
- FCS interface packets

- INDEXED FILES (ISAM IRAM MIRAM)
- DIRECT ACCESS FILES (DAM)
- SEQUENTIAL FILES (SAM)
- DYNAMIC FILES (DYN)
- TIPPRINT
- FCS copy elements
- LIBRARY FILES
- EDIT FILES
- DATA BASE INTERFACE
- TIP/30 JOURNAL FILE
- batch access to journal file

CHAPTER VII: MESSAGE CONTROL SYSTEM

- Screen format generator
- Line oriented I/O (PROMPT, ROLL etc)
- Direct Control I/O (TIPTERM)

CHAPTER VIII: SYSTEM MAINTENANCE

- TIP/30 file requirements
- TIP/30 Generation
- execution job control and options
- File recovery
- Batch Jobs [job control information]
- TIP/30 Operation and Error messages

CHAPTER IX: APPENDICES

- Directory of COBOL copy elements
- BASIC Language syntax

CHAPTER X: KEY WORD IN CONTEXT (KWIC) INDEX

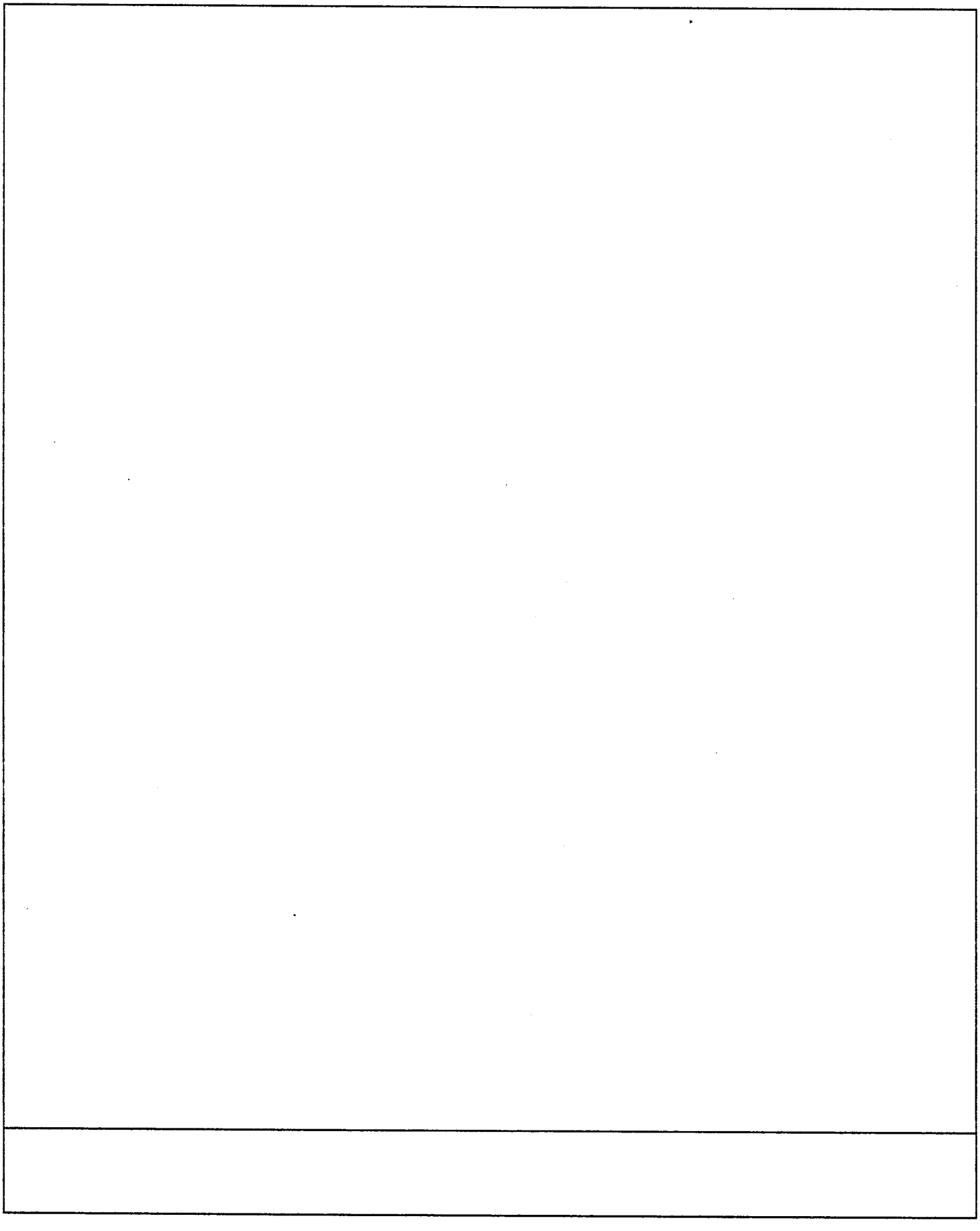


TABLE OF CONTENTS

1.4 TABLE OF CONTENTS **TOC**

- 1 -

- 2 -

- 3 -

TABLE OF CONTENTS

TABLE OF CONTENTS

- 4 -

TABLE OF CONTENTS

TABLE OF CONTENTS

1.5 TIP/30 GLOSSARY OF TERMS AND CONCEPTS GLOSSARY

This section of the manual supplies working definitions of some of the common terms used in this manual. The definitions are not intended to be rigorous; they are explanations within the context of the TIP/30 system.

TIP/30 GLOSSARY OF TERMS AND CONCEPTS

ACK	Acknowledge(ment). A signal indicating that error detection logic has failed.
asynchronous	Happening simultaneously but independently.
auxiliary device	A unit (such as a printer, diskette, or cassette) attached to a terminal.
batch	Not interactive.
bi-synch	Bi-synchronous; a communications protocol which implies that traffic is synchronized in both directions by acknowledgement messages.
catalogue (OS/3)	A directory of file names and corresponding location information.
catalogue (TIP)	A directory of information about user-ids, transaction programs, and on-line 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.
Direct Access (DA)	A file organization technique that numbers fixed size records using integers from 1 to the highest record number.
dynamic file	A TIP/30 pseudo-file that has characteristics of direct access. May be created, manipulated and erased (scratched) on demand by TIP/30 transaction programs.
edit buffer	A particular type of TIP/30 dynamic file that is used by the TIP/30 text editor (QED) as a work space for editting.
element	The name of a library member or module.
FCS	File Control System. TIP/30 interface between programs and on-line files.

Function Key A key on a UNISCOPE terminal keyboard (numbered F1 F2 ... etc) which signals the host computer when pressed.

hardware The physical computer equipment.

hashing A technique of computing a key from information in the record.

Host computer The main computer; the computer which is running TIP/30.

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 will present decisions for a user to make and act according to the response.

IMS/90 A Sperry Univac software product that provides an execution environment for transaction programs.

IMS/90 emulation A facility of TIP/30 which enables a transaction program written to use the facilities of IMS/90 to run under control of TIP/30 without change or recompilation.

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 or variable length (UNIVAC implementation).

key A portion of the data in a record which is used to index the record.

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. Connected to real LFD name by TIP/30 catalogue information.

TIP/30 GLOSSARY OF TERMS AND CONCEPTS

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.

multi thread A number of transactions concurrently sharing resources.

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. (not ACK).

OS/3 Operating System 3. The control software supplied by Sperry Univac for use on series 90 and 80 machines.

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"

single thread One transaction monopolizing 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 leftmost boundary of data to be transmitted to the host computer.

software the programs which control the operation of the hardware or other (application) programs.

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.

TIP/30 GLOSSARY OF TERMS AND CONCEPTS

TOTAL The name of a data base system marketed by CINCOM. An interface to TOTAL is supported by TIP/30.

unsolicited (message). A message sent to a terminal that is not necessarily a response to a previous input message.

XMIT (transmit). A key on UNISCOPE terminals that sends data from the CRT to the host computer.

1.6 TIP/30 OVERVIEW OVERVIEW

TIP/30 is an integrated system of transaction processing and program development software which is compatible with the Sperry Univac OS/3 operating system.

TIP/30 offers the user the following advantages:

* TIP/30 facilitates the development of application systems

* TIP/30 has a large number of productivity tools

* TIP/30 makes most efficient use of hardware resources

* TIP/30 will execute existing IMS/90 action programs without modification with no need to compile or link. Because it is a complete and comprehensive software system, TIP/30 represents the most powerful transaction processing and program development software available to the OS/3 user.

The heart of the TIP/30 software system is a multi-thread Program Control System, an integrated Message Control System, and a comprehensive File Control System. Included in this nucleus is an extensive system access security control facility as well as facilities for maintaining user data base integrity. In addition, Allinson-Ross supplies an extensive library of interactive utility programs to aid in program design, testing, implementation and system monitoring.

TIP/30 OVERVIEW

The TIP/30 Program Control System provides multi-thread control of application programs.

Through a concept of program stacking, TIP/30 provides for inline return from all external program CALL's. This feature allows each individual program to do more work, thereby reducing the number of programs required in an online system.

TIP/30 allows the application designer a great degree of freedom in the design of applications.

The TIP/30 Program Control System allows application designers to concentrate on the application instead of ways to overcome unreasonable system constraints.

1.6.1 MESSAGE CONTROL SYSTEM OVERVIEW

The TIP/30 Message Control System is an integrated facility which provides user programs and programmers complete freedom from terminal hardware characteristics.

MCS screen formats are developed interactively and stored in the TIP/30 catalogue.

TIP/30 provides a utility called MSGSHOW which is used to test developed screen formats with no programming required. Users therefore, can participate in the design of screen formats. MSGSHOW makes it easier to develop online systems that feel comfortable to the user.

MCS allows programs to be written with no concern for the physical hardware characteristics of the terminal. The user program deals only with data. TIP/30 assumes the responsibility for knowing the hardware characteristics of the terminal. TIP/30 users can take advantage of new terminal hardware with no programming changes.

-+*+-

1.6.2 FILE CONTROL SYSTEM OVERVIEW

The TIP/30 File Control System provides an efficient interface to all standard data management files as well as integrated data base systems such as CINCOM's "TOTAL Data Base Management System".

FCS provides both record and file locking to preserve data integrity.

Automatic journalling of file updates provides for on-line or off-line recovery from system failures.

A system for creating, maintaining and scratching temporary scratch-pad files allows for flexible application design.

-+*+-

1.6.3 SECURITY OVERVIEW

System-wide security in the TIP/30 system is maintained through the TIP/30 catalogue file. All users, programs and files must be catalogued before they can be referenced online. TIP/30 guarantees security for a user, his programs and his files.

A horizontal layering of security is achieved by the use of a security level number in the range of 1 to 255. A user can only access those system facilities permitted by his catalogued security level. A vertical partitioning of users, programs and files by application group can be achieved through the group specification in the TIP/30 catalogue.

A user logged on the TIP/30 system with a valid password only has access to those features of the system belonging to his application group for which his security level is high enough to permit him access.

-+*+-

1.6.4 INTERACTIVE UTILITIES OVERVIEW

As a TIP/30 user you will have access to an extensive library of interactive programs to assist in the design, implementation, testing and maintenance of online application systems.

-+*+-

1.6.5 PROGRAM PREPARATION OVERVIEW

For program preparation TIP/30 provides a powerful text editor, an online librarian and a spool file inquiry utility.

The TIP/30 text editor is used to create and modify text elements. These elements may be source programs, job control language, or documents.

The editor's work space is fully recoverable so that no work is lost due to a system failure. This feature alone results in higher morale and greater productivity on the part of the programming staff.

-+*+-

1.6.6 DISPLAY FORMAT PREPARATION OVERVIEW

TIP/30 Message Control System screen formats are created and maintained online by the MSGDEF utility.

Screen formats can be tested on-line using the MSGSHOW utility. The message archiver, MSGAR, can be used to print screen formats, save screen formats in an OS/3 library, and restore screen formats from an OS/3 library.

Creating and maintaining screen formats in a TIP/30 system is a very simple task. User departments can work with the development staff to design 'friendly' screen formats to help ensure on-line system success.

-+*+-

1.6.7 PROGRAM TESTING AND DEBUGGING OVERVIEW

TIP/30 provides utilities to help the programmer get his programs running quickly.

Program dumps may be displayed online by the Post Mortem Dump Analysis program. This eliminates waiting for the central printer and also eliminates the printing of most dumps.

User programs can be traced online by the Interactive Debug Aid (IDA). A programmer can set break points in his program or trace the program one instruction at a time. Errors can be corrected at execution time and the execution of the program can be resumed at a new location. A programmer can find more errors per program test using IDA thus reducing the number of compilations required to get a program implemented.

-+*+-

1.6.8 DOCUMENT PREPARATION OVERVIEW

The TIP/30 Document Generator (DOC) simplifies the maintenance of system documentation.

A programmer maintains the documentation for his programs using the same Text Editor that is used for program source maintenance. The documentation is permanently stored in an element of an OS/3 library. DOC reads the source element to produce a document formatted according to information supplied with the text of the document.

Good documentation can ensure online system success. Timely availability of new information can save a lot of headaches during system implementation.

-+*+-

1.6.9 UTILITIES OVERVIEW

TIP/30 provides a comprehensive set of utility programs. A Query Language program accepts english-like requests for information from indexed files with no programming.

A Remote Job Entry facility gives you the ability to transfer files of information from another computer or from a batch terminal to your TIP/30 system.

An Electronic Mail program allows a user to store full-screen messages in another user's mailbox file.

Allinson-Ross also provides the TIP/30 user with batch utility programs to initialize and back up system files as well as a program to analyze and summarize the system accounting information that is maintained in the journal file.

-+*+-

2. CHAPTER II - FUNDAMENTAL CONCEPTS CONCEPTS

This chapter of the TIP/30 reference manual describes a number of the fundamental concepts that are of interest to all TIP/30 users. This chapter of the TIP/30 reference manual describes fundamental concepts as they apply to the use of the TIP/30 system. These concepts are considered to be essential information for all users of the TIP/30 system. A thorough understanding of this chapter is required to be able to make proper use of the reference manual.

It is assumed in the following discussion that the reader is familiar with the operation of the UNISCOPE (TM) family of terminals. The reader that is not familiar with the operation of the terminal is advised to consult the terminal operator publications available from the manufacturer of the equipment.

2.1 USER IDENTIFICATION AND PASSWORDS USER ID

A user of the TIP/30 system is assigned a "user-id" by the installation administrator. This user-id is intended to be a meaningful pseudo name for the individual. It often takes the form of the individual's last name, his initials, or any character string of up to eight characters that might serve to identify the individual within the user community. For example, a user named "John Q. Doe" might well have a user-id of "DOE" or "JOHN" or "JQDOE".

To be able to enforce system security, the TIP/30 system must be capable of verifying that an individual is who he claims. To that end, there is a "password" associated with every user-id in the system. Each user is assigned an initial password with his user-id. The user must realize that the password is merely an agreement between the user and the TIP/30 system on a means of positively identifying the user. The password may be (and should be) changed at frequent intervals to eliminate the likelihood of unauthorized use of the user-id.

When a user identifies himself to the TIP/30 system, he must give his current password. He may also elect to change his password at this time. If he does change his password, all subsequent logon attempts will require the new password. This process may be repeated as often as deemed necessary by the individual user.

While it is intended that the individual's user-id be known to other users, the password is the first and most fundamental level of security. The password should not be known to anyone but the individual.

The TIP/30 system only requires a password to be given at logon time; the password is not required to run programs or to access files. Once a user has logged on the system his capabilities are well defined by his positive identification.

Associated with each user of the system is their security level. Each user is assigned a security level by the installation administrator. This security level is a numeric value from 1 to 255. In general terms, the security level is a statement about the access the user may have to programs and files. A numerically low security level indicates that the user has a high degree of access. Since there are 255 security levels, users may be easily organized into logical access groups. A user may not access a program or file if their security level does not permit them access.

TIP/30 users may be given membership in groups. These user groups are established by the installation administrator. A user is a member of two implicit groups: his own private group (with the same name as his user-id) and the system-wide group (named "TIPY"). Each user may also be given membership in one or two optional, elective groups. The installation administrator specifies the elective group memberships at the time a user-id is established. These elective group memberships may be changed at any time by the administrator.

Membership in a group grants a user the potential to access programs and files belonging to the group. Actual ability to execute a specific program or to access an individual file depends on the security level of the user with respect to the security level of the program or file in question.

2.2 LOGON AND LOGOFF PROCEDURES LOGON/LOGOFF

A user must LOGON the TIP/30 system in order to identify himself to the system and to establish his capabilities with respect to the TIP/30 security system. There are two methods of logon:

- immediate transmission of a valid user-id and password
- transmission of anything not valid as a user-id and password

If the user transmits (at an idle terminal) a valid user-id and password he will be logged on immediately. For example, a user with a user-id of "FRED" and a password of "QWERTYUI" might logon by transmitting the following:

```
>FRED/QWERTYUI

--OR--

>LOGON FRED/QWERTYUI
```

Note the required character (slash) that separates the password from the user-id. The word "LOGON" is not required in the current version of TIP/30. If specified however, it must be correct as shown.

If the user transmits an invalid user-id/password combination or presses a function key or msg-wait, the TIP/30 system will respond by displaying a screen format with areas in which the user may enter his user-id, password, and (optionally) new password. The user is expected to fill in the appropriate values and press transmit. TIP/30 will then validate the user-id and password.

LOGON AND LOGOFF PROCEDURES

When the user has successfully logged on the system, TIP/30 will display the standard system prompt:

```
TIP?>
```

To log off the TIP/30 system, the user may run the program "LOGOFF" as a response to the standard system prompt. To run the LOGOFF program the user would enter:

```
TIP?>LOGOFF
```

The logoff program will terminate the session and output a display giving the date and time of logoff and various statistics about the session that was just completed (for example: average response time, number of input and output messages to the terminal etc).

The installation administrator may (at his discretion) change the name of the "LOGON" and "LOGOFF" programs. Users are advised to review their installation's LOGON and LOGOFF requirements with the installation administrator when they receive their user-id and initial password.

2.3 TIP/30 COMMAND LINE COMMAND LINE

The TIP/30 system will display on the terminal the standard system prompt after a successful logon and whenever control returns to TIP/30 from a transaction program. In order to run transaction programs, the user must be familiar with the structure of the Command Line. When the user is given the standard system prompt, he is being given an opportunity to enter a command to the TIP/30 system. This command has the following structure:

```
TIP?>transaction-id[,options] [,parameter1] ... [,parameter8]
```

The transaction-id immediately follows the SOE character and is the name of the program the user wishes to run. The transaction-id may be up to eight characters long.

Some transactions (programs) allow the user to enter options immediately following the transaction-id. The options are separated from the transaction-id by a comma or a slash. Options are from one to eight characters which are defined by the particular transaction program.

Separated from the transaction-id (and possibly the option characters) by a blank are the parameters for the program. There may be up to eight parameters supplied to the program from the command line.

These command line parameters represent initial input data for the program. Each parameter is restricted to a maximum of eight characters. The parameters are positional; any omitted parameters are indicated by the presence of a comma separator without any data.

Example:

```
TIP?>PAYROLL MAR,,1982
```

In the example above "PAYROLL" is the transaction-id; there are no command line options; parameter1 is "MAR"; parameter2 is omitted; parameter3 is "1982".

It is important to note that the parameters on the command line are passed to the indicated program as initial data. The programmer who wrote the program is free to interpret the parameters in any manner he chooses. The TIP/30 system merely enforces this command line convention as a simple means of running a program. It is quite reasonable for a program to require no information from the command line (for example, menu-driven full-screen oriented systems frequently require that the user simply enter the name of the menu program).

Many of the utility programs supplied by Allinson-Ross Corporation make extensive use of the command line options and parameters. The documentation for these programs describes the command line parameters recognized by each utility and the command line options required.

2.4 TIP/30 SYSTEM SECURITY SECURITY

TIP/30 provides an extensive security system that may be utilized by the installation administrator to control access to programs and files. The security system cannot be selectively disabled or circumvented. The security system is implemented by entries in the TIP/30 catalogue. The catalogue is a TIP/30 file that is managed by the on-line catalogue manager program. The installation administrator uses the catalogue manager program to enter and modify information in the catalogue.

The catalogue contains entries for all authorized users of the TIP/30 system, all programs that are available on-line, and all files that are accessible on-line.

Each authorized TIP/30 user has an entry in the catalogue that states his user-id, current password, security level and the names of (up to two) elective groups to which he belongs.

There is an entry in the catalogue for each program (transaction-id) which states the group to which the program belongs and the security level required to access the program.

There is an entry in the catalogue for each on-line file in the system. The entry indicates which group has access to the file and the security level required to access the file.

A user may only access programs and files that are defined in the group(s) in which the user is a member. Furthermore, even though the user is a member of a group his access to programs and files in that group is restricted further by the requisite security level.

The catalogue manager program is generally assigned a high security level so that only users with high security (the installation administrator) may change entries in the catalogue. This ensures there is no mechanism whereby the average user can alter security levels or group memberships.

TIP/30 SYSTEM SECURITY

When a user attempts to run a program or access a file, TIP/30 will search for a corresponding program or file entry in the catalogue. The search follows a fixed order, known as 'the order of search'. TIP/30 will search the user's private group, elective group one, elective group two, and the system universal group ("TIPY"). The first program or file entry that is found is considered to be the intended one. The user's security level is then compared to the required security level to run the program or access the file.

If the security level does not imply access, TIP/30 will display a "SECURITY ERROR" message. It is very important to note that the catalogue search will NOT continue past the first entry found in the predefined order of search. If no appropriate entry is found in the catalogue, the user will receive an error message stating that the program or file could not be accessed.

The above description of search order also applies whenever a program attempts to call another program or access files. The catalogue is searched every time an attempt is made to access a program or file. The TIP/30 catalogue file is organized in such a fashion that this order of search may be accomplished very quickly. There is no appreciable overhead asociated with this security mechanism.

3. CHAPTER III - ON-LINE UTILITY PROGRAMS UTILITIES

This chapter of the TIP/30 reference manual contains documentation describing the operation of the on-line utility programs that are supplied with the TIP system.

The documentation is in alphabetical order by transaction name.

The transaction name of a program may be changed by the installation administrator at the site. Users are advised to determine what changes, if any, have been made at their site.

All of the programs that operate under the control of TIP/30 are subject to security restrictions that are maintained by the installation administrator; many of the programs described in this chapter may not be available all users.

Many of the utility programs supplied by Allinson-Ross Corporation provide on-line help information (available through a "help" command recognized by the program or through the HELP utility).

3.1 ACCESS A FILE ACCESS

The ACCESS program is used to assign a Logical File Name (LFN) to a file. Most transaction programs access files via (TIPFCS), through a Logical File Name.

Syntax:

```
ACCESS  aft-name,file-name
```

Where:

aft-name is the Logical File Name assigned to the file. This is the name used in the active file table.

file-name is the catalogue name of a file. If the file to be accessed is an FCS dynamic file, the catalog-name consists of three sections: USER-ID/CATL-ID/FILE-ID, which uniquely identify each FCS file in the catalogue. If the USER-ID is not specified, then the USER-ID used to LOGON to TIP will be used. If left blank then CATL-ID = FILE-ID.

In the following example assume that the user-id 'ARC' was used to log on.

Example:

```
ACCESS  UPDATE,MASTER
```

Assign the logical name 'UPDATE' to the file with the catalogued name of 'MASTER'.

Error Conditions:
TIPFCS errors may be reported.

3.2 DISPLAY ACTIVE FILE TABLE AFT

The AFT program is used to list the files that have been assigned to this terminal.

Syntax:

AFT[/qual]

Where:

qual a command line option available to master level users to display the active file table of a logged on user. May be a user-id or terminal name. If omitted, the active file table for the issuing user will be displayed.

Example:

```
AFT
AFT/MARY
```

Logical	User-id	Catl-id	File-id	Type	Class	Hold	Element
ARCUST	TIPY		ARCUST	MIRAM	S	YES	
QEDTRM11	ARC	TD$AFT	TD$AFT	EDIT	E		

Error Conditions:
The specified process may not be found.

Additional Considerations:

An asterisk ("*") preceding the Logical file name indicates that the file is currently in debug mode (updates ignored).

An asterisk ("*") preceding the File-id indicates that the file is a file that is journalled by TIP/30.

3.3 ALL POINTS BULLETIN APB

The APB program allows a MASTER user to send a message to all currently active terminals. When the message is received, it is prefaced by the USER-ID and terminal name of the sender.

Syntax:

APB [/ALL] text

Where:

[/ALL] command line option indicating that the APB message is to be sent to all terminals (logged on or not).

text is the text of the message (64 characters maximum) to be sent.

Example:

APB SYSTEM WILL BE DOWN AT 3:15 FOR 30 MIN.

Error Conditions:
None.

3.4 ASSIGN A FILE ASG

The ASG program is used to assign a Logical File Name (LFN) to a file. Transaction programs access files by reference to the logical file name. If the ASG program is used to assign a TIP/30 dynamic file that does not exist, the ASG program will create the dynamic file.

Syntax:

```
ASG  aft-name,file-name
```

Where:

aft-name is the Logical File Name assigned to the file. This is the name used in the active file table.

file-name is the catalogue name of a file. If the file to be assigned is an FCS dynamic file, the catalog-name consists of three sections: USER-ID/CATL-ID/FILE-ID, which uniquely identify each FCS file in the catalogue. If the USER-ID is not specified, then the USER-ID used to LOGON to TIP will be used. If left blank then CATL-ID = FILE-ID.

In the following example assume that the user-id 'ARC' was used to log on.

Example:

```
ASG  UPDATE,MASTER
```

assign the logical name 'UPDATE' to the file with the name of MASTER.

Error Conditions:
TIPFCS errors may be reported.

3.5 TIP/30 BASIC INTERPRETER - COMPILER BASIC

BASIC is a supplied program that implements a version of the BASIC programming language. BASIC will interpret and execute programs written in this language. The syntax of BASIC that is supported is defined in an appendix of this reference manual. This section describes the commands that may be given to the BASIC (monitor) program.

The BASIC monitor program recognizes the following commands:

- B - terminate monitor
- C - compile a program
- CP - compile a program and print compilation listing
- D - delete the "object" version of a program
- E - terminate monitor
- H - display help information on terminal
- L - list a program on the terminal
- LC - list all basic programs in TIP catalogue
- M - change screen roll mode
- N - edit a new program
- O - edit an old (existing) program
- P - print program listing
- Q - terminate monitor and LOGOFF system
- R - run a program
- S - save program source in a file

3.5.1 TERMINATE MONITOR — BASIC: bye

This command will terminate the BASIC monitor program in a normal fashion. This is analagous to the "BYE" command in some implementations of the basic language.

Syntax:

Bye

Where:
no parameters required.

Example:

B

Will cause the BASIC monitor to terminate normally.

Error Conditions:
None.

-+*+-

3.5.2 COMPILE BASIC PROGRAM BASIC: compile

The "C" command will compile a basic program. The compilation process actually produces a TIP/30 dynamic file containing pseudo-code (this file is referred to as the "object" file). The pseudo-code is later executed (via the "R" command) by the BASIC interpreter.

Syntax:

Compile [progname] [,group]

Where:

progname The name of the program to compile. Default is last program edited.

group The name of the group to own the object code. Default is the user's private group. A master level user may specify any group name; system level users may specify any group to which they have access; programmer level users may not specify a group.

Example:

C test

Will compile the program named "test" into (by default) the user's private group.

Error Conditions:
The named program may not exist.

-+*+-

COMPILE BASIC PROGRAM WITH LISTING

3.5.3 COMPILE BASIC PROGRAM WITH LISTING BASIC: cp

The compile process (described previously) does not produce a compilation listing. To compile a program and produce a hard copy listing, the user must use the "CP" command. The listing may be sent to the site printer (default) or to an auxiliary terminal printer.

Syntax:

CP [progname] [,group] [,dest]

Where:

progname The name of the program to compile. Default is the last program edited.

group The name of the group that is to own the object code file. Default is the user's private group.

dest The desired printer. Default is PRNTR (site printer). Other choices include: AUX1 etc.

Example:

```
CP test,TIP$Y$,aux1
```

Would compile the program named "test" into the group TIPY and generate a compile listing at the terminal auxiliary print device on interface 1.

Error Conditions:
The named program may not exist. The named printer may not be available.

-+*+-

3.5.4 DELETE BASIC OBJECT FILE BASIC: delete

The delete command will scratch the dynamic file containing the BASIC object code (pseudo-code) for a basic program. The object code file would have been produced by the BASIC compilation process.

Syntax:

Delete [progname] [,group]

Where:

progname The name of the program object file to delete. Default is the last edited program.

group The group owning the specified object file. Default is the user's private group.

Example:

Del test

Will delete the object code file for the program "test" from the user's private group.

Error Conditions:
The object file for the program may not exist in the specified group.

-+*+-

3.5.5 TERMINATE THE BASIC MONITOR BASIC: end

This command will cause the BASIC monitor to terminate normally.

Syntax:

End

Where:
No parameters required.

Error Conditions:
None.

-+*+-

3.5.6 DISPLAY BASIC PROGRAM HELP INFORMATION BASIC: help

This command will cause the BASIC monitor to display help information which will summarize the recognized command syntax.

Syntax:

Help

Where:
No parameters required.

Error Conditions:
The help information may not be available or may have been deleted.

-+*+-

3.5.7 LIST BASIC PROGRAM ON TERMINAL BASIC: list

This command will list on the terminal the source for a basic program.

Syntax:

List [progname]

Where:

progname The name of the program source to list on the terminal. Default is the last edited program.

Example:

L test

Will list the source for program named "test" on the user's terminal.

Error Conditions:
The named program may not exist.

-+*+-

3.5.8 LIST BASIC PROGRAMS IN TIP CATALOGUE BASIC: lc

This command actually calls the TIP/30 catalogue manager program to list entries in the catalogue for BASIC language programs. The executable pseudo-code for BASIC programs is stored in a TIP/30 dynamic file. The LC command will display all entries in the catalogue that are available to the user who issued the LC command.

Syntax:

LC [progname] [,group]

Where:

progname Optional program name (prefix specification allowed).

group Optional group name (prefix specification allowed).

Example:

LC

Error Conditions:
The user may not have sufficient security clearance to run the catalogue manager program.

-+*+-

3.5.9 CHANGE SCREEN ROLL MODE — BASIC: mode

This command will change the way in which BASIC manipulates the user terminal. The default mode is "roll" mode. This means that messages from BASIC will appear at the bottom line of the CRT and, in effect, push up any existing information on the CRT. An alternative is "scroll" mode. In this mode, output from BASIC appears on the next line of the CRT; when the last line has been used, the next output will appear on the top line. In "scroll" mode, all prompts from BASIC will be issued on the bottom line of the CRT.

Syntax:

Mode [option]

Where:

option A choice of "roll" or "scroll". Default is "roll".

Example:

M scroll

Error Conditions:
None.

-+*+-

3.5.10 EDIT A NEW BASIC PROGRAM — BASIC: new

This command will enable the user to edit a program. The BASIC monitor will attempt to find existing source for the program (as named). If no existing source is found, the user will begin with an empty edit work space. The BASIC monitor will call the TIP/30 TEXT EDITOR (QED) to allow the user to edit the program. The user should be familiar with the operation of the text editor before attempting to edit a BASIC program. When the user has completed editing the program it is important to terminate the text editor session with the "E" command (see section on QED). The "new" command will set the name of the "current" program which may be used as a default in some other BASIC monitor commands. BASIC program names are limited to a maximum of eight characters.

Syntax:

New progname

Where:

progname The name of the BASIC program to be edited.

Example:

N mytest

Error Conditions:

The user may not have sufficient security clearance to run the TIP/30 text editor.

-+*+-

3.5.11 EDIT EXISTING BASIC PROGRAM — BASIC: old

This command is identical to the previously discussed "New" command. It is provided primarily for compatibility with other implementations of the BASIC language.

Syntax:

Old progname

Where:

progname The name of the BASIC program to be edited.

Example:

O myprog

Error Conditions:
The user may not have sufficient security to run the TIP/30 Text Editor.

-+*+-

3.5.12 PRINT BASIC PROGRAM LISTING BASIC: print

This command will enable the user to generate a hard-copy print of a BASIC program. The BASIC interpreter calls the TIP/30 Librarian (TLIB) to produce the printout. The program may be either an existing edit work space or may have been filed in a library.

Syntax:

```
Print  [progname,,dest]

OR     Print  file,prog[,dest]

OR     PA     [progname]
```

Where:

progname The name of a BASIC program. If omitted, the current program name (from a previous "New" or "Old" command) will be assumed.

dest The name of the desired printer (eg: AUX1 etc). Default is "PRNTR".

file The catalogued file name of the library containing the program to print.

prog The name of the library element to be printed.

Example:

```
P myprog,,aux1
```

Error Conditions:
Various errors associated with the TIP/30 librarian may be displayed.

-+*+-

3.5.13 TERMINATE BASIC MONITOR — BASIC: quit

This command will cause the BASIC monitor to terminate interaction with the user. If the BASIC monitor had been executing at stack level one the user will be logged off the TIP/30 system.

Syntax:

Quit

Where:

No parameters required.

Example:

Q

-+*+-

3.5.14 RUN A BASIC PROGRAM — BASIC: run

This command will execute a BASIC program. The program must have been previously compiled.

Syntax:

Run [progname]

Where:

progname The name of the BASIC program to execute. Default is the current program name (as set by a previous "Old" or "New" command).

Example:

R spacewar

Error Conditions:
The program object file may not be found (it might have been deleted or the program was not compiled).

-+*+-

3.5.15 DIRECT EXECUTION OF BASIC PROGRAMS BASIC: run

A BASIC program may be executed by using the "run" command in the BASIC monitor. It is also possible to enable a basic program to be executed directly from the TIP command line.

To do this, the user must make an entry in the TIP/30 catalogue (see section on the catalogue manager program "CAT").

The entry in the catalogue would specify the transaction name as the name of the basic program. The entries for the program (eg: LOADM=, WORK=, CDA= etc) would correspond exactly with the entries for the BASIC interpreter ("BINT").

When the user runs the transaction with the name of the basic program, the BASIC interpreter is loaded. The interpreter detects that it was called from the command line (as opposed to being called from the BASIC monitor) and assumes that the user wants to simulate entering a RUN command with the program name equal to the transaction id.

When the basic program ends, the user will be prompted again by the TIP command processor.

An example of a program entry in the TIP/30 catalogue for a basic program called "spacewar" is as follows:

Example:

```
>PROG SPACEWAR GRP=TIP$Y$ FROM=TIP$Y$/BINT ENTER=YES.
```

-+*+-

3.5.16 SAVE A PROGRAM IN A LIBRARY — BASIC: save

This command will enable a user to save a BASIC program in an OS/3 library. For editing, the source for a BASIC program is retained in a text editor work file; to save the source in a more permanent place, the user would use the "SAVE" command.

Syntax:

Save file

Where:

file The catalogued name of the OS/3 library which is to contain the current program. The BASIC monitor will create a library element containing the source of the current program (set by a previous "Old" or "New" command).

Error Conditions:
None.

-+*+-

3.6 BATCH TERMINAL COMMAND PROCESSOR BCP

The TIP/30 Batch Terminal Command Processor (BCP) is a system level transaction program. BCP is activated whenever an input message arrives from an idle batch terminal. Typical batch terminals are IBM-3780, IBM-3741, IBM-2780, UDS-2000, etc.. Batch terminals are file oriented devices. They always send/receive complete files of data. A file may consist of one or more records, and take several communications I/O's to complete.

BCP's functions include:

- user logon and logoff
- interpreting and executing commands,
- running user programs.

The user interacts with BCP from a terminal by preparing and transmitting files of commands and/or data, and by receiving files from the host computer system.

3.6.1 SUMMARY OF BCP COMMANDS BCP

All BCP commands must begin in column 1 of the input message and begin with an @ (commercial at-sign). The following is a summary of the valid BCP commands available in this release:

Command	Function
CALL	call user program
DELETE	delete spool files
FIN	logoff the system, disconnect
FORK	call user program in background
IN	transmit from terminal to system spool file
LOGON	logon to the system
MSG	send message to computer operator
PRINT	transmit from system spool file to terminal
PUNCH	transmit from system spool file to terminal
QUEUE	display spool queue
RECEIVE	transmit from terminal to system file
RUN	start up a batch job on the system
SEND	transmit from system file to terminal
SUBMIT	copy to RBPIN queue and call RB symbiont

-+*+-

3.6.2 BCP KEYWORD SHORTFORMS BCP

Keyword	-	Alternate spelling
COMPRESS		CO
COPIES		COPY
DELETE		DE
FILEID		LABEL, LABLE
FIN		LOGOFF
FORM		FO
IN		IN
INLINE		INL
PAGE		PA
PRINT		PR, OUT
PUNCH		PU
QUEUE		QU
RECEIVE		RE
SEND		SE
STEP		STE
STOP		ST
USING		USE

-+*+-

3.6.3 BCP COMMAND LANGUAGE BCP

Once the user has successfully logged on he may begin transmitting BCP commands. All BCP commands are of a similar format. The commands begin in column one (1) with an @. Next is the command itself. Next are any positional parameters separated by commas. Next are any keyword parameters. The entire command may be terminated by a period.

Syntax:

```
@COMMAND   Parm1,..,Parm4 KEY1=val1,..,KEYn=valn
```

Where:

@ all commands begin with an at-sign '@'.

COMMAND is the BCP command being used. (each command is documented in later sections)

PARAMETERS are zero thru four optional fields which are used to specify variable information.

KEYl= keyword parameters as required by each command. The keywords must be spelled in full (the defined alternate spelling may be used).

Optional parameters are enclosed in square brackets when being documented in this section. Example: [filename].

Example:

```
@PRINT  ALL JOB=TESTBCP COPIES=2
```

An attempt is made to support Univac's RBP style of commands. If an equivalent RBP command is supported it will be listed at the end of each section describing the BCP command. Commands may begin with "/R" instead of "@". This will have the same effect as specifying INLINE=YES as a parameter.

-+*+-

BCP STATUS MESSAGES

3.6.4 BCP STATUS MESSAGES BCP: ack/nak

All informational and status messages returned by BCP will begin with one space and then the characters BCP:. (' BCP:'). The next three letters will be ACK if this is an affirmative reply or NAK if it is a negative reply.

Following the ACK/NAK status is some informational message. Following the message will be the time of day in the format HH:MM.

For example:

```
' BCP:ACK LOGON ACCEPTED FOR RJNORMAN SITE:ARC  17:30'
```

-+*+-

3.6.5 USER PROGRAM EXECUTION BCP: call

To execute a user written program use the CALL command. The transaction code may be followed by one or more spaces and up to six parameters. These parameters make up the called program's Command Line. Command line parameters may not be more than eight characters in length and are separated by commas (,), slashes (/), or one or more blanks. When the user program receives control, these parameters may be retrieved from the CDA (continuity data area). Alphanumeric parameters are left justified and space filled. Numeric parameters are right justified and zero filled. The parameters are stored in eight consecutive eight byte fields.

Syntax:

```
@CALL PROGNAME   Param-1,...,Param-4
```

Where:

PROGNAME is the name of the program to be activated. This name is the catalogued name.

Example:

```
@CALL TAX  WARD6
```

This command calls the program 'TAX' and passes the parameter 'WARD6' to the program in the first eight bytes of the continuity data area.

NOTE: the input file is not cleared, the user program must do that.

-+*+-

3.6.6 DELETING PRINT FILE — BCP: delete

This command must supply either a form-name or job-name of the printout which is to be deleted from the spool file. This printout must have already been created and placed in hold mode in the spooler.

The user-id must match either the job name or the form name before you are allowed access to the print file.

Syntax:

@DELETE [q-name] [ALL] [FORM=formname] [JOB=jobname] [STEP=n]

Where:

q-name optional positional paramter 1. This is the spooler queue name which is to be deleted. It will default to the printer spool (PR). Other choices include: RDR, PU, RBPPR, RBPPU.

ALL will cause all spool files which match the given criteria to be deleted.

FORM=formname is the form name of the printout wanted.

JOB=jobname is the JOB name wanted.

STEP=n is the job step number to delete.

Example:

@DELETE FORM=STAND1 JOB=MYCOMPLE

This command deletes the printout with the form name of 'STAND1' and job-name of 'MYCOMPLE'.

If INLINE=YES was not specified then the terminal will receive a message for each spool file deleted.

-+*+-

3.6.7 TERMINATING BCP BCP: fin

The FIN command is used to terminate BCP and log the user off the system.

Syntax:

@FIN

no parameters are required.

Example:

@FIN

Reply message:

LOGGED OFF - PLEASE DISCONNECT

RBP equivalent /RLOGOFF

-+*+-

3.6.8 BACKGROUND PROGRAMS BCP: fork

TIP/30 provides the facility to execute programs in the background environment. A background program is not associated with any terminal and therefore cannot solicit input. It may, however, use all other TIP/30 functions (ie. call other programs, send output to any terminal, call TIPTIMER to suspend itself, etc). To start a program in a background environment use the FORK command.

Example:

```
@FORK POSTAR   6
```

- In response to the above command, BCP would call TIP/30 to start the program 'POSTAR' in a background environment. BCP would then continue to interact with the user while the background program executes.

Reply message: - none, BCP continues to read input file.

-+*+-

3.6.9 CREATE INPUT READER SPOOL BCP: in

This command supplies the label which is to be assigned to an input reader spool file. BCP will create the spool file and store the incoming data records in it. This command would normally be the first record of a data file. The remainder of the data file (up to end-of-file or another command) is stored in the input reader spool file. At end of file a '/*' record is written to the spool file.

Syntax:

@IN LABEL='label' [SIZE=n] [RETAIN=Y] [INLINE=Y] [XPAR=YES]

Where:

LABEL='label' is the label of the input reader file to be created.

One of user-id, group names etc.. must match the prefix of the label or it will be considered a security violation.

NOTE: if an input spool file of the same label already exists then it will be deleted before this current one is stored.

SIZE=n n is the input record size. Default size is 128.

RETAIN=Y the spool file will be retained after being read by the batch job. The default value is R=N.

INLINE=Y the data follows immediately in this file. Normally the input file is cleared and BCP will tell the terminal to send the data file.

XPAR=YES send data in transparent mode.

Reply message: (if INLINE=Y was not specified)

READY TO RECEIVE label - PLEASE XMIT DATA

When the terminal has received the above message it should then send the data file.

RBP format // DATA FILEID=????? followed by the data, followed by // FIN.

CREATE INPUT READER SPOOL

Example:

Terminal	Host
@IN LABEL='MAILUPDT',SIZE=80	
	READY TO RECEIVE MAILUPDT - PLEASE XMIT
NAME1 ADDRESS1	
NAME2 ADDRESS2	
NAME3 ADDRESS3	
	3 RECORDS TRANSFERED. FILE: MAILUPDT

- This stores 3 data records in the input rdr spool 'MAILUPDT'.

If any error occurs while opening the spool file one of the following messages is sent to the terminal:

ERROR: CREATING SPOOL FILE COMMAND: IN SITE: site-id

ERROR: NO MEMORY FOR SPOOLER COMMAND: IN SITE: site-id

ERROR: SECURITY VIOLATION COMMAND: IN SITE: site-id

-+*+-

3.6.10 USER LOG-ON PROCEDURE BCP: logon

Traffic from an idle batch terminal (idle means no-one logged on), will cause BCP to be loaded for that terminal.

Once activated, BCP will consult the TIP/30 catalogue to see if the terminal name has been catalogued as a user-id. If the terminal name is a valid user-id then no logon is required. If not BCP will read the in-coming data.

The first record must contain

```
@LOGON user-id,password[,JOB=jobid][,SIZE=n][,MODE=type]
```

The 'user-id' is verified in the TIP/30 catalogue. If valid then the host console is notified of the successful logon (TI#67 message). If not valid then an error message will be sent back to the terminal attempting connection.

Where:

user-id as setup in the TIP/30 catalogue file.

password used for security purposes.

JOB=jobid is stored in the first eight (8) bytes of the CDA and may be accessed by user written programs which may be called by BCP. This 'jobid' is used later as the default JOB= value for other commands.

SIZE=n is the new default record size. The remote terminal must be able to handle a message of this size. The communications software on the host computer (ie. ICAM) must have large enough network buffers. This value is passed in bytes 9 thru 16 of the CDA as a right justified, zero-filled number.

COMPRESS=Y use 3780 data compression for non-transparent data transmission.

'N' - do not do any data compression.

TIMEOUT=n the time in minutes for which BCP is to wait for the next command. When BCP has nothing to do it will go to sleep for a given time interval. If no more commands arrive then BCP will automatically log the user off. The default value is the TIMEOUT

value given in the TIP/30 generation.

OBUFR=n the maximum output buffer size to be used. This overrides the MODE defaults.

FF=NO this will cause BCP to not send the form feed character as a home paper command when printing. An Escape 'G' will be sent instead. The default is to use the FF character.

MODE=type is the terminal type. BCP needs to know if it is an IBM-2780, IBM-3741, IBM-3780 or UDS-2000. The default is UDS-2000. If the remote terminal is an IBM-2780 then specify MODE=IBM-2780.

After a successful logon a TI#67 message is sent to the operator console. The rest of the input is read and ignored. Then the following message is sent back to the remote terminal.

LOGON ACCEPTED FOR user-id SITE IS site-id

Other error message may include:

INVALID LOGON FORMAT
INVALID USER-ID
INVALID PASSWORD

-+*+-

3.6.11 MODES OF OPERATION **BCP: mode**

The following table gives the MODE value and corresponding default actions unless overidden by keyword parameters.

		OUTPUT			INPUT			
MODE	Data Compress	Buffer Size	Lines / msg	Record size	Buffer Size	Lines / msg	Record Size	
IBM-2780	No	160	1	136	512	50	80	
IBM-3780	Yes	512	21	136	512	50	128	
MOHAWK	Yes	512	21	136	512	50	128	
IBM-3741	Yes	512	21	136	512	50	128	
UDS-2000	Yes	512	21	136	512	50	128	
NIM-3305	Yes	512	500	136	512	500	128	(DATAPAC)
DCT-1000	No	160	1	160	80	1	80	
DCT-2000	No	160	1	160	80	1	80	

-+*+-

3.6.12 SEND COMPUTER OPERATOR A MESSAGE BCP: msg

The MSG command allows a terminal user to send the host computer operator a message. When the message is received, it is prefaced by the USER-ID and terminal name of the sender.

Syntax:

@MSG .....TEXT.....

Where:

TEXT is the message (50 characters maximum) to be sent.

Example:

@MSG INVENTORY UPDATE IS COMPLETE.

-+*+-

3.6.13 TRANSMIT PRINT FILE BCP: print

This command supplies the form-name of the printout which is to be sent to the requesting terminal. This printout must have already been created and placed in hold mode in the spooler. The remainder of the communications file is read to clear the line, and then the print file is transmitted to the remote terminal.

The user-id must match either the job name or the form name before you are allowed access to the print file.

Syntax:

```
@PRINT [q-name] [ALL]     [FORM=formname] [JOB=jobname]
               [STEP=step#] [PAGE=n]        [STOP=n]
               [COPY=n]     [COMPRESS=N]    [SIZE=size]
               [FF=Y/N]     [DELETE=YES]
```

Where:

All parameters must fit on one input line. All parameters are optional, but you must specify at least one of FORM= or JOB=.

q-name optional positional parameter 1. This is the spool queue name which is to be transmitted. It will default to the printer spool queue (PR). Other choices include: RDR, PU, RBPPR, RBPPU.

ALL will cause all spool files which match the given criteria to be sent.

COMPRESS=N indicates that the data is not to be compressed.

COPY=n 'n' is the number of copies to print.

DELETE=YES this will cause BCP to delete the spool file after transmission.

FORM=formname is the form name of the printout wanted.

FF=NO this will cause BCP to not send the form feed character as a home paper command when printing. An Escape 'G' will be sent instead. The default is to use the FF character.

TRANSMIT PRINT FILE

JOB=jobname is the JOB name wanted

PAGE=n 'n' is the starting page number.

SIZE=size is the size of records to be sent.

STEP=n 'n' is the step number of the job for the printout

STOP=n 'n' is the stopping page number.

Example:

```
@PRINT ALL FORM=MAILLIST
```

This command transmits the printout with the form name of 'MAILLIST' to the terminal.

If the printout does not exist BCP will send back

```
NO DATA AVAILABLE FOR JOB=jobxx FORM=formxx
```

-+*+-

3.6.14 TRANSMIT PUNCH FILE BCP: punch

This command supplies the name of the punch file which is to be sent to the requesting terminal. This punch file must have already been created. The remainder of the communications file is read to clear the line, then the punch file is transmitted to the remote terminal.

The user-id must match either the job name or the form name before you are allowed access to the punch file.

Syntax:

```
@PUNCH [q-name] [ALL]    [LABEL=formname] [JOB=jobname]
                [STEP=step#] [PAGE=n]          [STOP=n]
                [COPY=n]     [COMPRESS=N]      [SIZE=size]
                [DELETE=YES]
```

Where:

All parameters must fit on one input line. All parameters are optional, but you must specify at least one of FORM= or JOB=, or LABEL=.

q-name optional positional parameter 1. This is the spool queue name which is to be transmitted. Default is the punch spool queue (PU). Other choices include: RDR, PU, RBPPR, RBPPU.

ALL will cause all spool files which match the given criteria to be sent.

COMPRESS=N indicates that the data is not to be compressed.

COPY=n 'n' is the number of copies to print.

DELETE=YES this will cause BCP to delete the spool file after transmission.

LABEL=label is the spooler label of the punch file.

JOB=jobname is the JOB name wanted

PAGE=n 'n' is the starting card-image number.

TRANSMIT PUNCH FILE

SIZE=size is the size of records to be sent.

STEP=n 'n' is the step number of the job for the printout

STOP=n 'n' is the stopping card-image number.

Example:

```
@PUNCH ALL LABEL=MAILLIST
```

This command transmits the punch file with the name of 'MAILLIST' to the terminal.

If the punch file does not exist BCP will send back

```
NO DATA AVAILABLE FOR JOB=jobxx FORM=formxx
```

-+*+-

3.6.15 DISPLAYING PRINT FILE QUEUE BCP: queue

This command may supply either a form-name or job-name of the printouts which are to be summarized to the terminal. The printouts must have already been created and placed in hold mode in the spooler. The job name, job step number, program name, form name, and number of pages in the spool file is listed back to the terminal for each spool file which satisfied the search criteria.

Syntax:

@QUEUE [FORM=formname] [JOB=jobname]

Where:

FORM=formname is the form name of the printout wanted.

JOB=jobname is the JOB name wanted

Example:

@QUEUE FORM=STAND1

This command summarizes the printouts with the form name of 'STAND1'.

In reply to this command a sequence of records will be sent back. Each record is as follows.

JOB=xxxxxx STEP=nn PROG=xxxxxx FORM=xxxxxx PAGES=nnnnn LABEL=label

-+*+-

3.6.16 SEND DATA FILE TO HOST BCP: receive

This command supplies the file name and an optional LFD name of a data file into which BCP is to store the incoming data records. The remainder of the input file is read to clear the line.

Syntax:

@RECEIVE FILE [FILEID=lfd] [INLINE=YES]

Where:

FILE is the file name as generated in TIP

FILEID=lfd is an LFD name to override that known to TIP.

This will modify the file (as it was generated into TIP/30) to now refer to the specified LFD name.

INLINE=YES data follows inline.

BCP will reply with:

READY TO RECEIVE label - PLEASE XMIT DATA

Example:

```
Terminal                          Host
------------------------          ----------------------------
@RECEIVE UDS80,FILEID=MAILUPD

                                  READY TO RECEIVE UDS80 - PLEASE XMIT
NAME1     ADDRESS1
NAME2     ADDRESS2
NAME3     ADDRESS3
                                  3 RECORDS TRANSFERED. FILE: UDS80
```

- this stores 3 data records in the data file 'MAILUPDT'.

-+*+-

3.6.17 RUN BATCH JOB BCP: run

The JOB command allows a user to RUN a batch job from a terminal.

Syntax:

@RUN JOB-NAME,,parameters

Where:

JOB-NAME is the name of the batch job to be run.

Reply message:

JOB REQUESTED: JOBNAME,,Parameters....

Example:

@RUN AP01,,MONTH=MARCH

-+*+-

3.6.18 SEND DATA FILE TO TERMINAL BCP: send

This command supplies the TIP file name and optionally the LFD-name of a data file which is to be sent the the requesting terminal. This file must be defined to TIP and should contain valid data. The remainder of the communications file is read to clear the line, and then the data file is transmitted to the remote terminal.

Syntax:

```
@SEND FILE [FILEID=lfd] [XPAR=YES]
```

Where:

FILE is the file name as generated in the TIP system.

FILEID=lfd is an optional LFD name to override the name known by TIP/30.

This command will actually modify the file as it was generated into TIP/30. If the file is later used without FILEID then it has the name as given with the last FILEID.

XPAR=YES send data in transparent mode.

Example:

```
@SEND UDS80 FILEID=MAILMST
```

This command transmits the file MAILMST using the DTF of UDS80. UDS80 was generated into TIP, and MAILMST was specified in the TIP job control.

-+*+-

3.6.19 SUBMIT REMOTE BATCH JOB BCP: submit

The SUBMIT BCP command allows a user to write data into the remote batch input reader queue (RBPIN) and call the RB symbiont to process it.

This command operates similiar to the IN command. At end of file a '// FIN' record will be appended to the spool file.

To use this facility the OS/3 supervisor must be generated with SPOOLING=REMOTE.

Syntax:

```
@SUBMIT [LABEL=label] [INLINE=YES]
```

Where:

LABEL=label spooler file label

INLINE=YES data folows inline, begining with a JOB card.

Reply message:

12 RECORDS TRANSFERRED.

Example:

```
@SUBMIT INLINE=YES.
// JOB TEST
// OPR 'HELLO WORLD'
/&
// FIN
```

-+*+-

3.6.20 USING BCP INTERACTIVELY BCP

BCP may also be called from an interactive terminal such as a UTS-400. The transaction code is BCP. Interactive BCP may be used to send/receive data files to/from a batch terminal. To do this BCP must not be concurrently running for the batch terminal.

A summary of available commands follows:

Command	Function
END	end BCP
IN	receive spool file from batch terminal
MODE	specify terminal type
PRINT	transmit spool file to batch terminal
RECEIVE	receive file from batch terminal
SEND	transmit file to batch terminal
USING	specify terminal name to SEND/RECEIVE

All of the commands (SEND, RECEIVE, PRINT, & IN) have the same format as described in previous sections with the following exceptions:

- The commands should be entered on the interactive terminal without the '@' character.
- One additional keyword parameter may be supplied:

  USING=terminal-id

Where 'terminal-id' is the name of the batch terminal to be involved in the data transfer process.

Example:

```
IN FILEID=BCPINPUT USING=BSC2
```

After successful initiation BCP will display

```
BCP STARTED ON xxx
```

If BCP could not be started (terminal not up or does not exist or is already busy) the following message would be received:

```
BCP NOT STARTED
```

All error and diagnostic messages are sent back to the initiating terminal as unsolicted messages (ie. you must press MSG WAIT).

-+*+-

3.6.21 ICAM GENERATION CONSIDERATIONS BCP: icam

ICAM must be generated with network buffers large enough to hold the largest message which may be generated by BCP. This seems to be required by ICAM itself. Otherwise you may get NO NETWORK BUFFERS or no data will be transmitted to the terminal.

You should use disk queueing for all of the terminal queues for Bi-Sync terminals.

Place LINE definitions for Bi-Sync terminals at the end of the ICAM gen. This should be done so as not to interfere with interactive terminals.

-+*+-

3.6.22 SAMPLE ICAM **BCP: icam**

```
COMMCT
NET1      CCA         TYPE=(TCI),FEATURES=(OPCOM,OUTDELV)
          BUFFERS     20,192,4,ARP=35,STAT=YES
LN08      LINE        DEVICE=(UNISCOPE),TYPE=(9600,SYNC),ID=08,STATS=YES
ARC1      TERM        ADDR=(21,51),FEATURES=(U400,1920),AUX1=(COP,73),     X
                LOW=DQFILE,MEDIUM=MAIN,HIGH=MAIN
ARC2      TERM        ADDR=(21,52),FEATURES=(U400,1920),AUX1=(COP,73),     X
                LOW=DQFILE,MEDIUM=MAIN,HIGH=MAIN
LN09      LINE        DEVICE=(BSC,516,EBCDIC),TYPE=(9600,SYNC),ID=09 .
BSC1      TERM        FEATURES=(BSC,512,MULTI,PRIMARY,TRANSPARENT,0,512), X
                LOW=DQBSC,MEDIUM=DQBSC,HIGH=DQBSC
DQFILE    DISCFILE    FILEDIV=4
DQBSC     DISCFILE    FILEDIV=4
TCIFLE    DISCFILE    MSGSIZE=2560
          ENDCCA
          MCP         MCPNAME=C3,                                          X
                CACH=(08,9600,SYNC),                                       X
                CACH=(09,9600,SYNC)
END
```

-+*+-

3.7 TIP/30 CATALOGUE MANAGEMENT CAT

The TIP/30 catalogue file contains the information needed to execute online programs. The catalogue is organized as a hashed (or calced) file containing three record types:

User-id These records identify valid users of the online system.

Program These records describe valid online programs (transactions). A program record identifies all the run time requirements of the program. (ie: load module name, MCS size, Work size etc).

File These records describe valid online files. A file record in the catalogue links the logical file name (the name used in a program) with the LFD name (the name used in the operating system).

Each record in the catalogue has a 25 character key. This key is composed of four fields which together uniquely identify each record in the catalogue. Duplicate keys in the catalogue are not allowed. The four fields that form the catalogue key are as follows:

Group This is the name of the group to which the item belongs. In the case of a user-id record, this field will contain the same value as the user-id. Any items (programs or files) catalogued in a group with the same name as a user-id record, are considered to be in that user's private group.

Id This is the id of the program (TRID) or file (FID) described by the catalogue record. If the catalogue record is a file type record, then this field would contain the FID (file id) of the file. If the record is a program type, then this field would contain the TRID (transaction id or program name). If the record is a user-id type, then this field is not used.

Elt this field is only used if the catalogue record describes a TIP/30 dynamic file. Dynamic files have a two level name (id/elt), and this field is used to contain the element name of the dynamic file.

Type this is the type code of the catalogue record. There are five type codes used as follows:

U - user-id record
P - program (TRID) record
F - file (FID) record
- file records in the catalogue may also be refered to by the class code which identifies the type of file as follows:
S - system (data management) file record
D - dynamic file record
E - dynamic edit file record

Since the catalogue is a hashed file (ie. no index) it cannot be processed in any order other than on a block by block basis (ie: block 1, 2, 3...). To produce an ordered listing of the file, either on-line or in batch, it must be entirely scanned at the block level to extract the desired records, then those records selected must be sorted to produce the desired listing. An understanding of this fact will help the user obtain listings of records in the catalogue in as short a period of time as possible.

The catalogue is implemented using a two partition SAT file, the second partition is used to store the screen formats developed using the TIP/30 Message Control System (MCS). The catalogue file should never be processed by programs other than those provided with the TIP/30 system. An exception is the operating system file dump/restore program (DMPRST).

3.7.1 ON-LINE CATALOGUE MANAGER CAT

TT$CAT (usually referenced by the transaction id of CAT) is a system utility program which displays, updates, adds, and deletes records in the TIP/30 catalogue. The on-line catalogue manager operates interactively in a free-format command mode. The user enters a command code, positional parameters (to identify the required item), and keywords to supply values for the item.

CAT also accepts command line parameters for the list function.

Syntax:

Command P1,P2,P3,P4 key1=v1,key2=v2,key3=v3,...,keyn=vn.

Where:

Command is the function to be performed. The valid functions are:

List List the TIP/30 catalogue on the interactive terminal.

Write Write the TIP/30 catalogue to a source element within a library.

DELete Delete catalogue record(s).

Prog Create/Update a program record.

User Create/Update a User-id record.

File Create/Update a file record.

End Terminate execution of the online catalogue manager.

P1,P2,P3,P4 are positional parameters supplied with the command. For the Prog, User, and File commands, only positional parameter one is used, positional parameters two, three and four must not be specified. For the List, Write, and DELete commands positional parameters one through four are used to identify the item(s) to be processed.

key1=,...keyn= are keyword parameters that are used to supply additional information.

-+*+-

3.7.2 SECURITY LEVEL SPECIFICATION CAT: security

There is one keyword common to each catalogue entry which is used by the TIP/30 Security system.

SECurity=level - determines the security level of the catalogue record.
- entered as a numerical value from 1 to 255.
- also may be entered as a reserved word as follows:

=TECH - sets SECURity=1
=MAST - sets SECURity=9
=SYST - sets SECURity=19
=PROG - sets SECURity=29
=APPL - sets SECURity=32

SECurity= specifies a value in the range of 1 to 255 which defines the security level of the item. The lower the security number the higher the priority. The security of an item may also be specified by using a reserved word in place of the numeric security level number. The five reserved words that are allowed with the security= keyword are TECH, MAST, SYST, PROG, APPL.

The following table lists the security ranges provided and shows the reseved word equivalent. This table also indicates the functions allowed by the the online catalogue manager for users within each range.

Security Level	Reserved Word Equivalent	Remarks (allowable functions)
1	TECH	Master User of the highest priority -may create other Master Users. -may list, create, update, delete any record in the catalogue.
2-9	MAST	Master User -may list, create, update, delete any record in the catalogue except user-id records of master users
10-19	SYST	System User -may list, create, update, delete any catalogue record in a group to which the user has access -may create user-id records, but may not allow access to a group that the system user does not have access to.
20-29	PROG	Programmer -may list any catalogue record in a group to which the user has access -may create, update, delete program records in the user's private group -not allowed to create, update file or user-id record.
30-255	APPL	Application user -not allowed to manipulate the catalogue in any manner.

SECURITY LEVEL SPECIFICATION

When cataloguing files and programs assign them a security level numerically equal to or less than the security level of the user who will be accessing them.

When a user attempts to access a file or to run a program the following security check takes place for these items:

```
IF user security is less than item security
 THEN  deny access.

IF user security is not lower than item security
  IF the item is time-locked,
     THEN  deny access
        ELSE  allow access.
```

-+*+-

3.7.3 DEFINITION OF CATALOGUE GROUPS CAT: security

The concept of grouping in the TIP/30 catalogue must be understood to properly utilize the catalogue and its features. Every program and file is catalogued within a group (a program or file may appear in several groups).

Every user of the system has a list of groups to which the user has access.

When a user requests access to a program or file, each group to which to user has access is consulted to determine if the requested item exists in that group. The order in which the groups are consulted is known as the catalogue order of search and is defined as follows:

Private This is the users private group. Any items (programs or files) catalogued in a group with the same name as the user-id name are considered to be in the private domain of the user.

Group 1 This is an optional (elective) group and is consulted if the user-id record was created using the GRouP=(a,b) keyword parameter. The first subparameter of the GRouP= keyword is used to name the group.

Group 2 This optional (elective) group is similar to Group 1 above. It is the third group to be consulted when looking for an item. To specify this group name, the second subparameter of the GRoup= keyword of the User command is specified as the group name.

System This group is the last group consulted in the order of search. The name of the group is TIPY and it is available to all users.

It is important to note that grouping and the order of search concepts only control which item in the catalogue is selected for a given user. It is the security level of the item that controls whether the item may be used.

It is through the specification of groups and security levels that the TIP/30 security mechanism is able to control user access to programs and files.

-+*+-

3.7.4 CATALOGUING A USER-ID CAT: user

The catalogue manager command USER either creates or modifies a user-id record in the TIP/30 catalogue.

Syntax:

```
User    user-id        . required positional parameter
        ACcounTs=      . list of valid accounts for this user
        GRouPs=        . user group 1 and 2
        MaXUSers=      . max concurrent uses of this user-id
        MENU=          . screen to be used as menu
        PassWorD=      . password to protect this User-id
        PROG=          . program to auto run at logon time
        SeaRCH=        . full catalogue search or just TIP$Y$
        SECurity=      . user security level
```

Where:

user-id Required and must be specified as the first parameter of the USER command. This is a positional parameter and is used to identify the name of the user-id being created or updated.

ACCOUNTS=(a,b,c) specifies a list of accounts that can be used by this user when logging on to the system. A maximum of 16 account numbers can be specified. Account numbers are a maximum of four characters long.

If a user is assigned account numbers then the user must supply one of the valid account numbers at logon time. If this is not done the user will not be allowed to logon.

GROUPS=(g1,g2) These optional (elective) groups are in addition to the user's private group and the system group when TIP searches the catalogue to resolve a reference to a program or file.

This parameter may be omitted, or only one group name may be given.

MAXUSERS= specifies the maximum number of concurrent users of this user-id. Specifying MXUSR=1 implies that the user-id can only be used on one terminal at a time.

If this keyword is not specified, there is no limit to the number of people logged on with this user-id.

MENU=mcsfmt This is the name of a screen format defined by MSGDEF to be used as a menu (or prompt). Whenever a program terminates without a reply to the terminal, this format will be displayed. The user may also have the menu displayed by pressing the MSG WAIT key while in system mode. If this parameter is not not specified, then a standard prompt message is used.

PASSWORD=pwd specifies the password to protect use of this user-id. (ie. the password must accompany the User-id at logon time).

PROG=trid trid is the transaction code which is to be called immediately after logon.

When this feature is used the user will automatically be logged off when the specified program ends.

This facility allows the user to be limited to a specific program (usually a menu program) thus excluding the user from all other facilities of the system.

SEARCH= controls the catalogue searching done for this user. If SRCH=NO is specified, then only the system group (TIPY) is searched for programs and files. If SRCH=YES is specified, then a complete order-of-search (see previous discussion) will be performed.

SECURITY=nn is the security level for this user. This controls access to programs and files. The security code may be specified as a number (1-255) or as one of the following reserved words (TECH, MAST, SYST, PROG, APPL).

The security level may not be specified as a numerically lower value than the security level of the user issuing the command.

-+*+-

3.7.5 CATALOGUING A TRANSACTION CAT: prog

The catalogue manager command PROG creates or modifies a program entry in the TIP/30 catalogue. All programs must be catalogued. The format of this command is as follows:

Syntax:

Prog	trid	. required positional parameter
	CDAsize=	. size of CDA required by program
	CMdLine=	. command line parameters required
	DeBug=	. type of debugging: YES / NO / IDA
	EDIT=	. removal of communications characters
	ENTer=	. allow execution from standard prompt
	FiLes=	. files to be auto accessed
	FRom=	. use keywords from another PROG entry
	GRouP=	. group to which this prog entry applies
	INsize=	. size of IMA required (IMS emulation)
	IMS=	. program run via IMS emulation
	LoaDM=	. load module name
	MAXsize=	. max activation record (IMS emulation)
	MCSsize=	. size of MCS area required by program
	OUTsize=	. size of OMA required (IMS emulation)
	SECurity=	. security level of program
	TiMeLocK=	. time range program canNOT be run
	TRANslate=	. translate input message to upper case
	USeage=	. program characteristics
	VOLatile=	. size of VOLATILE data area required
	WoRKsize=	. size of WORK area required by program

Where:

trid is a required positional parameter specifying the transaction-id used to schedule this program.

Programs can be called only using their catalogued transaction-id (trid).

CDASIZE=n n is the CDA size required by this program.

DEBUG=YES if the program is to be used in a debug mode. The use of this option causes the program to be executed with hardware storage protection in effect.

To use this facility the OS/3 supervisor must be be generated with RESMOD=SM$ASCKE.

DEBUG=IDA if this program is to be loaded with the Interactive Debug Aid (IDA) in control.

DEBUG=NO No debugging.

EDIT=YES for input messages received by TIPTERM or IMS/90 emulation all communications characters, sequences, dice and multiple spaces will be removed. Default is EDIT=NO.

EDIT=c specifies a character to be used as the field separator in input messages received by TIPTERM or IMS/90 emulation.

ENTER=NO this program may not be called directly by entering the TRID via the TIP command line.

That is, it may only be called by another program. Default=YES.

FILES=(,,) Up to 12 files to be opened when the program is loaded.

FROM=grp/trid Indicates that the keywords (except GRP=) for the program entry for "grp/trid" are to be copied to this program.

GROUP=name group name to which this program belongs.

IMS= YES if this is an IMS/90 program which is to be emulated.

INSIZE= The IMA size if this is an action program to be run under IMS/90 Emulation.

LOADM= names the load module associated with the transaction-id.

Default is the same as the trid positional parameter.

MAXSIZE= For programs running under IMS emulation and USAGE=RELOAD, and utilizing immediate internal succession, this keyword specifies the size of the largest program in the succession chain.

Size (in bytes) of the load module plus all required work areas (CDA, WORK, IMA, OMA etc).

MCSSIZE= The size of the MCS area (parameter passed to native mode programs.

OUTSIZE= The OMA size if this is an action program run under IMS/90 Emulation.

SECURITY= is the security level assigned to this transaction. Specified as a number between 1 and 255.

TIMELOCK=(bgn,end) The hours of the day that this program is NOT available. Specified using a 24-hour clock.

Eg: =(830,1730) or (1800,900).

TRANSLATE=YES For input messages received by TIPTERM or IMS/90 emulation all alphabetic characters will be forced to upper case.

Default=NO.

USAGE=REENT This program is to be used re-entrantly. This is also specified for COBOL programs which were compiled with the shared code option [OUT=(M) for COBOL68; IMSCOD=YES for COBOL74].

If a COBOL program you must also specify the VOLATILE data area size (see VOLATILE=).

Re-entrant programs have 'sticking' power and reduce disk I/O's needed to schedule the program.

USAGE=REUSE This program is serially re-useable. Only one process is allowed to use this program at a time.

The process must terminate before the load module may be used by another process.

Re-useable programs have sticking power.

For COBOL programs the VOLATILE data area size (VOLATILE=) must also be specified.

USAGE=RELOAD This program is to be reloaded each time the program is used.

This entry is required if the program is neither re-entrant nor re-useable.

VOLATILE= The size of the volatile data area of a shareable COBOL program.

If this program calls any data base management routines, add four times the number of parameters in the longest "CALL" parameter list to the size reported by the COBOL compiler.

WORKSIZE= The size of the work area required by this program.

-+*+-

3.7.6 CATALOGUING A FILE CAT: file

The FILE catalogue statement is used to create or modify a file entry in the TIP/30 catalogue.

All on-line files must be catalogued and it is recommended that the user take advantage of the logical naming of files and the assignment of security codes as these capabilities enhance the flexibility and security of the system. The format of the FILE statement is as follows:

Syntax:

```
File  lfn          . logical file name
      GRouP=       . this file statement for this group
      LFD=         . LFD name of the file as given in JCL
      ReaD=        . "NO" : read not allowed
      SECurity=    . security level assigned to this file
      WRite=       . "NO" : write not allowed
```

Where:

lfn A required positional parameter identifying the logical file name by which user on-line programs must access the file.

GROUP= Group to which this file entry applies. Default "TIPY".

LFD= LFD name of the file as given in the TIP/30 JCL and in the TIP/30 generation.

READ= "NO" indicates file may not be read. (Output only file for this group). Default is TIP generation specification.

SECURITY=n is the security level assigned to this file. See description of Security in previous section.

WRITE= "NO" indicates file may not be written. (Read only file for this group). Default is TIP generation specification.

-+*+-

3.7.7 CATALOGUE HINTS FOR TESTING PROGRAMS CAT

When testing a new program it is recommended that the transaction code be catalogued with slightly larger work areas (CDA, MCS, WORK, IMA, OMA, etc...) than actually required. When the program is completely tested update the catalogue to reflect the correct sizes.

During program development the areas tend to grow and the programmer usually forgets to keep the catalogue up to date. Having an area catalogued too small may result in some portion of another program or TIP/30 being destroyed.

Catalogue the program being tested as USage=RELOAD and possibly DEBUG=YES. When completed declare it re-entrant (if it is), DEBUG=NO, and specify the final volatile data area size. Only make re-entrant programs resident.

-+*+-

3.7.8 UPDATING CATALOGUE RECORDS CAT

When updating existing catalogue records you need only specify sufficient information to identify the key of the catalogue record desired and the keywords for the information which is to be updated.

Always give the type of record USER, FILE, or PROG; the item's name, and the GROUP name.

Example:

```
USER TOMMY PROG=TMENU.
PROG UPDT GRP=AP CDA=768.
FILE MAST GRP=AP SECUR=88.
```

-+*+-

3.7.9 CATALOGUE STATEMENT CONTINUATION CAT

CAT will only process the first 72 characters of an input line.

If you are entering data on the terminal, type as much as you can (up to 72 characters) and then press transmit. CAT will prompt you again.

Leave at least one space after the SOE character in the prompt and continue to enter the additional keyword parameters.

When you are entering the last line of a multi-line command, terminate the last line with a period.

CAT will automatically terminate the previous command if it reads a line which begins with a command.

-+*+-

3.7.10 LISTING CATALOGUE ENTRIES CAT: list

For the List and Delete commands, the command line format is as follows:

Syntax:

```
List      Group/Id/Elt [,Type]  [ GRouP=xxx]  [ LoaDM=xxx]
LS        Group/Id/Elt [,Type]  [ GRouP=xxx]  [ LoaDM=xxx]
DELete    Group/Id/Elt [,Type]  [ GRouP=xxx]
Write     Group/Id/Elt [,Type]  [ GRouP=xxx]
```

Where:

Group User-id or group name of the catalogue records to be processed. Only Master type users may process catalogue records that have a different group name than their own user-id or one of the groups to which they have access. If this parameter is not given, then the user's private group is used.

Id name of the item (program or file) to be displayed. If not given, then all items in the specified group are displayed.

Elt The element name of a dynamic file. This parameter is only valid when used with dynamic file entries. If not given, all elements are processed.

Type The types of catalogue records to be processed. The value for this parameter may be as follows:

- '*' = process all entries
- 'P' = process program entries
- 'U' = process User-id entries
- 'F' = process file (all files) entries
  - 'D' = process dynamic file entries
  - 'E' = process Edit file entries
  - 'S' = process System file entries

If type is omitted, all types are processed (exception is the delete command which insists on a supplied type!).

Note: for the first three parameters (Group, Id, and Elt), the value entered may be preceded with the asterisk (*) character to denote a prefix search. If the value entered is preceded with an exclamation mark (!), then this is taken to be a prefix search for items that do not begin with that prefix.

GROUP= This optional specification limits the scope of the command to entries matching the specified group name.

LOADM= This optional specification limits the scope of the command to entries (obviously PROGRAM entries) that refer to the specified load module.

Example:

```
List *,*pay
```

- this would indicate a list of all catalogue records (no type given) of any group (* alone is a 'match all') and of any name that starts with the letters 'PAY' (*PAY).

Additional Considerations:

The 'Write' command will list to the terminal as well as write the information to a library file called RUN/LISTCAT. This file may be edited by the text editor and later fed back to 'CAT' via '.IN' files.

List will produce an unsorted listing.

LS will produce a sorted listing.

-+*+-

3.8 COBOL REFORMATTER (CONVERSION AID) CC

This utility will reformat a COBOL source program. The input source element may be in COBOL-68 or COBOL-74 format.

The source is reformatted so that (where possible) the PICTURE clauses are column aligned, the full spelling of COBOL reserved words is used, IF clauses are indented to show the scope of nesting etc.

A keyword parameter is available to allow the specification that a COBOL-68 to COBOL-74 conversion is to be performed. This is a syntactic conversion; that is, the necessary spelling changes and cosmetic changes will be made. The user is still responsible for changes to the input/output statements that are required.

The CC program assumes that the input module is a syntactically correct program. If this is not the case, unpredictable results may occur.

Syntax:

IN=file/elt,OUT=file/elt [,LIST=N,COPY=N,RENUM=Y,MODE=COBOL74]

Where:

IN= Required keyword parameter giving the input file and element name.

OUT= Required keyword parameter giving the output file and element name. May not be the same as IN=.

LIST= YES/NO option whether to list the output at the terminal. Default value is "NO".

COPY= YES/NO option whether the input module is a DATA DIVISION copy book. Default is "NO".

RENUM= YES/NO option whether to automatically renumber the level numbers of item in the Data Division in the output element. Increments of 5 are used. Default is "YES".

MODE= Specifies the type of COBOL for the input module. The default is COBOL68 to COBOL74 conversion. (The conversion is only suitable for online programs)

MODE=COBOL68 input is COBOL68 program.

MODE=COBOL74 input is COBOL74 program.

MODE=DMS90 input is COBOL74 program which uses DMS/90. The DMS/90 verbs are processed.

Example:

```
IN=TIP/TTSAMP,OUT=RUN/TEST,MODE=COBOL74
```

Reformat element "TTSAMP" from file "TIP" and put the new version in element named "TEST" in file "RUN" (YRUN). The input module is in COBOL74 format.

Error Conditions:

The CC program may indicate FCS errors if an error occurs reading or writing a file/element.

Additional Considerations:

The CC program keywords must be terminated with a period or no action will be taken.

The CC program will display "Working - Please Wait" if it is successfully processing your command.

3.8.1 COMMUNICATIONS CONTROL AREA DISPLAY CCA

CCA is a utility program which displays information and statistics derived from tables within ICAM.

To use CCA the ICAM must be generated with "STAT=YES" on BUFFER statements and "STATS=YES" on LINE statements. This will cause ICAM to collect statistics for lines and buffers. CCA will display statistics required by a system programmer who is tuning the performance of a network.

This program should be run when TIP/30 is operating in a "production" environment for the statistics to be of any real meaning.

The user should re-run CCA when ever any additional communications hardware is added or after any major system changes affecting ICAM through-put.

Syntax:

CCA

The commands are as follows:

Arps	produces a list of A.R.P. penetration statistics.
Buffers	produces a list of buffer penetration statistics.
Lines	produces a list of lines showing line name, type, speed, number of terminals, errors, etc.
Terminal	produces a list of terminals on each line, showing terminal name, size, polling interval, status, polls, messages in, messages out, errors, etc.
End	end execution of CCA.
Quit	end execution of CCA, and logoff TIP/30.

-+*+-

3.9 SET U400 CONTROL PAGE CPAGE

The CPAGE program is used to set the control page of a UTS-400 type terminal. This also includes UTS-20 and UTS-40.

Syntax:

CPAGE[,opt]

Where:

opt command line option indicating the desired setting of the XMIT option of the control page.

"A" sets the control page to transmit all ("ALL")

"V" sets the control page to transmit variable (unprotected) ("VAR")

"C" sets the control page to transmit changed ("CHAN")

Example:

CPAGE,V

Error Conditions:
None.

Additional Considerations:

The preferred option for TIP/30 operation is 'V' but users doing IMS/90 emulation may have to use 'A' with its additional transmission overhead.

3.10 ABNORMAL TIP/30 SHUTDOWN CRASH

This command will cause TIP/30 to shut down immediately. It will not wait for all users to log off. A JOBDUMP will be taken if the JOBDUMP option was specified in the TIP/30 job control stream.

Syntax:

CRASH

Where:
No parameters required.

Example:

CRASH

Error Conditions:
None.

Additional Considerations:

The system SHUTDOWN program will NOT be scheduled.

3.11 CREATE A DYNAMIC FILE CREATE

The CREATE program is used to make a file entry in the TIP/30 catalogue. By doing this, the user is creating a new FCS dynamic file within TIP$RNDM.

Syntax:

CREATE[,type] aft-name,file-name

Where:

type the type of dynmic file to create

'P' : the file created is a permanent dynamic file and will remain in the system after the user logs off unless it is specifically scratched.

'T' : the file created is temporary and will be scratched by TIP/30 when the user logs off. In the case of an HPR or power failure, this file will be scratched during the subsequent TIP/30 initialization. NOTE: This is the default type.

aft-name is the logical file name to be assigned to the file. After the file has been created, it is automatically assigned to the user. This is the entry in the active file table (AFT).

file-name is the entry to be made in the catalogue for the new file. The catalogue-name consists of three sections, USER-ID/CATL-ID/FILE-ID which uniquely identify each file in the catalogue. The user must at least specify FILE-ID to access the file. If the USER-ID is not specified, then the USER-ID used to logon TIP/30 is used. If CATL-ID is not specified then CATL-ID is set to FILE-ID.

In the following examples assume that the USER-ID 'ARC' was used to LOGON, then:

Example:

```
CREATE    STRTUP,BGNFL
```

Will create the file 'ARC/BGNFL/BGNFL' as a temporary, dynamic file and assign it the logical name of STRTUP.

Error Conditions:
Errors may be reported from TIPFCS.

3.12 ON-LINE DISK DISPLAY AND UPDATE DD, DDU

Online disk display and update is a utility used to display and modify the contents of disk files at a terminal. It is designed to be a programming aid useful for testing and debugging.

For example, a file could be displayed to determine if a program being tested had altered it correctly or had erroneously left the file intact. Using the update feature would allow a quick modification to prepare for another test.

This utility can handle FCS dynamic files, indexed files, (including MIRAM), direct access files, and edit buffers. It displays one record at a time from the selected file. The records can be displayed in three formats: character, hexadecimal, or both character and hexadecimal. For records too large to fit on one screen, an option is provided to allow the user to 'page' through the rest of the record contents. The user selects records to be displayed by either record number or key depending on the file type.

STARTING THE DD UTILITY

Two transaction names are provided to call this utility program. DD is display only, but DDU allows both display and update functions.

It is suggested that DDU be catalogued at a higher security level than DD to reflect the relative power of the two transactions. To start up the utility enter:

```
>transaction-id filename
```

The following are all valid invocations to allow display only of the file MYFILE in the group ARC:

```
DD ARC//MYFILE

DD ARC/MYFILE

DD MYFILE
```

3.12.1 INTERACTION WITH DD & DDU DD, DDU

When the utility is invoked it displays the first record of the file specified. For indexed files this is the record with the lowest primary index. The initial mode of display is character with all non-displayable bytes shown as underscores.

At this point the user must 'tell' the utility what to do next. The possible actions are:

(1) specify another record to display

(2) specify another display mode

(3) go to another part of the current record (Paging)

(4) redisplay the current screen

(5) exit the utility (end)

(6) update the record displayed (DDU only)

A description of these actions immediately follows.

-+*+-

3.12.2 SPECIFYING A RECORD TO BE DISPLAYED DD, DDU

The method of specifying a record to be displayed is dependent on the file type.

For the purpose of specifying records, files are categorized as either indexed and non-indexed.

Records of indexed files are referenced through a key.

Records of non-indexed files are reference by a relative (to one) record number.

-+*+-

3.12.3 SPECIFYING A RECORD OF AN INDEXED FILE DD, DDU

The user can specify the next record to be displayed by entering the key in one of two fields provided at top of the screen. The fields are appropriately titled 'Hex' and 'Char' to indicate the type of key expected. Whatever value is entered as the key is right filled with low values. DD will display the next record with a primary index greater than or equal to the one specified.

If the key is given as character then no case conversion takes place. However, if a hex key is given all characters are converted to upper case since lower case 'a' through 'f' are meaningless. Invalid hex values are flagged and an error message is returned.

Pressing function key 2 will cause DD or DDU to display the next record (in sequence).

The actual key of the record displayed is shown in the same mode the user entered the key.

-+*+-

3.12.4 SPECIFYING A RECORD OF A NON-INDEXED FILE DD, DDU

The user can specify the record number in one of two fields filled with underscores at the top of the screen. These fields are appropriately titled 'Dec' and 'Hex' to indicate the type of number they expect.

Pressing function key 2 (F2) will imply that the user wishes to select the "next" record (sequentially).

When the record is displayed the current record number is displayed in both decimal and hex.

Specifying invalid record numbers results in a variety of actions:

- Entering a record number more than 4096 past the highest record number will cause the utility to abort.
- For FCS dynamic files, specifying a record number past the current allocation will result in more disk space being allocated! The user is advised not to do this.
- Negative record numbers will result in a 'not found' message.
- If invalid hex values are encountered in input they will be flagged and an error message will be sent.

-+*+-

3.12.5 SPECIFYING DISPLAY MODES DD, DDU

The user controls the mode of display by using the field titled 'display=' at the top of the screen. Valid values are:

C character
H hexadecimal
B both character and hexadecimal

Nondisplayable bytes of character fields are shown as underscores. All display modes show the zero relative position in the record of the first byte in the line. These byte numbers are given in decimal for the character display and in hexadecimal in the other two display formats. Switching modes sets the display to the beginning of the record being displayed.

-+*+-

3.12.6 PAGING THROUGH THE CURRENT RECORD DD, DDU

Whether or not a record will fit on a screen depends on the display mode being used and the record length of the file. If a record will not fit on the screen the user can move back and forth through the record a screenful at a time. Pressing function key 3 pages forward through the current record unless the end of the record is currently displayed. Pressing function key 4 pages backward through the current record unless the beginning of the record is currently displayed.

-+*+-

3.12.7 TERMINATING DD & DDU DD, DDU

When the user has finished working with DD or DDU he terminates the utility by pressing msg-wait.

-+*+-

3.12.8 UPDATING THE RECORD CURRENTLY DISPLAYED DD, DDU

This function can only be used by starting the utility with the transaction-id DDU. It is intended to provide a method for quick and simple changes to aid in testing and debugging.

The update procedure requires that the user display the record to be updated. If the record is larger than the screen area the user may have to page to the segment being updated. To update the record displayed:

(1) place a 'Y' in the update field,
(2) alter the record as desired,
(3) leave the cursor in the space provided in the bottom right corner of the screen,
(4) press transmit and wait for a reply.

After completing the above steps the user will receive an informational message about the processing that occurred. If the update was successfully executed the display is redisplayed with the updated record contents and the following message appears:

"RECORD UPDATED SUCCESSFULLY".

Otherwise, an error occurred and an informational message will be displayed. The user can now repeat some or all of the above steps and try to update the record again.

-+*+-

3.12.9 UPDATING A CHARACTER DISPLAY DD, DDU

When a record is displayed in character mode, nondisplayable characters appear as underscores. In updating a character display DDU ignores all bytes received as underscores. This prevents the non-displayable data from being converted to underscores. Even underscores entered by the user will be ignored! To insert underscores in a record the user must update from one of the other display modes.

The character display is case sensitive and will display both upper and lower case. This allows the user to make changes using mixed case. At the same time remember that no case conversion occurs on input.

-+*+-

3.12.10 UPDATING A HEX DISPLAY DD, DDU

When the user transmits the screen with the new contents of a record the data is converted to upper case. As a result lower case 'a' through 'f' will be valid input. Invalid hex digits are flagged and will prevent the update from taking place.

-+*+-

3.12.11 UPDATING A MIXED DISPLAY DD, DDU

We refer to the mixed display as showing character and hex simultaneously. The part of the screen displayed in character is protected. The reason for this is that only the hex fields are used in updating the record from this display.

The hex fields are converted to upper case so that lower case 'a' through 'f' will be valid input. As with the hex display, invalid hex digits encountered in input are flagged and result in an error message.

-+*+-

3.12.12 RECORD PROTECTION DD, DDU

When a user attempts to update the record being displayed there is a possibility that the record has already changed since it was displayed. For system files, DDU checks to see if the record has changed. If it has, no update occurs and an error message is sent to inform the user. It is necessary to do this check since DDU does not use file locking. Record locking is used with system files for the duration of the update.

For FCS dynamic files and edit buffers no record locking is used and no attempt is made to determine if the record has been changed. Since QED locks an edit buffer changes cannot occur simultaneously from QED and DDU. However, two users both using DDU could make changes to the same record of an edit buffer without being aware of changes made by the other user.

The user should remember that DDU is not a general purpose editor and is designed for quick changes to disk files.

-+*+-

3.12.13 FUNCTION KEY USAGE DD, DDU

MSG-WAIT

- Terminates DD and DDU

F1 or F5

- Redisplays the current screen as it was last sent to the terminal.

F2 or F6

- Displays the first page of the next record in the file. For non-indexed files, this means the relative record number is incremented by one. For indexed files, this means the next record found sequentially.

F3 or F7

- Displays the next page (or screenful) of the current record. If the end of the record is currently displayed, no paging occurs and a redisplay takes place.

F4 or F8

- Displays the previous page (or screenful) of the current record. If the beginning of the record is currently displayed, no paging occurs and a redisplay takes place.

-+*+-

3.12.14 POTENTIAL PROBLEMS DD, DDU

The following points are guidelines to help the user enjoy trouble free use of DD.

- In the hands of an animal DD can have serious side effects when used with FCS dynamic files. If the user asks to see a record past the current end of file then the file will be automatically extended to provide that record. For example asking for record 500 when only 40 exist would result in 520 records being allocated to the file. This is like a malignant tumour feeding itself on your TIP$RNDM file. To cure this disease use the TIP SCRATCH program to get rid of any mammoth files created.

- Specifying record numbers more than 4096 past the end of the file (for dynamic files only) causes DD to abort. Avoid this problem by making sensible requests.

- Dynamic files and edit buffers are updated without locking records. Avoid problems with updates by cataloguing DDU with at a higher security level than DD.

- When updating a file from a character display remember underscores are ignored. To replace bytes with underscores you must use the hex or mixed display.

- Updates can cause problems if the user forgets that both upper and lower case are accepted in character mode. If you want upper case then you would be wise to enter the data in upper case.

- Records in FCS dynamic files (includes edit buffers) and non-indexed files cannot be physically deleted.

- Since DD and DDU are sensitive to the case of character data, these transactions should be used with caution from terminals that do not support upper and lower case case.

-+*+-

3.13 SET FILE IN TEST MODE DEBUG

The DEBUG program places a named file in a READ ONLY mode for testing programs. A command line option indicates whether the file is to be placed in debug mode or removed from debug mode. In debug mode, any WRITE attempts from any program (at this terminal) will be ignored, thus ensuring the integrity of the file.

Syntax:

DEBUG,[opt] aft-name

Where:

opt furthur defines what operation is to be done.

'N' places a file in debug mode.

'F' removes a file from debug mode. (Default).

aft-name is the name of the assigned file that is to be placed in (or removed from) debug mode.

Example:

DEBUG,N CUSTOMER

This command would place the file assigned to the logical name of 'CUSTOMER' in debug mode and would ignore any subsequent write requests to the file from this terminal.

Error Conditions:
File not assigned.

Additional Considerations:

This option is only effective while the file is assigned to the user (ie. once the file is de-accessed the DEBUG option is no longer effective).

3.14 DEFINE FUNCTION KEYS DEFKEY

The DEFKEY program is a utility program that allows the user to specify a character sequence that will be "painted" on the screen whenever a function key is pressed as a response to the standard system prompt. After the character sequence is painted on the screen, TIP/30 will generate an auto transmit sequence to the terminal. The net effect of this is to simulate the keying of that character sequence.

The definition of function key contents may be specified by user group. The search for the appropriate function key contents follows the same sequence as the standard order of search in the catalogue: the user's private group is searched first, then elective groups one and two, and finally, the universal group "TIPY".

By utilizing the DEFKEY program, the user may assign character strings to function keys and make it simple to enter the character strings.

The DEFKEY program stores the function key definitions in a TIP/30 dynamic file with the name: <group>/FUNCTION/KEYS (where the group is name of the group that owns the definitions).

The DEFKEY program is NOT an interactive utility. It accepts information only from the command line.

DEFINE FUNCTION KEYS

Syntax:

DEFKEY [GROUP=xxxxxxxx] [LIST] [nn,'...']

Where:

GROUP= This specifies the name of the group desired. Default is "TIPY".

LIST A positional word indicating that DEFKEY is to list the current definitions of the function keys for the specified group.

nn Numeric specification of function key. Valid values 1 through 23. (Function key 23 may be returned by the Master terminal in a UTS-400 cluster when a power-on confidence test is initiated - this is a hardware strapping option of the UTS-400).

'...' The desired contents of the function key enclosed in quotes (single or double).

Example:

DEFKEY LIST

- Will list the current contents of the function keys for group "TIPY" (the default).

DEFKEY 12,'WHOSON',9,'LOGOFF'

- Will define function key 12 as the character string "WHOSON" and function key 9 as the string "LOGOFF".

3.15 ABORT A PROGRAM DIE

The DIE program may by used to force an abnormal termination of a user program running at another terminal.

Syntax:

DIE,identifier

Where:

identifier specifies the user or terminal name of the program to be aborted.

Example:

DIE,JOHN

This command would cause the program being executed by user JOHN to be abnormally terminated.

Error Conditions:
User or terminal cannot be found.

Additional Considerations:

The program is not aborted immediately. It will be aborted the next time it is activated. One may have to press transmit or msg-wait on the terminal running the program to cause TIP to reschedule the program and thus cause the abort.

3.16 DOWN LINE LOAD UTILITY DLL

DLL is a supplied program designed to assist the user working with the UTS-400 terminal. This program provides the capability of down line loading the UTS-400 memory from the host. The UTS-400 may be loaded with user developed programs, compiled with the Allinson-Ross Corporation 8080 Cross Compiler (UTSASM and ASM80) and/or programs produced using Univac's software. In addition, the UTS-400 may also be loaded with screen formats that have been created with the TIP/30 Message Control System (MSGDEF). A supplied UTS-400 program (MCS400) must be loaded whenever the user is down line loading MCS screen formats. Refer to the section on the Message Control System for further information on the use of down line loaded screen formats.

It is important to note that the operation of this (DLL) program involves a staging buffer within the program. All program and message requests are collected in this staging buffer, then the entire buffer is loaded into the UTS-400 with a single command.

Note: this program requires 18k of memory, therefore the MAXPROG parameter in the TIPGEN procedure should be at least 18000.

The DLL commands are as follows:

Include file/element

- add the UTS-400 object module generated by either ASM80 or UTSASM (described later) to the staging buffer. The transfer address is set to the address specified in the transfer record of this module

- TIP/MCS400 is a module which interfaces with MCS in the host to display screen formats on the terminal.

Get module

- add the UTS-400 object module as generated by either MAC80, PL/M, or UTSCOB to the staging buffer. The transfer address is set to the address specified in the transfer record of this module.

Ntr address

- the transfer address is changed to the address specified in the first parameter.

Message mcs-name,[U]

- De-code and load the specified MCS message into the staging buffer. Only heading information is normally stored. When this message has been loaded into the terminal TIP/30's Message Control System will only send the data portions, thus resulting in a complete screen. If the second parameter is 'U' this message will be stored with filler's to represent the data fields. If you have a data entry screen for which you want the data fields to be filled with underscores, then specify "U" with this command and specify '_' as the filler character in the MCS packet when your program calls TIPMSGO. This will result in the shortest possible XMIT time to display the message on the terminal.

Function key#,word,XMIT,SOE

- this equates a UTS-400 function key F5 thru F13 with a word of up to 8 characters. When the function key is pressed the word will be written on the terminal where the cursor is positioned at that time. If the 3rd parameter is XMIT then a transmit function will take place. If the 4th parameter is SOE, a start-of-entry will be placed in front of the word.

Load [terminal] [,dvc,name]

- the contents of the staging buffer will be down line loaded into the memory of the UTS-400. After the UTS-400 has been loaded, the transfer record is sent to the UTS-400. If no terminal is specified, then the down line load is performed on the terminal that is in use. Note that only the master or primary terminal of a UTS-400 cluster receive down line loading.

- 'dvc' is the auxiliary device index where the program is to be stored (e.g diskette).

- 'name' is the name to be given to the program when it is stored on the diskette.

LT [terminal] [,dvc,name]

- same command as Load except the time of day from the host will be loaded to location A06B in the format 'HHMMSST', 7 digits of hours, minutes, seconds, and tenths.

input re-direction

- re-direct input to the given file/element. This command is very useful. The user may make up a canned run stream for this program which may be run at the begining of each day to load all of the UTS-400 clusters with the screen formats. The .IN file as described in the section on TCP may be used here. This may be on the command line to DLL, example

```
DLL <file/elt
DLL R file/elt
```

End

- end execution of DLL

Note:

- Only the first letter of the DLL commands are required to identify the command (IE. L-load, I-include etc.).

Example stream which may be stored and called via a .IN file:

```
Include TIP/MCS400                  -MCS terminal program
Message ACCT1,U                     -accounting screen
Message PAY1                        -payroll screen
Function 5,PAYUPDT,XMIT,SOE         -send in the word PAYUPDT
Loadm   TM01                        -load into UTS-400 cluster
Loadm   TM06                        -load into UTS-400 cluster
End                                 -end of loader
```

3.17 UTS-400 MESSAGE CONTROL SYSTEM MCS400

This is a UTS-400 program which is written in Intel 8080 assembler language and supplied to the user as an ASM80 object module. It operates under the direction of TIP/30's MCS to display screen formats on the terminal and eliminate the need to continually transmit it down the line. This program and messages to be used should be loaded into the UTS-400 master when TIP/30 is started up, and any time the UTS-400 master (or controller) is initialized [ie: via a power-on confidence (POC) test].

Function keys 5 thru 13 can be programmed by DLL. The other function keys perform as follows.

- F14 := beeps the terminal to let you know that the program has been successfully loaded.
- F15 := takes ZZname from home position, looks for 'name' in the screen table, then displays it.
- F16 := displays the next screen format in the table. The screen name is displayed in bottom right hand corner.
- F17 := does an erase display and cursor home.
- F18 := re-displays the last screen format used for this terminal

All of these functions operate independently per terminal in the cluster.

- F20 := allows you to set the time of day in the terminal. The program will keep the time of day as HHMMSST in location A06B. To set this time, enter the time at the home position of the terminal and press F20. Time is kept in hours, minutes, seconds, and tenths of seconds.
- F21 := begins the display of the time of day at the home position of the terminal.
- F22 := end the display of time of day.

3.18 DOCUMENT GENERATOR DOC

The Document Generator (DOC) is a program which creates a formatted document from data files created using the TIP/30 text editor. A batch version of this on-line program is also supplied by Allinson-Ross. The data in the input file consists of the text to be printed as well as imbedded commands which provide formatting instructions to the DOC program. The DOC program reads this "raw text", acts on the imbedded commands, and produces a formatted output document at the terminal, auxiliary printer or the site printer.

The DOC program consumes resources at a rate roughly proprtional to the length of the input raw text and the number and nature of imbedded commands. It may be prudent to use the batch version of DOC for serious (or high volume) documenting and reserve on-line use of DOC for testing small, self-contained documents.

DOC recognizes two types of imbedded commands:

- declarative
- imperative

A declarative command is one which changes the format of the document or the mode of program operation. A declarative command does not necessarily have an immediate effect.

An imperative command is one which performs an action either on the current line or on the next line. All commands are imperative except as noted in the command descriptions.

A command (declarative or imperative) may be in one of three formats:

Syntax:

```
@F

@FA

@Fnn
```

Where:

'@' is the command delimiter (which is by default the commercial at sign character as shown);

F represents a single character which indicates the function to be performed;

A represents a single character which supplies further information to be used in executing the command;

nn represents a one or two digit value which specifies quantitative information to be used in executing the command.

If the value required is a single digit, it may be specified as a single digit unless the first character of text which follows the command is also a digit. In that case, a single digit would be erroneously associated with the digit in the raw text that follows. To avoid this situation, the single digit should be specified with a leading zero, as illustrated by '@I05' in this example:

```
@I051. PROCESSING THE COMMANDS...
```

3.18.1 ONLINE DOCUMENT GENERATOR DOC: online

The DOC program gives the user the ability to have documentation produced at the terminal, the central site printer, or a terminal auxiliary printer.

Syntax:

DOC[,option] file [,element] [,dest]

Where:

option choice(s) from the following list:

"U" - force upper case output;

"H" - print identification header page (default if destination is not an auxiliary device);

"N" - do not print header page (default if destination is an auxiliary device);

file The catalogued name of the library containing the element which is to be input to DOC, or the name of an existing edit buffer which is to be the input to DOC.

element The element name associated with "file" or omitted if the first parameter is the name of an edit buffer.

dest The destination of the output of the DOC program. Default is the terminal. Other possibilities are: PRNTR, AUX1 etc. If this parameter is strictly numeric, it will be interpreted as the number of lines to output to the terminal (the default destination) before prompting the user (see additional considerations following).

Example:

```
DOC,U   source/memo12,PRNTR
```

Will cause DOC to process the contents of element "memo12" from the library with catalogued file name "source". The formatted document will be sent to the batch printer, with forced upper case alphabetics.

Error Conditions:

The requested output device may not be available or the specified file/element or edit buffer may not be found.

-+*+-

3.18.2 ADDITIONAL CONSIDERATIONS DOC

When the output of DOC is directed to the terminal, the output is continuously rolled out (from bottom to top) until the number of lines specified by parameter 3 is encountered. At that point DOC will stop rolling out lines and prompt the user. The user has an opportunity to indicate whether or not to continue the output, go to a specific page or alter the pause interval. The choices are:

"E" or "Q"

- end document production.

"P nnn"

- proceed to page nnn.

"L nnn"

- change pause interval to nnn lines.

none of the above

- continue output.

-+*+-

3.18.3 SUMMARY OF IMBEDDED COMMANDS DOC

The Document Generator recognizes these imbedded commands:

COMMAND	FUNCTION	NOTES
@.	PHYSICAL FORM FEED	
@(	START MARGIN FLAGGING	
@! or @]	SAVE PARAGRAPH NUMBER	0-1 MODIFIER
@)	STOP MARGIN FLAGGING	
@-	CHANGE COMMAND DELIMITER	
@%	SWITCH INPUT TO FILE/ELEMENT	
@_	START/STOP UNDERLINING	(batch only)
@?	RECALL SAVED PARAGRAPH NUMBER	1 MODIFIER
@@	GENERATE LITERAL AT-SIGN	
@nn	CALL MACRO nn (0 thru 99)	1-2 MODIFIERS
@A	SPACE TO ABSOLUTE COLUMN	0-2 MODIFIERS
@B	GENERATE DOCUMENT INDEX	(batch only)
@C	END OF LINE (CENTRE)	0-2 MODIFIERS
@E	EJECT TO NEW PAGE	0-2 MODIFIERS
@F	FLUSH LINE	1 MODIFIER REQUIRED
@G	SET PAGE LENGTH	1-2 MODIFIERS
@H	HORIZONTAL SPACE	1-2 MODIFIERS
@I	SET INDENTATION (LEFT)	0-2 MODIFIERS
@J	JUSTIFY MODE	
@K	INCREMENT AND CALL MACRO	
@L	END OF LINE (JUSTIFY LEFT)	0-2 MODIFIERS
@N	NOTATION (HANGING INDENT)	
@O	START ODD/EVEN PAGE	
@P	RETRIEVE PAGE NUMBER	
@Q	DEFINE MACRO CONTENTS	
@R	END OF LINE (JUSTIFY RIGHT)	0-2 MODIFIERS
@S	SET LINE SPACING	1-2 MODIFIERS
@T	UNJUSTFIED MODE	
@U	SAVE COMPOSITION STATUS	ONLY USED IN HEADING
@V	RESTORE COMPOSITION STATUS	ONLY USED IN HEADING
@W	SET LINE WIDTH	1-2 MODIFIERS
@X	INCREMENT PARAGRAPH NUMBER	1 MODIFIER ("0"-"9")
@Y	LOG LINE IN TABLE OF CONTENTS	(batch only)
@Z	PRODUCE TABLE OF CONTENTS	(batch only)

-+*+-

3.18.4 PHYSICAL FORM FEED DOC: @.

This command will result in a real page eject. It is usually found within macro 20 [which is executed whenever a page overflow condition occurs (via @E or normal overflow)].

-+*+-

3.18.5 START MARGIN FLAGGING DOC: @(

This command causes the printing of a character in the left margin, thereby flagging the current line and all subsequent lines until an occurrence of the "turn off margin flagging" command [@)]. The character that is printed in the margin is usually the vertical bar character ('|'). This character is not printable on some print devices. The batch version of the DOC program allows the user to redefine the flag character.

-+*+-

3.18.6 SAVE PARAGRAPH NUMBER DOC: @!n ; @]n

This command is used to save the current paragraph number. The modifier 'n' uniquely identifies the saved number, and may have a value from 1 to 9. This command may be used in connection with the recall paragraph number command (@?) to facilitate references to previous paragraphs without having to guess (or explicitly include) the actual paragraph number.

This command may be spelled using the exclamation character ("!") or with the right side square bracket character ("]").

-+*+-

3.18.7 STOP MARGIN FLAGGING DOC: @)

This command terminates the printing of the margin flag on subsequent lines. This command is the logical inverse of the command "@(".

-+*+-

3.18.8 CHANGE COMMAND DELIMITER DOC: @-c

This command is used to specify a new command delimiter (ie: other than the default commercial at sign). Once this command is encountered, all subsequent commands must start with the character specified as 'c'. This command may be used to change the delimiter prior to calling a new input element (@%) which may have the standard delimiter ('@') as part of its text.

EXAMPLE:

When including the source of a program as part of a document and the program source has the character '@' in it, the following commands could be used:

```
@28@-↑↑%file/progname↑-@@29
```

-+*+-

3.18.9 SWITCH INPUT TO FILE/ELEMENT DOC: @%file/elt

This command specifies a source element that is to be used as input (read in) at this point. When this new element is exhausted, the data immediately following this command will be processed. The new input element may also have input switching commands (@%file/element) in it, however such nesting may only occur to a maximum of five levels. If the file is not specified, the last input file is used (ie @%eltname).

For the batch version of the DOC program, the filename must match an LFD name of a standard OS/3 library file.

For the on-line version of DOC, the filename is the catalogued name of the library file containing the new input element. In the online version, the filename may also be the name of an edit buffer that contains the input data. In this case, the eltname field remains blank (ie @%buffername).

Syntax:

@%file [/element] [,type]

Where:

file The catalogued file name of the library to use or, the edit buffer name (assuming element and type are omitted).

element The element name to read from the specified library.

type Standard element types: "S", "M", "P" [Default is "S"].

Example:

@%JCS/TIP30

Switch to reading element named "TIP30" from library catalogued with name "JCS".

Additional Considerations:

This DOC command should be followed by at least one blank (so that any trailing text is not erroneously associated with the parameters of this command.

-+*+-

3.18.10 START/STOP UNDERLINING DOC: @_

This command is implemented as a toggle. It will either start or stop underlining. The initial state is underlining off. The first ocurrence of @_ will begin underlining; the next will stop underlining etc.

Blanks in the string will not be underlined (see following example).

For example, the following line was underlined by the following string in the input document:

```
@_This entire sentence is underlined.@_
```

This entire sentence is underlined.

-+*+-

3.18.11 RECALL PARAGRAPH NUMBER DOC: @?n

This command is used to recall a previously saved paragraph number. The paragraph number must have been saved with the save paragraph command (@!n - see 3.18.6). The modifier 'n' uniquely identifies the saved number, and may have a value from 0 to 9. If a value of 0 is used, then the current paragraph number is recalled. (This would, for example, be employed in cases where the current paragraph number was needed in some heading or other information).

-+*+-

3.18.12 GENERATE LITERAL AT-SIGN DOC: @@

This command will cause a real commercial at sign to be generated. Since the at sign is the default command character, it is necessary to have this mechanism available for those situations when a real at sign is desired in the output.

-+*+-

3.18.13 CALLING MACROS DOC: @nn

A macro is called by a command expression of the form:

@nn

Where nn is an integer in the range 0 through 99 identifying which of the 100 macro definitions is to be called.

Any macro may be called by the user at any point after the definition of that macro.

-+*+-

3.18.14 SPACE TO ABSOLUTE COLUMN DOC: @Ann

This command causes DOC to generate in the current line a number of space characters. The number of spaces generated is calculated to be the difference between the current column location and the value specified as a modifier to this command. If the value specified is not greater than the current column location, the command is ignored.

-+*+-

3.18.15 GENERATE DOCUMENT INDEX DOC: @B

This command is used to produce an index. The index is produced from the logged (@Y) lines arranged in alphabetical order by the first significant word.

This command is ignored by the on-line DOC program, but is processed by the batch version.

The index of this document is initiated as follows

```
@EI N D E X @Y Page@R @B
```

-+*+-

3.18.16 END OF LINE (QUAD CENTRE) DOC: @Cnn

This command causes the line currently being constructed to be terminated. The line terminated by this command is unjustified and is centered between the (possibly indented) left margin and the right margin. The optional modifier 'nn' specifies the number of lines to leave after the centered line.

The information string

```
Table of Contents @Cl
```

Produces the following line

Table of Contents

-+*+-

3.18.17 EJECT TO NEW PAGE — DOC: @Enn,mm

Eject to a new page. The optional modifier 'nn' specifies the number of consecutive lines which must not be split between two pages. If nn lines (or more) remain on the current page, a new page is not initiated. If less than nn lines remain on the current page, a new page is initiated.

Modifier mm specifies the number of lines to leave if a page eject is <u>not</u> performed by this command.

If nn is not specified, an unconditional eject to a new page is performed. In any case, if a text line has not yet been terminated when this command occurs, the line will be terminated as if the command @L (with no nn specification) had occurred.

It is important to note that this command does not cause a physical form feed to occur, it generates a space command to position the current page at the bottom line of the page (as defined in the @Gnn command) then macro 20 is executed.

Macro 20 is assumed to contain the correct information to produce a page footing, eject the page and produce a page heading for the next page. An example of how this is performed for this document may be found in the section describing initial definitions of macros.

-+*+-

3.18.18 FLUSH LINE **DOC: @Fc**

Flush with character fill. This command causes the text already in the line to be unjustified and positioned at the beginning of the line. The text following this command and preceding the next @Lnn command is positioned at the end of the line. The intervening space between these two portions of the line is filled with repetitions of the character c. If space fill is desired, then the character c must be a space.

The information string

```
Reader@F.600 CPM@L1
```

Produces the following line

```
Reader.....................................................600 CPM
```

-+*+-

3.18.19 SET PAGE LENGTH DOC: @Gnn

This is a declarative command which controls the number of lines which may be printed on each page. The "nn" is the number of lines which will be printed and/or spaced before performing a skip to the home position. This number must be less than the number of lines between two consecutive home positions as determined by the physical carriage control mechanism.

Default = @G55

-+*+-

3.18.20 HORIZONTAL SPACE **DOC: @Hnn**

Immediate horizontal space. This command inserts space in the 'nn' print positions following the text most recently placed in the current line or, in the absence of such text, inserts space at the beginning of the (possible indented) line. If less than nn print positions remain in the current line, the excess is ignored and the line is not justified.

-+*+-

3.18.21 SET INDENTATION (LEFT) DOC: @Inn

This command changes the left indentation by resetting the indentation at nn print positions from the left margin. This command will neither terminate nor change the indentation for a line in progress; the change in indentation will only affect subsequent lines.

Default = @I00

-+*+-

3.18.22 JUSTIFY MODE **DOC: @J**

This command is used to set the mode of the Document Processor back to the standard 'justify' mode. This command is the logical inverse of the 'card image' (@T) command.

-+*+-

3.18.23 INCREMENT AND CALL MACRO DOC: @Knn

This command increments by 1 and retrieves (as text) a 6-digit counter (leading zero suppressed).

Before using this command, the macro 'nn' should be set to an all decimal string of 6 digits (ie. @Qnn000000").

This command is useful when a running counter is desired (for example, a STEP number to be used in a heading).

-+*+-

3.18.24 END OF LINE (QUAD LEFT) **DOC: @Lnn**

Terminate current line and start new line; the terminated line is positioned so that its beginning is at the (possibly indented) left margin and its end is followed by spaces. The value of nn is the number of lines of spacing in addition to the standard line spacing as specified in the command @Snn. The value nn need not be specified. If nn is greater than the number of lines remaining on the current page, a new page is initiated and the excess is ignored. The line terminated by this command is unjustified. This command should normally be preceded by a space character.

The information string

```
@L1Table of Contents@L1
```

Produces the following three lines

Table of Contents

-+*+-

3.18.25 NOTATION (HANGING INDENT) DOC: @Nnn

This command is used to produce a notation format (often called 'hanging indentation'). The text preceding this command is placed at the beginning of the (possibly indented) line. The text following this command is placed starting at a point which is nn print positions from the current position in the line; subsequent lines of text also begin at the same point. The effect of this command is terminated by the command @Lnn.

It should be noted that if spaces occur in the text preceding this command and if the line is subsequently expanded to the right margin, these spaces may be expanded thereby producing an undesired result. Alternatively, a macro such as macro 10 (see later description) may be used.

Example:

point one: this is the description of point one. Note that this description illustrates hanging indentation (otherwise known as notation format) and may go on for a rather long time and keep the temporary hanging identation.

-+*+-

3.18.26 START ODD OR EVEN PAGE DOC: @O

This command will call macro 36 if the current page number is even, otherwise macro 37 will be called. These macros do not have any specific pre-defined contents. They may be used to cause a new section to start on an even (or odd) number page.

-+*+-

3.18.27 RETRIEVE CURRENT PAGE NUMBER DOC: @P

This command will retrieve (as text) the current page number. The text retrieved is the page number expressed in the minimum number of decimal digits without leading zeroes. The user would normally precede and follow this command with any desired spacing or other decorations (ie: 'Page @P.').

-+*+-

3.18.28 DEFINING MACRO CONTENTS DOC: @Qnn..."

There are 100 macros that may be defined by the DOC user. The contents (ie: definition) of a macro may be changed any time and as often as required.

To define the contents of a macro the user would include a command such as:

@Qnn...string-of-commands-and/or-text..."

Where:

'nn' A number in the range of 0 through 99 identifying the macro being defined.

the string of commands and/or text is limited in length to 70 characters. The string must be terminated by the double quote character.

The string which the macro represents may contain calls to other macros, but the user should be careful to avoid defining a macro which calls other macros in such a manner that an endless loop of calls is created. The DOC program allows nested macro calls to a depth of 5.

A macro may be called by using the command "@0" through "@99". There is no provision for passing parameters to a macro.

-+*+-

3.18.29 END OF LINE (QUAD RIGHT) DOC: @Rnn

This command terminates the line currently being constructed. The terminated line will be unjustififed and positioned with the end at the right margin and the space to the left of the first non-space character is filled with spaces. The optional modifier 'nn' specifies the number of lines to leave after the line that was right-justified.

The information string

```
@L1Table of Contents@R1
```

Produces the following line:

Table of Contents

-+*+-

3.18.30 SET LINE SPACING DOC: @Snn

This is a declarative command which controls the spacing that occurs between lines of the generated output document. Single spacing is the default. If nn is 1, 2, or 3, then normal spacing is single, double or triple. If nn is greater than 3, the value 1 is assumed. If nn is zero, no vertical advance (implying print with no space or overstriking) will occur until another @Snn occurs with nn greater than zero.

-+*+-

3.18.31 UNJUSTIFIED MODE DOC: @T

This command causes the Document Generator to enter what is called 'card image' mode. In this mode, strings of space characters are not reduced to a single space (as they would be in justify mode), they are treated as real data characters.

If this command is not followed by any data characters in the current input, and the line width has been set to 72, then the following records will be printed as they appear in the input stream (card image). This is very useful in the production of complicated diagrams or tables.

It should be noted that in this mode column 72 should always be left blank as the end-of-line sequence is only invoked whenever a space is detected in column 72.

This command is often considered 'as is' mode because the raw text shows the exact format of the generated output text.

-+*+-

3.18.32 SAVE COMPOSITION STATUS DOC: @U

This command saves the pertinent information concerning the composition of the current line. It should be the first command in the definition of the macro which is executed at page overflow time (ie. Macro 20). The information which is saved is line width, indent value, case shift, and margin flagging.

-+*+-

3.18.33 RESTORE COMPOSITION STATUS DOC: @V

This command restores the information saved by the @U command concerning the composition of a prior line. It must be the last command in the definition of the macro which specifies the composition of page headings (ie. Macro @30) and has no other purpose.

-+*+-

3.18.34 SET LINE WIDTH **DOC: @Wnn**

This is a declarative command which sets the width of the line. The value nn is the number of print positions to be contained in the unindented line. There is no provision within DOC to allow lines wider than 99 print positions.

Default = @W72

-+*+-

3.18.35 INCREMENT PARAGRAPH NUMBER DOC: @Xn

This command will increment the current paragraph number by one. DOC generates paragraph numbers that are of the form:

nnn.nnn.nnn. etc

There may be from 1 to 9 levels within the paragraph number. Each level may be from 1 to 3 digits. By issuing the @Xn command the user is requesting that level 'n' of the current paragraph number be incremented by one and the levels to the right of the incremented level be discarded. This command may be considered to mean: 'start a new paragraph at level n '. For example, issuing the command @X2 when the current paragraph number was '12.3.4' will change the current paragraph number to '12.4'.

Given the paragraph number in the lefthand column below as the current paragraph number, the command @X3 produces the corresponding paragraph number in the righthand column.

CURRENT NUMBER	@X3	NEW NUMBER
4.1.	-->	4.1.1.
6.13.5.	-->	6.13.6.
7.2.9.4.	-->	7.2.10.
9.6.14.3.7.	-->	19.6.15.
12.2.7.10.3.14.	-->	12.2.8.

-+*+-

3.18.36 LOG LINE IN TABLE OF CONTENTS DOC: @Y

This command is used to have the current line stored in the table of contents file. The line is stored in the file in the order logged. An internal pointer chain is maintained through the file in alphabetical order by the first significant text in the line. Any indents or horizontal skips at the begining of the line will not affect the resulting order of the logged records.

The logged lines may be recalled into the document (usually near the end of the document) by using the @Z (sequential order) command or the @B (alphabetical order) command.

This command is ignored by the on-line DOC program, but is processed by the batch version.

-+*+-

3.18.37 SEQUENTIAL TABLE OF CONTENTS DOC: @Z

This command is used to compose the table of contents containing those lines logged via the @Y command. This command will cause the table of contents to be inserted in the document at the point the @Z is coded.

This command is ignored by the on-line DOC program, but is processed by the batch version.

This command normally appears near the end of the document and should normally be preceded by commands to format the first page of the table of contents.

The table of contents may be located at any point in the document.

The table of contents of a document might be initiated as follows (for example):

```
T A B L E@H3O F@H3C O N T E N T S@C Page@Rl @Z
```

-+*+-

3.18.38 EXAMPLE OF MACRO USE AND DEFINITION DOC

Assume that the user wishes to define a macro that will make it simpler to achieve the following:

- leave two blanks lines
- display the company name (centered)
- leave two blank lines

One way to do this would be to select a macro number to be used (assume 61 for example) and then DEFINE that macro as follows:

```
@Q61@L2Allinson - Ross Corporation@C2"
```

Whenever the user wishes to leave two blank lines before and after displaying the company name (centered) all that would be needed would be the inclusion of @61 in the text at the appropriate point. For example:

Allinson - Ross Corporation

EXAMPLE OF MACRO USE AND DEFINITION

Another potential use is rather simple but powerful. Assume that a certain phrase is used very often in a document. Under normal circumstances, the user would have to key that phrase in every time it was needed. A much simpler approach would be to define a macro containing the desired text and then it is just a matter of calling the macro whenever the text is required.

For example:

DEFINITION:	@Q61the party of the second part,"
USE:	whereas @61 the appellant...
RESULT:	whereas the party of the second part, the appellant...

The user should note that assigning a number to a macro definition is a critical part of the process - there is no provision for defining a temporary macro; once a macro is defined, the previous definition of that macro is no longer obtainable.

-+*+-

3.18.39 PREDEFINED MACROS 0-39 **DOC: @0-@39**

Macros 0 through 39 (inclusive) have been given default definitions. These definitions are in effect when the DOC program begins processing the user raw text. It is advisable to avoid modifying the definitions of these macros (but under some circumstances it may be required).

The initial definition of the general purpose macros and their intended use are described below:

@0 = @Q30"@E@X

- End document

@1 = @E@I@X1@A6@I05

- Produce a first level numbered paragraph heading

@2 = @L2@E30@I@X2@A10@I05

- Produce a second level numbered paragraph heading

@3 = @L2@E20@I@X3@A12@I05

- Produce a third level numbered paragraph heading

@4 = @L2@E15@I@X4@A15@I05

- Produce a fourth level numbered paragraph heading

@5 = @L2@E10@I@X5@A18@I05

- Produce a fifth level numbered paragraph heading

@6 = @L2@E10@I@X6@A20@I05

- Produce a sixth level numbered paragraph heading

@7 = @L1@E2@I05

- Terminates the line in process. Generates one blank line space, and assures that the next two lines will not be separated by a page break

@8 = @L1@E2@I5*@N04

- Generates a "bullet" paragraph using the character * as the bullet; following is an example:

* This is an example of the @8 bullet paragraph the bullet is placed in print position 6 and subsequent lines start in print position 11. The effect of this macro is terminated by initiation of a new line. This bullet paragraph is logically superior to the second bullet paragraph, @9, explained below.

@9 = @L1@E2@I5 -@N04

- Generates a "bullet" paragraph using the character - as the bullet; following is an example:

- this is an example of the @9 bullet paragraph, the bullet is placed in print position 11 and subsequent lines start in print position 16. The effect of this macro is terminated by initiation of a new line. This bullet paragraph is logically subordinate to the first bullet paragraph, @8, explained above.

@10 = @S@L@S1@N

- Produces a notation format using the @n command (see section 3.18.25) but escapes the limitation of that command concerning spaces within the noted text. Following are two examples; the first showing the limitation in the @n command; the second showing the same text preceding the macro @10:

(1) no def notice that, although ten spaces were desired following "def", the spaces preceding "no" and "def" might have been expanded when the line was justified.

(1) no def notice that now, with the use of the @10 macro, those spaces preceding "no" and "def" remain unchanged.

Note that the last command (@n) in the macro definition above is incomplete; that is, no specification of "pp" is included. Accordingly, the first two characters following the call on this macro (@10) must be the digits which complete the @n command at the end of the macro definition; for example, @1019, the value of the two digits following the macro call, @10, should be one less than the number of print positions from the start of the line to the point at which the text following the macro call is to be placed.

Of the two examples shown above, the second was created by the following sequence:

```
... line was justified. @7(1) no def@1019notice that
now, with the use of...
```

@11 = @Y@L01@H03

- Terminate a heading line

@12 = @L1@I5@H03

- Terminates a paragraph

@13 thru @18 = --reserved for future use--

@19 = @E00

- Called after the table of contents is produced.

@20 = @U@L2@I@.@30

- This macro is called when a page overflow condition occurs. The intent of this macro is to produce a page footing, eject the form, and produce a page heading for the next page. The initial definition shown above however does not contain any page footing data. Macro @20 could be redefined as follows if the user wanted a page footing which consisted of the page number.

  @U- @P -@C@I@.@30

@21 thru @25 = --reserved for future use--

@26 = --spaces--

- Contains the break word during the production of the index or table of contents.

@27 = Allinson-Ross Corporation Document Generator

- This macro is called by macro @30 in the generation of page headings.

@28 = @L1@E10@I@T

- Is used to simplify entry into the "card image" mode described under the command @T (see command @T).

@29 = @L@J

- Is used to simplify termination of the "card image" mode described under the command @J (see command @J). An indentation in effect prior to entering the "card image"

mode will no longer be in effect after termination of the "card image" mode.

@30 = @27@F Page @P@L2@V

- This macro is used to define the page heading. It is called via macro @20 which is called at page overflow time. Since the page heading macro (@30) above calls macro @27, for the text of the heading, the user need only modify macro @27 to set up his personalized page headings.

@31 = @E5,1- @26 -@C01

- Generate sub-heading for index or table of contents.

- This macro is called automatically during the production of an index (@B) or table of contents (@Z) whenever a logical break occurs between two lines. In an index, a logical break occurs when there is a change in the first letter. In the table of contents, however, a break is a change in the first level of the paragraph number.

@32 = 000000

- This macro contains the current page number.

@33 = 000002.5

- This macro contains the version number as specified by the '// PARAM VER=' job control card. If the version number is not specified via job control, then the version number is taken from the first input module read.

@34 = 82/08/18

- contains the date the document was generated.

@35 = 9:07:02

- contains the time received from the Operating System at the start of composition.

@36 = --empty--

- This macro is called if the page number is even when the 'call odd or even page macro' (@O) command is executed. This macro has no initial definition.

@37 = --empty--

- This macro is called if the page number is odd when the 'call odd or even page macro' (@O) command is executed. This macro has no initial definition.

@38 = WEDNESDAY AUGUST 18 1982

- This macro contains the current date in literal format

@39 = 0

- This macro contains the revision number of the input element.

@40 through @99 are not predefined and may be defined by the user to suit the requirements of the document being generated.

-+*+-

3.18.40 LIBRARY ERRORS **DOC**

If a library error should occur (usually file or element not found) the following error message will be printed:

```
**********************************************************************

                        LIBRARY ERROR!!!

        FILE/ELEMENT = xxxxxxxx(n), FUNCTION = f, ERROR= e.

**********************************************************************
```

Where:

xxxxxxxx	is the name of the file or element (depending on the function being performed) that was being accessed at the time of the error.
n	is the file number, each time a new file is opened, it is assigned a number from 1 to 10 starting at 1 for the first file, 2 for the second etc.
f	is the function being performed as follows:

FUNCTION	DESCRIPTION
0	open element (xxxxxxxx = element name)
2	open file (xxxxxxxx = filename)

e is the error type as follows:

ERROR	DESCRIPTION
2	file not found (// LFD missing)
3	element not in file
3	I/O error

-+*+-

3.19 NORMAL TIP/30 SHUTDOWN EOJ

This command will post the END OF JOB REQUESTED flag in TIP/30. When all users have logged off, TIP/30 will terminate gracefully.

Syntax:

EOJ

Where:
No parameters required.

Example:

EOJ

Error Conditions:
None.

Additional Considerations:

The system SHUTDOWN program (if one is specified) will be scheduled.

3.20 PHYSICALLY CLOSE ON-LINE FILE FCLOSE

This program enables the user to physically cause a Data Management "CLOSE" to be issued for up to eight files. Once closed, the files will not be available to on-line programs until a subsequent "FOPEN" is issued. This facility is also available as an OS/3 operator unsolicited command to TIP/30 ("CLOSE"). This program does NOT operate interactively. It requires up to eight filenames on the command line; OS/3 Data Management will be presented with an CLOSE request for each file name given.

Syntax:

```
FCLOSE  file1 [,file2]  [,file3]  ...  [,file8]
```

Where:

file1...8 the LFD name of the file(s) to be closed.

Example:

```
FCLOSE CUSTMAST,INVMAST,ORDENTRY
```

Will close the three specified files.

Error Conditions:

The LFD name specified may not be a valid LFD name (ie: not in the TIP/30 job control stream).

Additional Considerations:

If the operation is held pending (eg: deferred until users have relinquished control of the file) the user will be not be notified of actual completion because the FCLOSE program will terminate before the actual Data Management function is performed.

The LFD names used must be catalogued in the TIP/30 catalogue in a group to which the user has access.

3.21 LOGOFF TIP/30 FIN

The FIN command is used to logoff TIP/30. If TCP was called via the escape function, control returns to the program that was active at that time, otherwise the user is logged off.

Syntax:

FIN

Where:
No parameters are required.

Example:

FIN

Error Conditions:
If the user has not logged on, TIP/30 will not allow a logoff.

Additional Considerations:

FIN is a reserved word recognized by the TIP/30 command processor.

The FIN command is recognized to maintain downward compatability with previous releases of the TIP/30 system. The LOGOFF program is the preferred method of logoff.

3.22 PHYSICALLY OPEN ON-LINE FILE FOPEN

This program enables the user to physically cause a Data Management "OPEN" to be issued for up to eight files. This facility is also available as an OS/3 operator unsolicited command to TIP/30. This program does NOT operate interactively. It expects up to eight filenames on the command line; OS/3 Data Management will be presented with an OPEN request for each file name given.

Syntax:

```
FOPEN  file1 [,file2]  [,file3]  ...  [,file8]
```

Where:

file1...8 the LFD name of the file(s) to be opened.

Example:

```
FOPEN CUSTMAST,INVMAST,ORDENTRY
```

Will open the three specified files.

Error Conditions:

The LFD name specified may not be a valid LFD name (ie: not in the TIP/30 job control stream).

Additional Considerations:

Note that this program references files by the real LFD name - NOT the catalogued logical file name.

The LFD names used must be catalogued in the TIP/30 catalogue in a group to which the user has access.

3.23 DEACCESS A FILE FREE

The FREE program is used to release a file from assignment to a user. The effect is to remove the file from the active file table for the terminal.

Syntax:

FREE[,type] [aft-name]

Where:

type type of FREE to be done.

'A' : all assigned files are to be free'd. Any temporary files are scratched by this option.

'F' : any records held for update for the aft-name are to be released.

'X' : all records held for update for the user in any file are to be released.

aft-name active file name.

Example:

```
FREE  UPDATE
```

release the file that was assigned with the logical name of UPDATE.

Error Conditions:
TIPFCS errors may be reported.

3.24 DISPLAY USER HELP INFORMATION HELP

The HELP program is a utility which will display help information for a specified program. The user may ask to see the help information for many of the supplied utility programs.

Help information may also be provided (by the installation administrator) for the installation's user programs.

The HELP program is NOT interactive. It requires only one parameter (see following) and expects this parameter on the command line.

Syntax:

HELP [name]

Where:

name The name of the program for which the user needs help information. If omitted, the HELP program will display information to help the user run the HELP program.

Example:

HELP VTOC

Will display the supplied help information for the program "VTOC".

Error Conditions:
The requested help information may not be available.

3.25 INTERACTIVE DEBUG AID IDA

IDA is a utility program which facilitates the debugging of on-line programs. When the user activates IDA on behalf of a program, IDA is given control by TIP/30 and then executes the user program using the hardware execute instruction. After each user program instruction has been 'executed' (by IDA), the results and effects of that execution are displayed on the terminal in a format similar to the assembly language representation of an instruction. The information presented includes:

- program and job region relative address
- condition code setting
- instruction mnemonic and operands
- effective addresses (operand 1 and operand 2)
- first four bytes of operand 1 and operand 2

If you wish to debug a program that you call directly, and you only want to debug it once, enter the question mark ('?') character preceding the transaction code on the command line. This is only useful when the program to be debugged is the first one called. Programs that are called from other programs or via IMS/90 succession must be debugged by altering the catalogue entry for the program.

In order to have a program loaded with IDA activated, catalogue the transaction code with DEBUG=IDA. Now, whether invoked directly or by succession, when the transaction is loaded IDA will also be loaded and given control.

When IDA receives control, it will display various information about the current environment (transaction id, load module name etc) and will then prompt the terminal user for a command.

To begin debugging simply press transmit. IDA will begin executing your program at the address specified as the entry address by the Linkage Editor. At any time the user may interrupt the display by pressing MSG/WAIT or a FUNCTION key and enter any of the following IDA commands.

IDA COMMANDS

3.25.1 IDA COMMANDS — IDA: commands

(blank) no command - continue tracing program

+,offset Display next storage

Display next 16 bytes from the address last used in a 'D' or 'A' command. IDA remembers the last address which was displayed or altered. The user may specify a hexadecimal offset <offset> to be added to the address. If an offset is not specified a default value of 16 is used.

-,offset Display previous storage

Similar to "+,offset" but treats any specified offset as a negative value.

A PRA,n Alter storage

Begining at the program relative address (given as PRA) store the value specified as 'n'. The value may be specified as a hexadecimal string or a character string within single quotes (').

AA addr,n Alter storage at absolute address

Similar to "A" command except that the <addr> is specified as an absolute address rather than a program relative address.

AR r,n Alter general purpose register

Alter the contents of the register. <r> is the decimal number of the register to be changed. Leading zeros are not significant.

B PRA,N,C Specify breakpoint address

PRA is the program relative address at which the user wishes to interrupt execution of the program to be able to call other IDA functions. A maximum of eight breakpoints is allowed. This command is usually used in conjunction with the DISPLAY OFF command when the user wishes to inhibit the IDA display until a specific address has been reached.

<N> is the display mode option to be executed before prompting; usually "C", "I", or "N".

<C> is the decimal count of the number of times the breakpoint address is to be encountered before prompting the user.

When a breakpoint has been reached its address is displayed as well as information describing the display status (on/off) and mode (Continuous or Instruction).

C Continuous Mode display

The display is scrolled up for each instruction displayed. To interrupt the display the user must press a function key or MSG/WAIT.

D PRA Display storage

Display 16 bytes in hex and graphic starting at the user program relative address given as PRA.

DA addr Display storage from absolute address

Display 16 bytes in hexadecimal and graphic starting at absolute address given as <addr>.

DB Display Breakpoint Table

Display the user defined breakpoint addresses and their associated options and counts.

DE PRA Display edited

Treat the specified program relative address as if it was an instruction and display that location as an instruction.

DF Display floating point registers

Display floating point registers 0, 2, 4, and 6. (Each floating point register is 64 bits wide but is displayed as left and right 32 bits).

DI r,offset Display indirect

Display 16 bytes in hex and graphic from the address computed as the sum of the contents of register <r> and hex offset <offset>.

IDA COMMANDS

DR r Display General Purpose Register(s)

Display the contents of the specified (in decimal) register. If <r> is omitted, all 16 registers are displayed.

E End tracing

Tracing of the program is discontinued. The user program continues executing in normal mode (ie: not executed via IDA). If a subsequent program check occurs, control reverts to IDA.

ED End tracing and IDA

Similar to the "E" command except that a subsequent program check will pass control to PMDA instead of IDA.

F Display OFF

Turn the IDA display off. Usually used to inhibit the display of instructions in anticipation of reaching a breakpoint.

G PRA GO TO address

Alter the PSW to execute the next instruction at the program relative address given as <PRA>.

I Instruction mode display

The user is prompted for an IDA command after every instruction; forces display mode ON.

L addr Specify linked address

The user may enter the linker assigned address of the traced routine. This maintains a zero relative display of user program addresses for easy reference to the program listing.

N Display ON

Turn the IDA display on. Usually used as a breakpoint option.

O,PRA Omit Breakpoint

The program relative address specified by <PRA> is omitted (deleted) from the breakpoint table.

Q Stop run

IDA is terminated and control returns to the previous program on the stack, usually the TIP COMMAND PROCESSOR.

R term Redirect IDA display

Direct IDA'S output to the specified terminal. The terminal must be idle, (ie. no user logged on). This command allows the user to view the debugging information displayed by IDA at another terminal. This would normally be done so that the output from IDA does not destroy the output of the program being traced.

The "R" command may leave the alternate terminal locked up if the program terminates. You must set some breakpoint to get control back, then Redirect back to the original terminal sometime before the program terminates.

S PRA,n Search for memory contents (program relative)

Search for the value <n> (1, 2, 3, or 4 bytes) from program relative address <PRA>. The value <n> may be specified as a hexadecimal string or as a character string enclosed in single quotes.

T mnemonic Translate mnemonic to opcode

Translate the <mnemonic> for an instruction to its internal hexadecimal representation. Used to determine opcodes when displaying memory contents.

TN hexop Translate opcode to mnemonic

Translate the specified hex opcode to its mnemonic equivalent.

-+*+-

IDA COMMAND EXAMPLES

3.25.2 IDA COMMAND EXAMPLES — IDA: exmaples

COMMAND	FUNCTION
(blank)	CONTINUE EXECUTING TRACED PROGRAM
L 128	DISPLAY ADDRESSES ZERO RELATIVE TO 128
D 23A	DISPLAY 16 BYTES AT 'PRA' 23A
+20	DISPLAY 16 BYTES AT 'PRA' 25A
-	DISPLAY 16 BYTES AT 'PRA' 24A
DR	DISPLAY GENERAL PURPOSE REGISTERS
DR 11	DISPLAY GENERAL PURPOSE REGISTER 11
DF	DISPLAY FLOATING POINT REGISTERS
DI 12	DISPLAY 16 BYTES AT ADDRESS IN R12
DI 12,40	DISPLAY 16 BYTES AT ADDRESS R12 + 40
B D6,N,10	STOP AT PRA D6 ON 10'TH ENCOUNTER AND EXECUTE OPTION 'N' IE. DISPLAY ON
O D6	OMIT BREAKPOINT ADDRESS D6
B 7A	STOP AT PRA 7A, AUTO-OMIT ENTRY
DB	DISPLAY ENTRIES IN BRKPT TABLE
F	DISABLE IDA DISPLAY
N	ENABLE IDA DISPLAY
I	SINGLE INSTRUCTION DISPLAY MODE
C	CONTINUOUS DISPLAY MODE
A 23A,D200F002E000	ALTER MEMORY AT 23A TO X'D200F002E000'
A 23A,'T301'	ALTER MEMORY AT 23A TO C'T301'
S 24A,47B0F00E	SEARCH FOR 47B0F00E FROM ADDRESS 24A
S	SEARCH FOR NEXT OF PREVIOUS ARGUMENT
S 45E	SEARCH FROM 45E FOR PREVIOUS ARGUMENT
S 0,C1C2	SEARCH FOR C1C2 FROM ZERO
R TRM4	REDIRECT IDA OUTPUT TO 'TRM4'
T CLC	TRANSLATE 'CLC' TO HEX OPCODE
TN 41	TRANSLATE 41 TO MNEMONIC OPCODE
G 128	GO TO PROGRAM RELATIVE ADDRESS 128
DA 2364A	DISPLAY ABSOLUTE LOCATION 2364A
DE 128	DISPLAY INSTRUCTION AT 128
ED	END TRACING ALLOW PMDA DUMP
E	END TRACING, ALLOW USER CPROGRAM TO EXECUTE (RECALL IDA IF SUBSEQUENT PROGRAM CHECK)
Q	CANCEL IDA SESSION; EXIT TRACED PROGRAM

-+*+-

3.26 DISPLAY OS/3 JOB QUEUE INFORMATION JBQ

The JBQ program is a utility program that will display information about the OS/3 job queue. It is functionally similar to the OS/3 operator command "DI JBQ". The JBQ program recognizes the following commands:

All	-	display all queues
End	-	end the JBQ program
Help	-	display help information on the terminal
High	-	display the high priority job queue
List	-	display step information for a selected job name
Normal	-	display the normal priority job queue
Pre	-	display the pre-emptive priority job queue
Quit	-	end the JBQ program and logoff TIP/30

The JBQ program may be executed interactively or may be given a single command via the command line. If a single command is given on the command line, JBQ will attempt that command and then terminate normally. If used interactively, JBQ will prompt the user for each command until and "End" or "Quit" command is given.

3.26.1 DISPLAY ALL OS/3 JOB QUEUES — JBQ: all

This command will cause the JBQ program to display the status of all the OS/3 job queues (Normal, High and Pre-emptive priority). All jobs in each queue will be shown; those job names in parentheses are currently on hold.

Syntax:

All

Where:
No parameters required.

Example:

A
Will display all OS/3 job queues.

Error Conditions:
None.

-+*+-

3.26.2 END INTERACTION WITH JBQ PROGRAM JBQ: end

This command will cause the JBQ program to stop prompting the user for further commands and terminate normally.

Syntax:

End

Where:
No parameters required.

Example:

E

Will end the JBQ program.

Error Conditions:
None.

-+*+-

3.26.3 DISPLAY HELP INFORMATION ON TERMINAL JBQ: help

This command will cause the JBQ program to display help information on the terminal. The help information is a summarization of the recognized command syntax.

Syntax:

HElp

Where:
No parameters are required.

Example:

HE

Will display help information on the terminal. Note that "HE" is the shortest possible string of characters that may be entered for this command.

Error Conditions:
The help information may not be available.

-+*+-

3.26.4 DISPLAY HIGH PRIORITY JOB QUEUE JBQ: high

This command will display jobs that are in the OS/3 high priority job queue. Jobs currently held by the OS/3 operator will be displayed with the job name in parentheses.

Syntax:

High

Where:
No parameters required.

Example:

H

Will display the high priority job queue.

Error Conditions:
None.

-+*+-

3.26.5 LIST JOB STEP INFORMATION JBQ: list

This command will display step information about a selected job. The job may be queued for execution, rolled out, or executing. The information displayed includes the job status (executing, queued etc), the queueing priority of the job, the number of steps in the job and, for each step in the job, the program executed in that step, the LFD name of the step library and the switching priority (if currently executing.

Syntax:

List jobname

Where:

jobname the name of the selected job (8 characters max).

Example:

L TIP30

Will display the step information about job named "TIP30".

Error Conditions:
The specified job may not be found in any queue or may not be currently executing or rolled out.

-+*+-

3.26.6 DISPLAY NORMAL PRIORITY JOB QUEUE JBQ: normal

This command will display the jobs in the normal priority job queue.

Syntax:

Normal

Where:
No parameters required.

Example:

N

Will display the jobs in the normal priority queue.

Error Conditions:
None.

-+*+-

3.26.7 DISPLAY PRE-EMPTIVE PRIORITY JOB QUEUE JBQ: pre-emptive

This command will display jobs in the pre-emptive priority job queue. If the system was generated without pre-emptive job scheduling the display will indicate no jobs in that queue.

Syntax:

Pre

Where:
No parameters required.

Example:

P

Will display jobs in the pre-emptive job queue.

Error Conditions:
None.

-+*+-

3.26.8 END INTERACTION WITH JBQ — JBQ: quit

This command will cause the JBQ program to stop prompting the user for further commands. If the JBQ program is executing at program stack level one (ie: was NOT called by another program) the user will be logged off the TIP/30 system.

Syntax:

Quit

Where:
No parameters required.

Example:

Q

End JBQ program and logoff.

Error Conditions:
None.

-+*+-

3.27 INTERACTIVE JOB CONTROL SUBMITTOR JCL

The JCL program allows the user to enter job control statements at the terminal that will be submitted directly to the OS/3 run processor. This eliminates the necessity of creating an element in the YJCS library for quick one-time-only jobs. The JCL program calls the TIP/30 text editor (QED) to enable the user to create (or modify) a job control stream. When the editing session is completed, the JCL program submits the edit buffer to the TIP/30 librarian (TLIB) to be submitted to the remote batch reader queue. A facility for editing and resubmitting job control streams is provided so that the user need not re-type a control stream that was almost correct.

Syntax:

```
JCL  [file,element [,type]]   [,buffer]

          -OR-

JCL  [buffer]
```

Where:

file,element an optional file and element to initially read into the editor work buffer

type the type of element to read [default is source ("S")]

buffer the name of the edit buffer that will be accessed. If an edit buffer of that name does not exist, JCL will create it.

If the name is not specified, it defaults to "JCL$tttt" where "tttt" is the ICAM name of the submitting terminal.

Example:

```
JCL TEST
```

Will access or create as necessary the edit buffer named GROUP1/TEST (where GROUP1 is the name of the first elective group to which the user belongs). The TIP/30 text editor will be called and, if the user exits the editor with the "E" command, the buffer will be submitted to the remote batch reader.

Error Conditions:
None.

3.28 LOG OFF TIP/30 LOGOFF

The LOGOFF program is used to logoff TIP/30. Only users that have previously logged on may log off.

Syntax:

```
LOGOFF
```

Where:
No parameters are required.

Example:

```
LOGOFF
```

Error Conditions:
None.

Additional Considerations:

This program may only be executed in response to the standard system prompt. The logoff request will not be honoured at stack levels higher than one.

3.29 LOG ON TIP/30 SYSTEM LOGON

To be able to use the TIP/30 system, the user must first "logon". This may be accomplished by executing the LOGON program. The LOGON program requires the user to supply his user identification and current password. Other methods of logging on the TIP/30 system are described in Chapter II (logon and logoff procedures). The user id and password supplied by the user are checked for validity in the TIP/30 catalogue.

Syntax:

```
LOGON  userid/password [,account]
```

Where:

userid the user name (max 8 characters) assigned to the user by the installation administrator.

password the current password associated with the userid. The password is a maximum of 8 characters. It is quite legal to have a password that is all blank (omitted) but this is ill-advised since it provides rather minimal protection against unauthorized use.

account the account number the user wishes to associate with this session. The account number is defined by the installation administrator and may or may not be optional.

Example:

```
LOGON FRED/QWERTYUI
```

Will logon a user named "FRED" who has a password of "QWERTYUI".

Error Conditions:

The userid may not be recognized, the password may not match the current password for the userid, or the account number does not appear in a list of valid account for the userid. After three attempts, TIP/30 will lock the terminal keyboard and inform the OS/3 system operator that a logon attempt was unsuccessful at that terminal. The keyboard may be unlocked for more attempts by pressing the "KBOARD UNLOCK" key.

3.30 TIP MAIL SYSTEM MAIL

MAIL is a program designed to provide user to user communication through a mailbox file. Each user who wishes to receive mail must first create a mailbox using the mail program 'Create' command.

If mail is sent to a user who is not logged on, the message is simply stored in the user's mailbox. However, if the receiving user is logged on, the message is stored in his mailbox and he is informed that the message is available for him to read by way of the 'message/waiting' alarm.

Syntax:

command param

Where:

command is one of the following

Create Create a new mailbox. If the user already has a mailbox then this command has no effect. There are no parameters on the Create command.

? Check the mailbox to see how many new messages and how many old messages it contains. A new message is a message which has not yet been read. An old message is a message which has been read by the user.

Send USER-ID Send a message to a specific user. The Send function will display a blank screen into which the user can enter the message he wishes to send.

List List the directory of the mailbox.

List ? List all NEW messages.

List msg# List the message with the specified number.

List USER-ID List all messages sent by the specified USER-ID. An asterisk may be used to denote a prefix search. For example, if the user enters "L *ABC" then all messages from USER-ID's begining with the characters "ABC" will be listed.

List *	List all messages in the mailbox.
Delete	Delete all OLD messages. Note that MAIL will not allow you to delete a message he has not read.
Delete msg#	Delete the message with the specified number.
Delete USER-ID	Delete all messages sent by the specified user. As in the List command, an asterisk may be used to denote a prefix search.
Delete *	Delete all OLD messages in the mailbox.

Additional Considerations:

MAIL may be called to perform a single function by entering the desired function and its parameters in the command line.

MAIL uses only the first character of a command to identify the requested function.

3.31 OS/3 MEMORY DISPLAY MEM

The MEM program is a utility which displays the current OS/3 memory utilization (map). The program details job name, memory region in hex, size in decimal, type, program executing, CPU time, account number, storage protect key, executing and scheduling priority.

Syntax:

MEM [Wait] [,Buffers]

Where:

Wait If given causes the MEM program to refresh the display on the screen every 20 seconds or until a function key or msg-wait is pressed. This also causes the screen to be scrolled rather than rolled.

Buffers Optional parameter to cause program to display buffer pool information (Release 7 and above).

Example:

MEM

Will display the current OS/3 memory usage map.

Error Conditions:
None.

Additional Considerations:

To discontinue the memory display with the wait parameter, press a function key or msg-wait and reply "No" to the continuation prompt.

3.32 SPECIFY MODE OF OPERATION MODE

The MODE program is used to place a user in debug mode, to specify an MCS screen format to be used in place of the standard system prompt or specify a transaction program which will replace the standard system command processor (TCP).

Syntax:

MODE[,opt] [menu][,prog]

Where:

opt defines the mode of operation.

'DB' places the terminal in debug mode: all files are placed in debug mode. File updates are ignored.

no option; remove debug mode.

menu name of a message format which is to be used as the prompt message.

prog is the name of a prompt program which is to be called.

When this program terminates, the user will be logged off.

Example:

MODE ARMENU

This will change the standard system prompt for the user to the MCS screen format "ARMENU".

Error Conditions:
None.

3.33 SENDING A MESSAGE MSG

The MSG program allows a terminal user to send a message to another TIP/30 user, to a specific terminal, or to the computer operator. If the destination is not valid, the sender will receive an error message. When the message is received, it is prefaced by the USER-ID and terminal name of the sender.

Syntax:

MSG[/dest] text

Where:

dest is the name of the user (user-id), or the terminal name (as defined in ICAM) to which the message is to be sent.

'dest' follows standard prefix notation.

If the destination is not specified (omitted), the message will be sent to the computer operator.

text is the message (64 characters maximum) to be sent.

Example:

```
MSG/BETTY INVENTORY UPDATE IS COMPLETE.
MSG/TRM1  YOU CAN LOG ON NOW.
MSG   HOW LONG WILL THE SYSTEM BE UP?
MSG/*MF  MANUFACTURING FILES CLOSED!
```

Error Conditions:

No such user or terminal found.

Additional Considerations:

A message that is sent to the OS/3 operator may be split into two console lines if the text is too long.

3.34 MESSAGE ARCHIVER (LIBRARIAN) MSGAR

The message archiver (MSGAR) is a utility program that provides librarian services for TIP/30 screen formats (messages). Screen formats are stored in a partition of the TIP/30 catalogue file and may also be pooled in memory for fast access (refer to "TIP/30 System Generation").

The message archiver recognizes the following commands:

CURSOR	- specify cursor resting location for a message
DELETE	- delete a message
DIRECTORY	- print a directory of message names and information
END	- end interaction with MSGAR program
HELP	- display help information on terminal
LIST	- list message names and information
PRINT	- print a hard copy image of a message
QUIT	- end interaction with MSGAR program and logoff
RENAME	- rename a message
RESTORE	- restore a message from an OS/3 library element
SAVE	- save a message in an OS/3 library element
WRITE	- create a library element containing message names

The message archiver may be used interactively or may be given a single command on the command line. If a single command is given on the command line MSGAR will attempt only that command and then terminate normally. When used interactively, MSGAR will prompt the user for each command.

3.34.1 CURSOR RESTING LOCATION — MSGAR: cursor

This command specifies the cursor resting location for a message. The cursor location may be specified as a row and column (relative to 1) or may be omitted. If the row and column are omitted, the archiver will compute the resting location as the first position of the first unprotected field. If there are no unprotected fields in the message, the cursor will rest at (1,1).

Syntax:

CUrsor name [,row,column]

Where:

name the name of a single screen format (prefix specification not allowed)

[,row,column] resting location [home position is (1,1)]

Example:

cursor testmsg,5,51

Will force the cursor resting location for screen format named "TESTMSG" to row 5 column 51.

Error Conditions:
The specified screen format may not be found or the row and column specification may be incomplete or invalid.

-+*+-

3.34.2 DELETE SCREEN FORMAT MSGAR: delete

This command will delete a screen format. In order to minimize the possibility of inadvertent wholesale deletes, the screen format name for this command may NOT be given as a prefix specification.

Syntax:

DELete name

Where:

name the name of a single screen format (prefix specification not allowed)

Example:

DEL testmsg

Will delete the screen format named "TESTMSG".

Error Conditions:
The specified screen format may not be found.

-+*+-

3.34.3 DIRECTORY OF SCREEN FORMATS — MSGAR: directory

This command produces a printout containing information known about the selected screen formats. The information printed includes: screen name, author, date and time created, total data field count, etc.

Syntax:

DIrectory *name [,printer]

Where:

***name** a single message name or a prefix specification

printer the output printer destination. The default destination is PRNTR (the site printer). The printer may also be specified as an auxiliary print device.

Example:

```
DIR    !test,aux1
```

Produce a directory listing of all screen formats which have a name NOT starting with the string "TEST" The printout is to be directed to the auxiliary printer for the issuing terminal.

-+*+-

3.34.4 END MESSAGE ARCHIVER MSGAR: end

This command causes the message archiver to terminate processing normally.

Syntax:

End

-+*+-

3.34.5 HELP INFORMATION — MSGAR: help

This command causes the message archiver to display a summary of recognized commands and required parameter syntax.

Syntax:

Help

-+*+-

3.34.6 LIST SCREEN FORMAT INFORMATION MSGAR: list

This command displays (on the terminal) a summary listing of information known about the selected screen formats. The information is similar to that shown by the DIRECTORY command.

Syntax:

List *name

Where:

***name** a single message name or a prefix specification

Example:

LIST *test

Produce a listing of all screen formats which have a name starting with the string "TEST".

-+*+-

3.34.7 PRINT SCREEN FORMAT — MSGAR: print

This command will create a hard copy image of specified screen formats. The image is a representation of the screen as it was defined to the Message Definition program (see "MSGDEF"). Data fields and heading fields are shown with original edit and control information. The hard copy image may be routed to the site printer PRNTR (the default destination) or to an auxiliary print device (for example: AUX1).

Syntax:

Print *name [,printer] [,case]

Where:

***name** a single message name or a prefix specification

printer the name of the destination printer (default is PRNTR; other examples are: AUX1 AUX1*BYP etc.)

case a choice between "Upper" and "Lower" indicating the desired case of the printout. "Upper" is the default when the destination is the site printer; "Lower" is the default when the destination is an auxiliary printer.

Example:

```
PRINT  *test,,LOWER
```

Produce a hard copy printout of all screen formats with a name starting with the string "TEST" on the site printer and attempt to print lower case data.

-+*+-

3.34.8 QUIT MSGAR PROGRAM MSGAR: quit

This command will end the message archiver. In addition, if the user was executing the message archiver at stack level 1 (ie: the message archiver was NOT called from another program) then the user will be logged off the TIP/30 system.

Syntax:

Quit

-+*+-

3.34.9 RENAME SCREEN FORMAT — MSGAR: rename

This command will rename an existing screen format. The new name must be a name that is not currently in use.

Syntax:

RENname name,newname

Where:

name the name of an existing screen format

newname the desired new name for the screen format

Example:

ren testmsg,xtestmsg

Will change the name of screen format "TESTMSG" to "XTESTMSG".

-+*+-

3.34.10 RESTORE SCREEN FORMAT — MSGAR: restore

This command will restore a screen format that was previously saved (by the message archiver) in an OS/3 library element. The name of the element containing the saved screen format need not be the same as the name of the message. If the specified screen format already exists the user is asked whether or not the existing screen format is to be overwritten.

Syntax:

```
REStore    name      ,file      [,elt]
```

Where:

name the name of a single screen format (prefix specification not allowed)

file the logical file name of the OS/3 library

elt the name of the element in the library which contains the saved screen format (default name is same as message name)

Example:

```
REST   TESTMSG,PRODSRC/XTESTMSG
```

Will restore (recreate) a screen format called "TESTMSG" from library "PRODSRC" element "XTESTMSG".

-+*+-

3.34.11 SAVE SCREEN FORMAT **MSGAR: save**

This command will save one or more screen formats in an OS/3 library. Each selected screen format will be written to an element (default element name is the same as the screen name). The save command is useful for taking a backup of screen formats before undertaking extensive modifications or in preparation for transporting screen formats to another TIP/30 system.

Syntax:

Save *name ,file [,elt]

Where:

***name** the name of a single screen format or a prefix specification.

file the logical file name of the OS/3 library

elt the name of the element in the library which will be created containing the screen format. The element name will default to the name of the screen format that is being saved. If screen formats are selected by prefix the element name must be omitted.

Example:

SAVE *TF$,BACKUP

Will save all screen formats with a name starting with "TF$" into the OS/3 library catalogued with the logical file name "BACKUP". Each element will be created with the name of the screen format it contains.

-+*+-

3.34.12 WRITE SCREEN FORMAT NAMES **MSGAR: write**

This command will create an element in an OS/3 library which will contain all the specified screen format names. Each "line" of the created element will contain a single screen name. The write command is especially useful for creating command files for a subsequent run of the message archiver. The element created by the write command can be edited later using the TIP/30 Text Editor (QED).

Syntax:

```
Write      *name     [,file]   [,elt]
```

Where:

***name** the name of a single screen format or a name prefix specification.

file the logical file name of the OS/3 library (default is "RUN")

elt the name of the element to create (default is "MSGAR")

Example:

```
WR     !TF$,RUN/NONTIP
```

Will create an element named "NONTIP" in library "RUN" containing lines of message names that do NOT begin with the string "TF$".

-+*+-

3.35 MESSAGE DEFINITION MSGDEF

MSGDEF first prompts for the name of the screen format to be created or modified. To create a new display, enter its NAME in the "new" field and leave the "old" field blank. To modify an existing display enter its name in "old" field and leave "new" blank. To make a new display from an existing one, fill in both fields. Although a prompt screen is provided, the user may enter these two parameters on the initial command line, when first calling the program.

Defining a message can involve three or four steps. Each step will be preceded by a display describing both functions and options.

Step 1:

At this time the user must select various options which affect the entire the display. Step 1 functions through a 'fill in the blanks' style menu where options are simply selected from a list. (Some default options are given, but may be overlaid.) Place the cursor at the end of the screen and press transmit. The options are:

- ↑ is the character to be used in following steps to identify characteristics of portions of the display.

- \ is the character which may be used to represent a start-of-entry (SOE) character in the display.

- _ Enter Y if all data fields are to be unprotected.

- Y Enter Y if the entire screen is to be erased before the screen format is sent to a terminal.

- 01 Enter the row to which this display is to be transmitted.

- __,__ Enter the row and column where the cursor is to rest after the display is transmitted (MSGDEF will normally compute this).

- nn,nn is the row and column where the cursor used to be set (this is displayed here on old screens by MSGDEF).

- __ Enter display intensity for data fields (UTS-400 typically). TN is Tab (in front of fields) and Normal intensity. TL is Tab (in front of fields) and Low intensity. N is no Tab and Normal intensity. L is no tab and Low intensity.

- _ Enter the display intensity for heading fields (UTS-400 typically). N for normal and L for low.

- _ Enter 'Y' if you want the informational messages which are displayed between each step of MSGDEF processing.

Step 2:

If a 'new screen' is being generated, step 2 will present an entirely blank screen upon which the user designs the screen layout; otherwise, the previous definition will appear on the CRT. Create or modify the CRT data on the terminal until the desired screen is complete with both field headings and data fields. Define the data fields using the following Field Definition Codes (FDC's). (Data fields are those which are to be filled in by a terminal operator or program.) When this entire process is complete, place the cursor immediately behind the last field defined on the CRT and TRANSMIT it into MSGDEF for analysis.

FDC		Meaning
U	-	defines an upper case alpha-numeric data field. Any lower case alpha input is forced to upper case.
X	-	defines an alpha-numeric data field.
E	-	defines an ERROR field. E fields are displayed using the TIPMSGE call.
Z	-	numeric only, on output leading zeros will be suppressed.
9	-	numeric only, right justified, zero filled
2	-	numeric only, right justified, zero filled (not zero filled if blank)
,	-	comma to be edited into numeric field if significant
-	-	reserve room for leading minus sign
.	-	decimal point to be edited into numeric field, This will be used to determine the number of decimal places in the field.
/	-	edit character for numeric field
:	-	edit character for numeric field
B	-	defines an upper case alpha-numeric data field which is to blink
0	-	defines an alpha-numeric data field, which is to have the display turned off.

A - defines an upper case alpha-only field.

_ - defines an unprotected underscore heading character

Note: codes B, O, and A are only effective if your terminal has such capabilities (UTS-400's typically).

Note: numeric fields are zero suppressed during output (TIPMSGO) and zero filled during input (TIPMSGI). A negative numeric field is displayed with a leading minus (-) sign. To allow room for this minus sign, it may be necessary to begin the field definition with a comma, to force it one postion larger on the terminal without affecting its size in the program.

Step 3:

At step 3, the user must indicate which portions of the designed screen really represent data fields. MSGDEF may have taken poetic licence during step 2's analysis of headings versus data fields. The user does this by overlaying the Field Definition Codes with the option character (usually ↑) and TRANSMITing the defined screen back in to MSGDEF again. MSGDEF will have attempted to sort out the differences automatically and will have overlaid all sequences of 2 or more " U's, E's, X's and digits " with the option character. Check to be sure that it was not too clever and overlaid something it should not have.

Step 4: (optional)

This step will only be entered if the user indicated in step 1 that not all data fields were unprotected. At step 4, the user must indicate which portions of the display are to be unprotected. As in step 3, this is done by overlaying the FDCs with the option character and TRANSMITing the design screen back to MSGDEF. Just before entering step 4, MSGDEF will prompt the user to request whether the automatic option character processing should be done. If the 'SOE ↑' choice is made, MSGDEF will attempt to automatically define, by overlay, the unprotected fields. As before, MSGDEF replaces all sequences of 2 or more " U's, E's, X's and digits " with the option character. Check to be sure that it was not too clever and overlaid something it should not have. Remember to TRANSMIT the final screen back to MSGDEF for inclusion into TIP's screen file.

Editing the message

For users who have older CRT's, limited in editing features, MSGDEF implements certain hardware extensions via software with function keys. For example,

- F1 - 'insert line in display' wherever the cursor is resting.

- F2 - 'deletes line in display' where the cursor is resting.

Many displays are of a repetitive nature with lines containing similar data from several records from a file. On UTS-400's, the LINE-DUP key is very helpful in defining this type of display. Older CRT's do not have such a feature. However, even the UTS-400 will not allow duplication of a set of lines. MSGDEF therefore has been written to provide both single and multi-line duplication facilities for all terminal types. Function keys 3 and 4 together provide this capability;

- F3 - save line(s) - place the cursor on the last column of the line above the line(s) to be saved and press F3. This begins the save function by sending a SOE to the cursor position. Then place the cursor on the 2nd last column of the last line to be saved and press TRANSMIT. MSGDEF will then save those line(s) and erase the SOE character.

- F4 - to recall the lines saved by F3 and TRANSMIT, place the cursor on the last column of the line above the line where the saved lines are to be repeated and press F4. For multiple copies keep pressing F4.

3.35.1 MESSAGE DEFINITION **Negative Fields**

There are several ways to display negative numeric data fields. You may select one of: trailing minus sign, leading minus sign, trailing "CR", trailing "DB", or enclosed in parentheses "( )".

The minus sign may be placed at either the beginning or end of a numeric field during step 2 and then overlaid during step 3.

The letters CR or DB may be placed at the end of a numeric field and then overlaid during step 3.

To get parentheses place a ")" at the end of the numeric field (and leading minus sign if there is a possibility that the field may require all available digits) and overlay during step 3.

In all cases, the negative editing is used during output only if the data is negative. During input processing the program will receive negative data by any of the above methods (when keyed by the terminal operator).

On FCC type terminals, a minus sign is the only allowable method since TIP/30 sets up numeric fields using FCC codes.

-+*+-

3.36 MESSAGE TESTING MSGSHOW/MSGTST

To test a display created by MSGDEF enter 'MSGTST name' or 'MSGSHOW name'; where "name" is the name of your screen format. MSGTST will prompt the user for test data and present it on the CRT using the named screen format. MSGSHOW expects the user to fill in the test screen and displays the data a program would receive back. In either case, the unformatted data screen expects the test data to be a continuous character string. A user can cycle back and forth between screens trying various data entry options.

When the user's formatted message is displayed, intentional errors may be introduced to check error field options. Entering a "↑" (circumflex) as the first character in a field and pressing transmit will cause the field to blink and an error message to be displayed.

MSGTST and MSGSHOW display the data received exactly as it would appear in a user program input buffer. Note that no header information or communications characters are received, and that the number of characters sent is a function of cursor position. Numeric fields are returned to the user right justified and zero filled. Data characters entered into a field which are incompatible with the field definition are replaced by blink characters (or are blinked) by the Message Control system. The errors may be corrected and data changed to try out various options available. Simply place the cursor behind the data and TRANSMIT.

```
MSGTST      name  [,_]  [, ]
MSGSHOW     name  [,_]  [, ]
```

The underscore in the second parameter is used to cause the program to display the format with an underscore character in the MCS-FILLER field of the MCS packet.

An optional third parameter may be supplied which will be used as the MCS-FUNCTION value.

If further information is desired for either program, place a question mark on the call line as the MCS-FILLER (causing an error) and invoke the program. This will solicit a HELP screen which details other options including the definition of the function keys. Remember to fix the question mark when the parameter screen is presented.

3.37 SPECIFY CHANGE IN USERID AT TERMINAL NEWUSER

There is often the necessity to terminate the current session (logoff) and start another session (logon) using a different user-id or account number. To simplify this process, the NEWUSER transaction has been provided.

NEWUSER enables a logged on user to logoff and logon in one step.

Syntax:

NEWUSER userid [/password] [,account]

Where:

userid the user name (max 8 characters) assigned to the user by the installation administrator.

password the current password associated with the userid. The password is a maximum of 8 characters. It is quite legal to have a password that is all blank (omitted) but this is ill-advised since it provides rather minimal protection against unauthorized use.

account the account number the user wishes to associate with this session. The account number is defined by the installation administrator and may or may not be optional.

Example:

NEWUSER FRED/QWERTYUI,A106

Will logon a user named "FRED" who has a password of "QWERTYUI" using the account code "A106".

Error Conditions:

The userid may not be valid, the password does not match the current password for the userid, or the account number may not be valid for the specified user-id. If any of these errors occur, TIP/30 will present the user with the logon screen format (the implied logoff will have been successful).

3.38 INFORMATIONAL MESSAGE NOTE

The NOTE program allows a terminal user to send a message to the terminal which invoked the NOTE. This command is designed for use within DOT-IN files (See section on DOT-IN) to notify the user that various functions being requested have been performed.

Syntax:

NOTE[,W] text

Where:

W is used to specify that the user must generate some input from his terminal (usually a function key or MSG-WAIT) before TIP will continue to process the next command. The text is displayed and the NOTE program will wait for any input.

text is the message (64 characters maximum) that is to be displayed on the terminal.

Example:

```
NOTE   ALL USER-IDS HAVE BEEN CATALOGUED
NOTE,W PRESS MSG-WAIT TO CONTINUE
```

Error Conditions:

None.

3.39 ON-LINE DATA DISPLAY ODD

ODD provides the capability of performing general inquiry/update functions on any indexed file (ISAM, IRAM, MIRAM) with a minimum of time and effort. The TIP/30 user need only define the record format in a COBOL style definition language and define the screen displays with the TIP/30 Message Control System (MSGDEF). By using MCS, the TIP/30 user is able to instruct ODD to present data the way the end user would like to see it. The user is not restricted by a pre-defined display style.

This program is a preliminary version of the TIP QUERY LANGUAGE (TQL) and will be replaced by TQL. The user is advised to use this program only if his requirements cannot be (currently) met by TQL.

The user is not required to do any programming. Instead he defines his record layout in COBOL format followed by a few simple ODD directives. QED is used to enter these definitions; perhaps using some existing COBOL COPY element as a basis.

The ODD definition may be stored in a permanent OS/3 library; however, the library/element must be placed in a QED edit buffer when ODD is invoked. The edit buffer must be specifically named, but may differ from the library/element name. It is mandatory to create the edit buffer in the group "DBA" (Data Base Administrator). Normally one individual would be designated as the 'D.B.A'. This individual would have the responsiblity of maintaining the ODD definitions and setting naming conventions.

The reason for using an edit buffer to hold the ODD definition is simply to enhance the speed of retrieval for the compilation. When ODD begins execution, it takes a name from the command line prefixes GROUP-ID of DBA and then looks for the QED buffer of that name. When it finds the definition, it then compiles it into memory and into a work file for the duration of the session.

Suppose a definition element of 'DDF123' exists in a library 'DDFSRC' and that users want to call it 'PAYMAST' under ODD. The following QED command line would be used to setup the edit buffer the first time:

```
QED DDFSRC/DDF123,,DBA/PAYMAST
```

and the end user would enter,

```
ODD PAYMAST   or   OPEN PAYMAST
```

If changes have to be made to this definition, the QED command line would be:

QED ,,,DBA/PAYMAST

In either case, the editor must be terminated with the 'E' command to leave the edit buffer intact.

Compiling the definition at execution time makes it easier for the D.B.A to keep the definitions up to date. No batch job need be run to implement a new definition or change an existing one. Everything is done online at the terminal.

As mentioned earlier, in addition to the file definition, ODD requires a few directives. These must follow each record definition and contain "*/" in columns 7-8 (QED - Cobol mode).

ODD directives indicate:

Key field
Record-type field
Fields to be displayed
Field display sequence
MCS display name

The user may browse through indexed files using multiple record formats; in the case of MIRAM, browse using multiple keys.

The record definition source element may begin with

INDENTIFICATION DIVISION.

Next, identify the FD of the file to be processed. This is the logical name as defined in the TIP/30 catalogue.

FD. lfname.

If the file is not to be updated this is indicated by a statement which contains

READ-ONLY.

If records may be not be deleted include a statement which contains

NO-DELETE.

If records may not be added you should enter a statement which

contains

NO-ADD.

When ODD is searching a file it may operate in one of two modes, normal or read ahead. If your requests usually require several screens of data to be displayed you may specify the following in the definition.

READ-AHEAD-ON.

If this is specified, then while you are viewing one screen of data, ODD will read ahead in the file and collect the data for records on the next screen. When the next complete screen has been collected, the wait light be illuminated on a Uniscope terminal. Pressing function key 2 will cause the data to be displayed; ODD will continue with the next screen full. To stop the read ahead process before wait light notification, press message waiting; then enter your next request (which may be message waiting again to go back to the menu of commands).

NOTE: the read ahead feature is excellent for file browsing but becomes awkward if the intent is to update the file. Therefore, it is recommended that this feature NOT be used if the file is to be updated.

After ODD has compiled the specified data definition, the user may interactively specify:

READ-AHEAD-ON

to switch to read ahead mode or:

READ-AHEAD-OFF

to switch back to normal mode. To update records, processing must be in 'normal' mode.

ODD will always prompt with a M.C.S. display format. The default display used is named TF$ODD1. The user has the option of designing and specifying his own prompting display. This may be useful for providing instructions to the end user of the system. To specify your own prompting display provide a 'MENU' statement after the 'FD' statement.

Example:

```
MENU IS PAYDDMNU
```

Comment statements may be included in the definition. Since record and field names are used as part of the interactive search criteria specification, it is recommended that the names be 16 characters or less in length and be easy to remember. For example:

```
DISPLAY MAST-PAY IF PAY GT 24000 AND HIRE-DATE LE 68
```

Group names may be left in the definition, but ODD views them as comments and does not store the identifier name.

Multiple files may be processed. However the first record described must contain the full key of the secondary files. To define a secondary file, include a statement such as:

```
FD file2 POINTER is field1.
   ...record layout for secondary files...
```

Following each record definition identify which fields make up the key. This information is also placed on an ODD directive statement (ie. */ in columns 7,8).

```
KEY IS NUMBER NAME
```

If your are processing a MIRAM file which has a multiple index, specify all of the keys. This is done as follows:

```
KEY1 IS NUMBER
KEY2 IS NAME, CODE
KEY3 IS ADDRESS
```

Next, indicate how ODD may verify the record type. If all records have the same format then this directive is not required. Several identifying conditions per record may be specified, however, only one condition need be satisfied for ODD to select that record. There is a logical 'OR' relationship between the ID statements. Note that an ID statement can not span more than 1 line.

```
ID IS TYPE = 'A6'
ID IS TYPE = 'A6' AND HOURS 100
ID IS TYPE = 'A6' AND HOURS >= 100
ID IS TYPE = 'B0'
```

Next, specify how many DATA SETS the display which you have defined, can hold; also the field names and order in which they are to be displayed on the screen. A DATA SET is defined as a set of fields from one record. If no field names are given, ODD will assume that all fields of the record are to be displayed. Example:

```
DISPLAY 16 (NAME ADDRESS NUMBER) USING MYMESG
```

In this example the fields 'NAME ADDRESS NUMBER' from 16 indexed records will be displayed on the screen MYMESG; perhaps 16 lines of 3 columns, each under headings.

Several types of displays may be extracted from the same record definition. To distinguish between them you must name each by preceding the keyword "DISPLAY" with a LABEL, followed by a semi-colon.

Example:

```
TOTAL: DISPLAY 16 (NAME RATE HIRE-DATE) USING MYMESG
```

Note that it is possible to use the same display (see MYMESG in the previous examples), but different data fields. If the display name is not specified, the record name is used. Duplicate display names are not allowed.

Data Definition Examples:

```
     IDENTIFICATION DIVISION.
     FD. AR000.
     01  REC-A.
         05          SORT-KEY.
           10        PHONE          PIC 9(7)        COMP-3.
           10        ACCT-NO        PIC X(8).
           10        INV-NO         PIC X(8).
           10        REC-TYPE       PIC X.
           10        INV-DTE        PIC 9(6)        COMP-3.
         05          NAME           PIC X(52).
         05          ORDER-TYPE     PIC X.
         05          CLASS          PIC XXX.
         05          REF            PIC X(20).
         05          PO-NO          PIC X(16).
         05          AR-STATUS      PIC X.
         05          START-DTE      PIC 9(6)        COMP-3.
         05          TIMES-RAN      PIC 99.
         05          UNITS-CDE      PIC X.
         05          UNITS          PIC S9(7)       COMP-3.
         05          INSER-DTE      PIC 9(10)       COMP.
    */ KEY IS PHONE ACCT-NO INV-NO REC-TYPE INV-DTE
    */ ID IS REC-TYPE = 'A'
    */ DISPLAY 4 (PHONE      ACCT-NO   INV-DTE  NAME
    */             ORDER-TYPE CLASS     REF      PO-NO
    */             START-DTE  TIMES-RAN) USING AROMSGA
     01  REC-B.
         05          SORT-KEY.
           10        PHONE          PIC 9(7)        COMP-3.
           10        ACCT-NO      PIC X(8).
           10        INV-NO         PIC X(8).
           10        REC-TYPE       PIC X.
           10        INV-DTE        PIC 9(6)        COMP-3.
         05          PAYMNT-DATE-1   PIC 9(6)         COMP-3.
         05          PAYMNT-AMNT-1   PIC 9(7)V99      COMP-3.
         05          FILLER         PIC X(88).
    */ KEY IS PHONE ACCT-NO INV-NO REC-TYPE INV-DTE
    */ ID IS REC-TYPE = 'B'
    */ DISPLAY 16 (PHONE      ACCT-NO   INV-DTE  INV-NO
    */      PAYMNT-DATE-1 PAYMNT-AMNT-1) USING AROMSGB
```

Second example:

```
 IDENTIFICATION DIVISION.
 FD. AOM00.
 READ-ONLY.
*****************************************************************
***                O R D E R   M A S T E R   R E C O R D      ***
*****************************************************************
 01  ORDER.
       10          KEY.
         15        ACCT                PIC X(8).
         15        NO                  PIC X(8).
       10          STATUS              PIC X.
       10          TYPE                PIC X.
       10          ADJ-TYPE            PIC X.
       10          ORIGIN              PIC XXX.
       10          PROD-CDE            PIC XXX.
       10          BRAND-CDE           PIC X.
       10          AGT-CDE             PIC XXX.
       10          TEXT                PIC X(20).
       10          UNITS               PIC 999.
       10          UNITS-CDE           PIC X.
       10          SALESPER            PIC XXX.
       10          PREPRICED-AMT       PIC S9(7)V99   COMP-3.
*/ KEY IS ACCT  NO
*/ DISPLAY 1 USING OASODD
**
*/ SUMMARY: DISPLAY 19 (ACCT NO STATUS TYPE BRAND-CDE
*/                      AGT-CDE TEXT) USING OASODD2
```

3.39.1 ON-LINE DATA DISPLAY — Command Format

Most of the commands follow the same syntax.

Syntax:

```
<cmd> <display> [IF <selection>]
                [BY <KEYm>]
                [FROM <keyvalue>]
                [TO <keyvalue>]
                [SUM <field>]
```

Where:

<cmd> is the command name. Most commands may be truncated to the first two letters. You may also enter a display name and ODD will perform LIST <display>.

All commands are documented in the following sections.

<display> this is the label of the display statement in the definition; essentially identifying the fields of the record to be displayed and the message format used. However, the display command will only display one record on the screen. This is the record which would be updated if the update command was used next.

IF <selection> selects records based on a conditional expression. A simple conditional expression is a comparison between a field and some value, or one field and another. In ODD two fields must be of the same type and size to be compared to each other, and the left operand must always be a field name of the record displayed. The comparison operators may be

EQ = - equal to

NE <> - not equal to

GT > - greater than

GE >= - greater than or equal to

LT < - less than

LE <= - less than or equal to

Conditional expressions may be combined by Boolean operators (AND, &, OR, |) into more complex expressions.

BY <KEYn> indicates that the file is to be processed by an alternate index. 'KEYn' may be one of the reserved words KEY1, KEY2, KEY3, KEY4, or KEY5. 'KEYn' may also be an actual data field name which is also a key field, such as NUMBER.

If 'BY' is used then it must precede 'FROM' and 'TO' in order for ODD to know how to interpret the key field.

FROM <keyvalue> indicates the first record to be considered for displaying. The key may be enclosed in single or double quotes. The key may be a period (.), which means that the search is to continue from the last record displayed.

TO <keyvalue> indicates the last record to be considered for displaying. The key may be enclosed in single or double quotes.

SUM <field> specifies that the named 'field' is to be summed. At the end of the display the total value of this field for all records selected is displayed along with the average value of the field. If you wish to sum several fields use this clause several times.

-+*+-

3.39.2 ON-LINE DATA DISPLAY ODD: add

The ADd command allows the you to place new records in the file. It will display an empty screen which must be filled in and transmitted to the program.

Syntax:

ADd <display>

Where:

<display> the screen is displayed with no data, fill it in and press transmit.

-+*+-

3.39.3 ON-LINE DATA DISPLAY — ODD: close

The CLose command will terminate the ODD session.

Syntax:

CLose

Where:
None required.

Additional Considerations:

Note: The CLose command and the ENd command are synonyms.

-+*+-

3.39.4 ON-LINE DATA DISPLAY **ODD: count**

The COunt command will count records based on the selection criteria, starting position, & ending position in the file.

Syntax:

```
COunt <display> [IF <selection>]
                [BY <KEYn>] [FROM <keyvalue>]
                [SUM <field>]  [TO <keyvalue>]
```

Where:
Refer to command format.

Example:

```
COUNT REC-A IF TIMES-RAN > 5
  SUM BASIC-CHRG  SUM TOTAL-CHRG
```

-+*+-

3.39.5 ON-LINE DATA DISPLAY — ODD: delete

The DElete command will re-display the last record selected and prompt for the 'YES' to delete it. If this is the record to delete then transmit 'Y' back to the program. The record is deleted from the file and the menu is re-displayed for the next ODD command. The delete will only be succesful if the file was generated with the 'DELETE' parameter specified on the 'FILE' statement in the TIP/30 generation. This command must be used without READ-AHEAD.

Syntax:

DElete

-+*+-

3.39.6 ON-LINE DATA DISPLAY **ODD: display**

The DIsplay command will select a record based on the selection criteria and starting position in the file.

Syntax:

```
DIsplay <display> [IF <selection>]
                  [BY <KEYn>] [FROM <keyvalue>]
                  [SUM <field>]  [TO <keyvalue>]
```

Where:
Refer to command format.

Example:

```
DISPLAY SUMMARY IF TIMES > 50 AND UNITS = 9
                FROM 71219348
                TO   75225543

DISPLAY MAILTO IF TIMES LT 32 & UNITS NE 0
                BY ADDRESS
                FROM HARCOURT TO   MILLVIEW
```

The text of latest request to ODD is saved and may be recalled by pressing function key 3 on the terminal.

-+*+-

3.39.7 ON-LINE DATA DISPLAY — ODD: list

The LIst command will select a screen full of records based on the selection criteria and starting position in the file.

Syntax:

```
LIst <display> [IF <selection>]
                  [BY <KEYn>] [FROM <keyvalue>]
                  [SUM <field>]  [TO <keyvalue>]
```

Where:
Refer to command format.

Example:

```
LIST SUMMARY IF TIMES < 75 AND UNITS NE 9
             SUM PREPAY-AMT SUM PREPICED-AMT
                 FROM 71219348 TO   75225543
```

The text of this command is saved by ODD and may be recalled by pressing function key 3 on the terminal.

-+*+-

3.39.8 ON-LINE DATA DISPLAY ODD: next

The NExt command continues the DIsplay or LIst command from the last record displayed.

Syntax:

NE×t

Where:
No parameters required.

-+*+-

3.39.9 ON-LINE DATA DISPLAY — ODD: print

The PRint command will build full displays of records, (as the LIst command does), based on the <selection>, starting position, & ending position in the file. When a display is collected it will be sent to the terminal with a print command to print on the auxiliary printer attached to the terminal. The print command will continue to print all records from the file which satisfy the selection criteria.

Syntax:

```
PRint <display> [IF <selection>]
                [BY <KEYn>] [FROM <keyvalue>]
                [SUM <field>]  [TO <keyvalue>]
```

Where:
Refer to command format.

Example:

```
PRINT REC-A IF TIMES-RAN = 0
```

-+*+-

3.39.10 ON-LINE DATA DISPLAY **ODD: show**

The SHOW command allows you to get a list of all available display names in the current definition. Or you may get all of the field names within a given display.

Syntax:

SHOW <display>

Where:

<display> from which you want the field names

Syntax:

SHOW

Where:
to get summary of all display names

-+*+-

3.39.11 ON-LINE DATA DISPLAY **ODD: update**

The UPdate command will re-display the last record selected. You may then update the information on the screen and transmit. The updated record is written to the file and the ODD menu is displayed for the next command. This command must be without READ-AHEAD.

Syntax:

UPdate

Function key 4 may be pressed after a record is displayed. This will re-display the same record for update.

If you decide not to proceed with the update press MSG WAIT to cancel the update.

-+*+-

3.39.12 ODD COMMAND LINE FORMAT ODD

When ODD starts up it reads and compiles the source element specified on the command line and stores the record definition in memory. This approach makes it very easy to change definitions using QED.

You should use the editor to create and maintain the data file definition and use the Message Control System to define the format of the messages used by ODD to display the user's data in a format which is useful and easy to read.

To use ODD, the definition must be stored in an edit buffer which was created with a GROUP-ID of DBA. The ODD call uses the edit buffer name.

For example:

```
ODD AOMDEF
```

-+*+-

3.39.13 ODD FUNCTION KEYS ODD

MSG-WAIT this always means to cancel your most recent request. (Ie. cancel record update, stop searching file, etc..)

F1 or F5 Re-display the most recent message. If a LIst or DIsplay was last entered F1 will also step backwards through all displays given since the original request was given.

F2 or F6 During LIst or DIsplay this will cause the next full screen of data to be displayed.

F3 or F7 will return to the menu and display the most resent request.

F4 or F8 If a record was just displayed, this will re-display the record for update.

-+*+-

3.39.14 PROGRAM LIMITATIONS ODD

ITEM	MAXIMUM	
RECORD TYPES	15	
FIELDS PER RECORD	230	
DISPLAYS	20	
SEARCH CONDITIONS	40	
VARIABLE DATA AREA	600	BYTES
KEY SIZE	256	
RECORD SIZE	2560	
DATA IN MESSAGE	1792	
CHARACTERS PER NAME	16	
PROGRAM SIZE	23500	

-+*+-

3.39.15 ODD - PITFALLS TO AVOID **ODD**

Some syntax errors may cause ODD to abort - be careful when entering commands. It has been found that by cataloging 'ODD' and/or 'OPEN' with EDIT=YES, some problems can be avoided.

Further limitations are:

- maximum of 15 digits may be entered to be compared to a numeric field (if this becomes a problem change the field from PIC 9 to PIC X if not COMP or COMP-3).
- maximum of 15 digits may be entered in the FROM and/or TO clauses.
- ODD truncates field names to 16 letters and does not tell you.
- ODD does not handle some field descriptions very well. (For example SV99 should be coded as 99.
- ODD becomes confused when DISPLAY field definitions entered on the CRT go past column 70.

-+*+-

3.40 POST MORTEM DUMP ANALYSIS PMDA

PMDA is a dump analysis program that enables a programmer to interactively examine a dump from an on-line program. PMDA is automatically invoked by TIP when a user program aborts. PMDA creates a dynamic file containing a copy of the user program memory areas at the time of the dump. The dynamic file is created with a name constructed as: "userid/DUMPtttt/trid" where "userid" is the userid of the user executing the program that aborted, "tttt" is the ICAM terminal name of the user terminal, and "trid" is the catalogued transaction name that invoked the program that aborted.

If the user is an application level user PMDA merely prints the dump at the site printer and ends processing at that point. However, if the user is a programmer level user, PMDA will allow the user to enter commands to 'browse' through the dump at the terminal. The programmer level user may specify that the dump file is to be printed and/or kept. PMDA may be invoked directly from the terminal to browse through a previously kept dump file. Another important function of the PMDA program is to roll back any file updates that the aborted program may have done and to release any files that may have been assigned to the program.

PMDA is most often encountered as a result of a program aborting. However, it is possible to execute PMDA directly as a transaction to continue analysis of a previously retained dump.

To execute PMDA interactively, the command line syntax is:

Syntax:

```
PMDA   trid  [,tttt] [,userid]
```

Where:

trid the name of the transaction that aborted

tttt the ICAM name of the terminal where the original abort occurred (default is the current terminal).

userid the userid of the user that was running the program at the time the program aborted (default is the current userid).

PMDA recognizes the following interactively submitted commands:

Display	-	display area of memory
End	-	end interaction with PMDA (retain dump file)
Print	-	print hard copy dump
Quit	-	end interaction with PMDA (scratch dump file)

Most programmers find that it is generally advisable to print a dump whenever a transaction program aborts. In some cases, it is possible to browse through the dump at the terminal and discover the cause of the dump (and therefore eliminate the need to print the dump).
Some familiarity with assembler programming concepts is assumed in the following discussion of PMDA commands.

DISPLAY MEMORY CONTENTS

3.40.1 DISPLAY MEMORY CONTENTS — PMDA: display

This command enables the user to display the contents of the memory allocated to the program that aborted. The display command has several variations (which are described below) enabling the user to specify storage, registers etc to display.

D address — display 16 bytes in hexadecimal and graphic from the specified address.

D name [,offset] — display 16 bytes in hexadecimal and graphic from the start of the linkage area given by <name> plus optional offset. The recognized names are: PIB CDA MCS IMA OMA WORK. Offset is specified in hexadecimal; if omitted, the offset defaults to zero.

DF — Display the contents of the floating point registers.

D PSW — Display the abort address and the PSW at time of abort.

DR — Display the contents of the general purpose registers.

Example:

D 5800

Will display 16 bytes starting at address X'5800'.

Example:

D MCS,40

Will display 16 bytes at offset X'40' from the start of the MCS linkage section area.

-+*+-

3.40.2 END PMDA PROGRAM PMDA: end

This command will end interaction with PMDA and keep the retain the dynamic file containing the dump for later analysis.

Syntax:

End

Where:
No parameters required.

Example:

E

Error Conditions:
None.

-+*+-

PRINT HARD COPY DUMP

3.40.3 PRINT HARD COPY DUMP — PMDA: print

This command will cause PMDA to create a printed dump for off-line analysis by the programmer. The dump will be printed to the site printer. The dump will be formatted for ease of analysis; major areas of storage will be identified in much the same fashion as OS/3 SYSDUMP.

Syntax:

Print

Where:

No parameters required.

Example:

P

-+*+-

3.40.4 END PMDA AND SCRATCH DUMP FILE PMDA: quit

This command will cause PMDA to scratch the temporary dump file and end interaction with the user. This command is most useful after issuing a PRINT command.

Syntax:

Quit

Where:
No parameters required.

Example:

Q

Error Conditions:
None.

-+*+-

3.41 TIP/30 TEXT EDITOR QED

QED is an interactive program for creating and modifying 'text', using directives provided by a user at a terminal. The text may be a program, a runstream, a document or perhaps data for a program.

The TIP/30 editor is patterned after 'QED' by BELL Labs. You may also find it similar to the editors used on many mini-computers and micro-processors; being primarily line number independent and string/contextually oriented.

First, a bit of terminology. In QED the text being processed is said to be kept in 'the buffer'. Think of the buffer as a work space, or simply as the information to be edited. In effect, the buffer is like a piece of paper on which things are written, changed and finally 'filed' for future reference.

The user interfaces with QED and his text via the QED command language. Most commands consist of a single letter, which may be typed in either upper or lower case. Generally (although not always) each command is typed at the beginning of a new line and is followed by a transmit. (Sometimes the command is preceded by information about what line or lines of text are to be affected). At the completion of each command, QED will respond by moving the cursor to a new line and prompt with a ? (question mark).

This document was developed in a tutorial style and it is recommended that the user actually perform the examples and exercises described in the manual. In the QED examples which follow, the User will notice both upper and lower case letters being used. QED accepts either, for commands or text.

3.41.1 GETTING STARTED **QED: intro**

After logging on to TIP/30 (see USER GUIDE), activate QED by entering

QED <elt>

to create a new element or

QED FILE/ELT

to update an existing element.

If the element to be updated is a macro (or proc) then parameter three must be ',M' or ',P' respectively (',S' meaning 'source' is assumed default).

QED proclib/macroname,M

The fourth and fifth parameters identify the group id. and edit buffer name to QED. If the user only provides the fourth parameter to 'name' the buffer, QED defaults to the User's group id. at LOGON time. For example

QED FILE/ELT,,RCVNAM

or

QED FILE/ELT,,DBA/DATADEF

In the event of a system crash, the buffer may be recovered by logging back on to TIP/30 and calling QED using the same buffer name. If no buffer name is given, QED will default the buffer name to the element name provided. Using the examples above, the QED calls would be

```
QED ,,,RCVNAM

     or

QED RCVNAM

     or

QED ,,,DBA/DATADEF
```

QED will respond by displaying the name of the edit buffer file assigned and then prompt the user with a ?SOE sequence to indicate its ready (command mode) state. It is preferable to start a new program by the second method, to ensure that the edit buffer will have a reasonable name.

-+*+-

3.41.2 QED CONTROL CHARACTER, DOUBLE QUOTE QED: "

The double quote (") is the QED control character and requires care in its use, even when adding text. Why this is so, should become clear later on!

-+*+-

3.41.3 ERROR MESSAGES QED: errors

QED will respond to user command errors by displaying a self explanatory error message.

-+*+-

3.41.4 LINE LENGTH QED

The current version of QED copies the first 80 characters of each record to the edit buffer. Different sizes are displayed depending on the mode you set with the option command (OR,OC,OA):

RPG - 6 to 74
COBOL - 7 to 72
BAL - 1 to 72

-+*+-

3.41.5 ADDING TEXT; THE ADD COMMAND QED: a

It means 'Add' (or Append) lines to the edit buffer.

To enter lines of text, just type an 'A' and transmit. QED will prompt with an 'A' (which replaces ? while in Append mode) at the beginning of each new line to be entered. The user may enter data a line at a time or by the screenful. Just follow the last text line with a "F as follows:

```
A
now is the time
for all good men
to come to the aid of their party.
"F
```

(The "F is the QED command which says 'end Add mode'.)

After the Add mode has been ended, the edit buffer will contain the three lines:

```
now is the time
for all good men
to come to the aid of their party.
```

Of course the 'A' and '"F' are not there, as they were not text but QED commands.

To append text to data already in the buffer, just issue another 'A' command and continue. To append at a specific line, give its number and follow it with an 'A'.

-+*+-

3.41.6 DISPLAYING LINES; THE PRINT COMMAND QED: p

To print or display the contents of the QED buffer at the terminal, use the print command 'P'

As follows: specify the lines where printing is to begin and end, separated by a comma, and followed by the letter 'P'. Thus to print the first two lines of the buffer for example,

```
1,2P      (starting line=1, ending line=2 p)
```

QED will display:

```
now is the time
for all good men
```

and prompt the user for the next command.

To print all the lines in the buffer, 1,3p may be used since the exact number of lines is known. Normally this is unknown; therefore, QED provides a shorthand symbol for 'line number of last line in buffer' - the dollar sign $. Use it this way:

```
1,$P
```

This will print all the lines in the buffer (line 1 to last line).

NOTE: QED will check for unsolicited input after every 14 lines of continuous display. To end the command being serviced, press any one of the function keys on the CRT, or the break or attention key on other terminals. QED will then prompt for the next command.

To display the last line of the edit buffer, enter $.

To print any line, enter the line number followed by a 'P'. Thus

```
1P
```

produces the response

```
now is the time
```

which is the first line of the buffer.

It is common to use '$' in combinations such as

```
$-1,$P
```

to print the last two lines of the buffer.

-+*+-

3.41.7 THE CURRENT LINE QED: dot

Suppose the buffer still contains the three lines as above; that the operator has just typed

```
1,2P
```

and QED has displayed lines 1 and 2. Entering

```
P       (no line numbers)
```

will cause QED to return

```
for all good men
```

Although this is the second line of the buffer, it is in fact the line most recently processed (ie. 1,2p the last line printed). If the 'p' command is repeated without line numbers, QED will continue to display line 2.

QED maintains a record of the last line referenced which can be used instead of an explicit line number. This most recent line is referred to by the shorthand symbol

```
.       (pronounced 'dot').
```

Dot is a line number, meaning more exactly, 'the current line', or 'the line most recently processed.' It can be used in several ways - one example is

```
.,$P
```

This will print all the lines from (including) the current line to the end of the buffer. In this case, lines 2 and 3.

Some commands change the value of dot, while others do not. The print command sets dot to the number of the last line printed. For example, after the command ".,$P" (as above), "dot" will be set to 3 (the last line in the buffer).

Dot is implied when used in combinations like:

```
+1p
```

this means 'print the next line' and is a handy way to step through a buffer. Also

```
-1p
```

which means 'print the line before the current line.' This enables backward referencing. Another example of the relative line number addressing (dot implied) is:

```
-3,-1p
```

which prints the previous three lines from dot.

Remember that Print commands change the value of dot itself. To find the value of 'dot' enter

```
. (or just null XMIT)
```

QED will respond by displaying the dot line and/or current value of dot, ie. latest line number.

Review of the 'P' command and dot.

Essentially 'P' can be preceeded by 0, 1, or 2 line numbers. If there is no line number given QED displays the current line. If there is one line number given QED displays that line and sets dot there. If two line numbers are given QED displays all lines in the range and sets dot to the last line displayed. If two line numbers are specified, where the first is greater than the second (see exercise 2), QED will logically invert the arguments before execution of the command.

Pressing XMIT twice will cause printing of the next line - This is equivalent to entering +1P.

-+*+-

3.41.8 DELETING LINES **QED: d**

Suppose the edit buffer contains

```
now is the time
for all good men
to come to the aid of their party.
now is the time
for all good men
to come to the aid of their party.
```

To remove the duplicate lines in the example use

```
starting line, ending line D
```

Either command

```
4,6D or 4,$D
```

would delete lines 4 through the end. Try this example and verify the change by displaying the contents of the buffer using

```
1,$P
```

Notice that $ equals 3. Dot is set to the next line after the last line deleted unless the last line deleted is $, in which case dot equals $.

-+*+-

3.41.9 MODIFYING TEXT; THE SUBSTITUTE COMMAND QED: s

The substitute command 'S' is used to change text within a line or set of lines. Suppose by a typing error line 1 reads:

```
now is th time
```

The command

```
1S/th/the/
```

would set dot to line 1 and substitute for the characters 'th' the characters 'the'. To verify the change, issue P and see the results. This yields

```
now is the time
```

Note that dot is set to the line where the substitution took place, since the P command displayed that line.

In general, the format of the substitute command is

```
starting-line, ending-line s/change this/to this/
```

Whatever string of characters is between the 1st and 2nd delimiter is replaced by whatever is between the 2nd and 3rd; everywhere it occurs; in all the lines between the starting line and the ending line. The rules for line numbers are the same as those for 'P', except that dot is set to the last line changed.

Thus

```
1,$S/speling/spelling/
```

will correct a 'speling' error everywhere in the text. If no line numbers are given, the 'S' command defaults the substitution to line dot(ie. it changes text only on the current line). This leads to the very common sequence

```
S/----/----/P
```

which makes a change to the current line, and prints it to verify the change. Note that this is not the same as

```
SP/----/----/
```

which only prints a line if the substitution occured. The 'P' in this case is a optional modifier to the 'S' command itself. (Notice the use of multiple commands on a line in the 1st example. This is often possible; substitute then print is the most common case.)

Also of interest is

S/----//

which will 'change every occurrence of the first string of characters to nothing' (ie. remove/null them). This is useful for deleting extra words in a line or removing extra letters from words.

-+*+-

3.41.10 CONTEXT SEARCHING **QED**

Suppose the buffer contains

```
now is the time
for all good men
to come to the aid of their party.
```

Problem: Find the line that contains 'their' to change it to 'the'. With only three lines in the buffer, it's easy to keep track of what line the word 'their' is on. If the buffer contained several hundred lines and many additions and deletions had been made, it would be difficult to establish line numbers. Context searching is simply a method of specifying the desired line, without knowing its number. This is done by specifying data within a line which fixes its location uniquely (hence context).

To search for a line that contains the particular string of characters enter

```
/character string to locate/
```

For example, the QED line

```
/their/P
```

is a context search sufficient to find the desired line - it will locate the next occurrence of the characters between /'s ('their'). It also sets dot to that line and prints the line for verification:

```
to come to the aid of their party.
```

'Next occurrence' means that QED starts looking for the string at line DOT+1, searches to the end of the buffer, then continues at line 1 and searches to line dot. (That is, the search 'wraps around' from $ to 1.) It scans all the lines in the buffer until it either finds the desired line, or returns to dot again. If the given string of characters cannot be found in any line, QED responds with an error message, Otherwise it prints the line it found.

Both the search for the desired line and a substitution can occur in the same command sequence as follows

```
/ aid /sp/their/the/
```

which will yield

to come to the aid of the party.

There were three parts to that last command: context search for the desired line, make the substitution, if so print the line.

The expression / aid / is a context search expression. In their simplest form, all context search expressions are a string of characters surrounded by delimiters. Although slash is used as the delimiter in these examples, any other special character which does not have significance to QED may be used (ie: apostrophe (') or back slash (\) for example). Context searches are interchangeable with line numbers, and may be used to find and print a desired line, or as line numbers for some other command, like 'S'. Both uses were shown in the examples above.

Suppose that the buffer contains

```
now is the time
for all good men
to come to the aid of their party.
```

Then the QED line numbers

```
/ for /
/ good /
/ all /
```

are all context search expressions, and they all refer to the same line (ie. line 2). To make a change in line 2, for example

```
/ for /S/good/bad/
```

or

```
/good/S/good/bad/
```

or

```
/ all /S/good/bad/
```

Would all achieve the same result. The following example would print all three lines

```
/now/,/party/P
```

Of course, if there were only three lines in the buffer,

```
1,$P
```

would be acceptable but not if there were several hundred.

If a context search is preceded by a minus sign (-) then the editor will search from the current line towards line 1 and then wrap around to the last line, back to the current line. The search stops when a match is found or the starting line is reached again.

If a context search is preceded by an exclamation mark (!) then the editor will search for a line which does not contain the given pattern. (- and ! may be used together).

Note: the search always begins from the current line.

The basic rule is: a context search expression is the same as a line number, so it may be used whenever a line number is required. Remember that once the line numbers are resolved, QED will test for start number less than end number and invert them if necessary.

Example:

```
48,/fox/sp/a/b/
-/quick/,+1sp/x/y/
```

-+*+-

3.41.11 REPEATED SEARCHING FOR THE SAME STRING QED

QED provides a shorthand method for repeating a context search for a previously specified string. Example, the QED line number

```
/string/
```

will find the next occurrence of 'string'. It often happens that this is not the desired line and the search must be repeated. (ie. there may be other occurrences of 'string' in the element so look around with -2,.P etc.). This can be done by entering:

```
//
```

This 'shorthand' argument represents 'the most recently used context search expression.' It can also be used as the first string of the substitute command, as in

```
/string1/S//string2/
```

which will find the next occurrence of 'string1' and replace it with 'string2'.

The substitute command may have modifiers before the 1st delimiter. If 'S' is followed by a number (say 2) then the 2nd occurrence of the string on a line is substituted. If 'S' is followed immediately by 'p' then those lines matched will be printed. If 'S' is followed by 'd' then those lines matched will be deleted. If 'string2' is not supplied (ie. just carriage return), then no substitution is done but only lines matched will be printed or deleted as requested. If the 's' is preceded by ! (exclamation mark) which means 'logical not', then all lines which do not contain 'string1' will be matched; hence printed or deleted according to the modifier.

-+*+-

3.41.12 CHANGE AND INSERT QED: c

This section discusses the change command

C

which is used to change or replace a group of one or more lines, and the insert command

I

which is used for inserting a group of one or more lines.

'Change', written as

C

is used to modify or replace a number of lines with different lines, which are entered via the terminal. For example, to change lines 2 through 4

```
2,4C
lines are displayed
2,4DI
.....enter new lines here....
"F
```

The lines entered between the 'DI' commands and the "F will take the place of the original lines between start line and end line. This is most useful in replacing a line or several lines which have errors in them. It should be noted that the number of text lines between 'sss,eeeDI' and '"F' can be varied. This means that a 'C' function can encompass multi-line addition or deletion during the change operation.

QED will display the lines to be changed. For convenience, a tab stop will be set after the generated "F. On a CRT they may be modified using the hardware editing features of the CRT and added/inserted into the edit buffer with one XMIT.

If only one line is specified in the 'C' command, then just that line is replaced. One line may be changed to many, or many to one, but QED imposes a maximum limit of 1 screen page for each change command. Notice the use of "F in the add step of the 'C' and that it must start on a new line. If "F is omitted, the user will be left in append mode and may continue to enter further text. If no line number is given, line dot is assumed. The value of dot is set to the last line added.

'Insert' is similar to ADD - for instance

```
/string/I
...enter lines to be inserted here...
"F
```

Will insert the given text before the next line that contains 'string'. The text between I and "F Is inserted before the specified line. If no line number is specified dot is used. Dot will be set to the last line inserted.

-+*+-

3.41.13 MOVING BLOCKS OF TEXT; MOVE **QED: m**

The 'M' (move) command enables rearranging sections of code or text. Position 'dot' to the line after which the text is to be added and specify which lines are to be moved. For example

```
10p

110,140m
```

copies lines 110 through 140 after line 10 and deletes them from their previous position.

-+*+-

3.41.14 COPYING BLOCKS OF TEXT; COPY **QED: k**

The 'K' (copy) command enables duplicating sections of text. Position 'dot' to the line after which the text is to be added and specify which lines are to be copied. For example

```
10p

110,140k
```

copies lines 110 through 140 after line 10.

-+*+-

3.41.15 GLOBAL COMMANDS QED: g

This section discusses QED's 'global' command,

G\string\

The global command provides a way to perform one or more editing operations on all lines in the buffer that match some specified context search.

For example to print all lines that contain the word 'comment'. The context search

/comment/

matches a line containing the word 'comment', and the global command 'G' is used, together with the print command 'p', as follows

G/comment/P

This says 'for each line that matches the context search (for the word 'comment'), execute all of the commands on this line' - of which in this example, there is only one, a print. (A similar case would be to delete all the lines containing a particular string.)

The substitute command can operate on many lines at once; consider:

1,$SP/xxx/yyy/

This scans an entire buffer but operates line by line for changes. This may not be desirable. But consider

G/zzz/S/xxx/yyy/P

which will perform the same substitution only on /zzz/ context lines, and print all of the matched lines whether or not a change occured in those lines.

This example used two commands on one line; in general, as many commands as common sense permits may be used.

The 'G' command may be preceded by two line numbers, in which case only the lines within this range are considered. As implied above, if no line numbers are given, the range '1,$' is assumed. In the following, no context criteria has been specified to limit the 'S' commands.

```
G//S/xxx/yyy/S/zzz/www/
```

The preceding 'S' commands, given as separate commands would incur twice the buffer scanning overhead to the computer. If the global command is followed by a null string then it will match all lines. Note that this is different from the usual, ie. G// does not match the most recently used context expression.

-+*+-

3.41.16 RE-DIRECTED QED INPUT QED: "<

Another way to get text into the buffer is to go into add mode (A command) and select the input from a file in the permanent library file system. This is a simple method by which tabulation may be imposed onto elements not previously structured by QED. The command:

```
"< bktext/elt
```

will read the element 'elt' from file 'bktext'. If the element to be read is a proc then ',P' must follow the element name. For example

```
"< bktest/pname,P
```

will read the proc 'pname' from file 'bktest'.

While in Add mode, the "< command may be used to select input from a file/elt. In command mode, the data read from file/elt will be interpreted as QED commands.

In this way a set of commands may be coded once and passed against many elements to minimize repetitive editing. The file processed by the re-direction command is treated as if it were an extension of the keyboard.

The file/elt selected as re-direction input may also contain a "< command but that is the limit for nesting.

-+*+-

3.41.17 READING TEXT FROM A FILE QED: r

The Read command normally appends (at the end of the buffer) data from file/elt to the lines already in the edit buffer.

For example:

```
R SOURCE/STORY
```

will read the element 'STORY' from file 'SOURCE'. If the element to be read is a proc then ',P' must follow the element name. For example

```
R FILEX/MACRO,P
```

will read the proc 'MACRO' from file 'FILEX'.

Remember, the Read command will add the text from file/elt to the end of the edit buffer. The user may then move it around as necessary. A more sophisticated version of the Read command is:

```
line R bgn,end file/elt,type
```

This allows specifing a line range (bgn,end) of text from file/elt to be appended (at 'line') into the current edit buffer; pushing existing data down. For example:

```
100 R 50,103 COPY/MODULE3
```

would copy lines 50 through 103 of element 'MODULE3' in file 'COPY' to the edit buffer after line 100 pushing 101,$ down.

The user may create an edit buffer containing the directory of a library by specifying a type of "D" or "F". A type "D" will include the comments and timestamp for each element; a type "F" will omit comments and timestamps. Each line of the edit buffer contains one element name.

-+*+-

3.41.18 WRITING AN EDIT BUFFER TO A FILE/ELEMENT QED: w

To write out the contents of the edit buffer to a permanent library file use the write command 'W'

```
W file/elt,type comments
```

This will copy the entire content of the edit buffer to the specified file. To save the text as an element named 'PROG' to a file named 'PRGFIL', for example, enter

```
W PRGFIL/PROG
```

If the element is to be written out as a proc, then ',P' must be added after the element name.

If you specify the file, element and type then you may also specify up to 20 bytes of comments. These comments are placed in the library header record for the element. The editor will not drop these comments on later editing sessions, therefore you need only supply them once. Likewise the editor will also recover the type of module from the library header record.

QED will respond with the number of lines copied after the 'write' is complete and prompt the user for the next command. It is possible to limit the number of lines written out by providing a line range. For example:

```
100,159 W COPY/MODULE1
```

would copy lines 100 through 159 of the edit buffer out to an element named 'MODULE1' in the file 'COPY'. Writing an element to a file does not delete or disturb the edit buffer. This remains intact until a 'Quit' command is issued to end the update session and scratch the work space. This is an important point ! QED at all times works on a 'copy' of an element, in a fast Edit File. No change in the contents of a library takes place until a 'W' (write) command is issued.

If you attempt to write out an element which already exists you will be prompted for over-write. Reply 'Y' or 'N'. To avoid this over-write check message, use the 2 letter command 'WR' (not generally recommended).

If you wish to write the element out, only if changes have been made, you may use the 2 letter command WC. If nothing was changed in the module the write is not done. This is useful when making up execute files of QED commands.

WRITING AN EDIT BUFFER TO A FILE/ELEMENT

Since QED at initialization, saves the file/elt,type parameters, it is possible to simply issue 'R' or 'W' commands without re-stating 'file/elt,type'.

NOTE: Whenever another or subsequent write, or input re-direction occurs, QED resets the default file/elt parameters. Under these conditions, the user should always check the settings with command '=' before executing a default I/O command.

Failure to do so may produce results which are quite undesirable.

-+*+-

3.41.19 END OF EDIT SESSION: QUIT / END QED: q, e

Prior to QED termination, the user may save his updated text by writing it to a permanent file using the 'W' command. To terminate an QED run, enter

Q

or

E

'Q' scratches the buffer while 'E' does not. Each returns control to the program which called the editor or TIP/30.

To leave QED but retain the edit buffer for later use employ 'E'nd (see 'getting started' for buffer naming).

If the user has modified any text within the buffer, QED will issue a warning message before deleting the work space. This is the last chance to save the session. If the user has written the text to a permanent file, the warning is not given.

-+*+-

3.41.20 VERSION NUMBERS QED: v

A version number for each line in the element is maintained by QED to aid the user in keeping track of updates. This number indicates when the line was last changed. Each time the element is read into the edit buffer, the version number is incremented by 1. For COBOL, Assembler, and text files the version number of each line is stored in columns 72 to 74 The user may change the current version number with the 'V' command. eg.

```
V4
```

will set the current version number to 4.

The version number is stored by QED modulo 256; that is, the remainder when the version number is divided by 256 is the new version number. (ie: v300 would result in version 44).

The version number and the name of the last person to update an element is stored in the comment area of the element's header record.

-+*+-

3.41.21 SUPPLEMENTARY QED REFERENCE QED

This section describes some of the more advanced features of the text editor.

The use of regular expressions is controlled by the options OI (option in) and OO (option out). In QED OO is the default state and the use of regular expressions is turned off. To turn 'on' regular expressions issue an OI command. The OI command remains active until the next OO command.

Regular expressions allow the user of QED to perform more complex editing, but introduce a degree of complication. For example, most special characters can no longer be used with complete freedom in substitute and context searches. Normally, to search for a string of characters, it is sufficient to type

/string of characters to be found/

The expression between delimiters is referred to as a context search expression which is, in fact, the simplest case of a regular expression. The /'s are not part of the regular expression, although most regular expressions are written between slashes.

By definition, a regular expression specifies a set of one or more strings of characters which satisfy a given context search; it is a complex form of context search.

The regular expression is a mask which provides degrees of acceptable search argument. It might specify any one of a whole set of strings of characters that will satisfy the search, or a particular string in a particular position on a line. A particular regular expression 'matches' a string of characters whenever the string contains one of the desired character strings satisfying the match pattern.

Regular expressions are typically formed from ordinary context searches elaborated by using special characters with specific functional meanings. The special characters can be interpreted as search mask operators. In fact, the discussion of regular expressions is largely a discussion of the special characters:

↑ . * $ " # % &

In OO mode ('option special characters out'), the regular expression meanings are off; use OI ('option in') to activate them.

The (↑) is a circumflex or roof-top character on Uniscopes. This character is typed as an up-arrow or cent sign on some terminals.

-+*+-

3.41.22 MATCHING AT THE BEGINNING OF A LINE QED: ^

It is often useful to be able to look for a line that begins with a specific string of characters. The regular expression

/↑string/

will find the next occurence of a line that begins with 'string'. This is a restricted context search since it only finds 'string' if it is at the beginning of a line. Thus, for example, if the buffer contains:

```
he said,
'now is the time
for all good men
to come to
the party.'
```

and dot is set at line 1, then the regular expression

/↑the/p

will display

```
the party.'
```

The scan ignores 'the' in line 2 because it is not at the beginning of the line.

A substitution for a string at the beginning of a line is a frequently used QED command. For example, as above, suppose the line dot contains:

```
the party.'
```

Then the commands

s/↑/the aid of /p

will yield

```
the aid of the party.'
```

The substitution takes place at the beginning of the line, since ↑ means in effect 'the beginning of the line' (notice the space after 'of'). It is usually easier to type ↑ than to type sufficient context data to uniquely identify the beginning of the line: For

example,

s/the/the aid of the/p

could be used but this is clumsy and long.

To modify 'the' at the front of this line for example

s/↑the/the immediate/p

would yield

the immediate aid of the party.'

The use of ↑ makes definite which occurrence of 'the' to change; without it, the result would be:

the immediate aid of the immediate party.'

This was not our intent.

Notice the use of regular expressions in two places, as a line number (for a context search) and as the text to be replaced in a substitute command. The text which is to be replaced in the substitute command is technically a regular expression; therefore all of the regular expression features may be used there.

-+*+-

3.41.23 MATCHING AT THE END OF A LINE QED: $

Another regular expression special character is the dollar sign ($). The dollar sign in a regular expression means 'the end of the line'. For example, the regular expression

```
/string$/
```

will find the next line that ends with 'string'. Do not confuse this with '$' used as the last line of the buffer; the OI mode has altered the normal meaning.

Again, '$' like circumflex, is probably most useful as part of a substitute command, where it can be used to add characters to the end of a line. For example, suppose the buffer contains:

```
the other side of the coin
```

Then the commands

```
s/$/ is a tail./p
```

produce

```
the other side of the coin is a tail.
```

Note the blank before 'is'. Leaving it out yields:

```
the other side of the coin is a tail.
```

If the line in the buffer is

```
to come to
```

,then to change the 2nd 'to'

```
s/to$/immediately to/
```

gives,

```
to come immediately to
```

The '$' definites which word 'to' is referenced.

As an illustration of the both uses of '$', the command

```
/strings$/$p
```

will print all lines from the next one ending in 'string' to the end of the buffer.

-+*+-

3.41.24 MATCHING ANY LETTER **QED: %**

The percent sign (%) will match any letter of the alphabet when used in a regular expression. For example, *SP/ a% /

would print the lines containing words 'as','at','an'...etc. not, however, 'a ','al','a2'...etc..

-+*+-

3.41.25 MATCHING ANY DIGIT **QED: #**

The number sign (#) will match any digit when used in a regular expression. For example, *SP/ a# /

would print the lines containing 'a0','a3','a9'...etc. not, however, 'a ','as','at'...etc..

-+*+-

3.41.26 DISPLAYING A COLUMN SCALE QED: O#

When used as a command, # will cause a scale of numbers to be placed on the terminal. This is useful for aligning column dependent data in control cards or RPG source.

-+*+-

3.41.27 SAVE THE CURRENT LINE NUMBER **QED: >n**

In the > command, the modifier 'n' may be a digit from 0 to 7. You may save 8 line numbers for use later. Generally this is used in implementations of re-direction procedures, where searches are employed to bracket block moves/copies of text.

-+*+-

3.41.28 RECALL SAVED LINE NUMBER QED: <n

n may be a digit from 0 to 7. The previously saved line number is recalled and used within the current command. This function is not technically a command in its own right since it is used as line number equivalent within other commands. For example, the sequence:

```
<1,<3S/old/NEW/
```

sets the 'start,end' line range for the 'S' command to content of the save registers 1 and 3 respectively.

-+*+-

3.41.29 OI MODE REPETITION **QED: ***

The asterisk * is used to indicate an arbitrary number of repetitions (including zero) of some string. As an example,

/ab*c/

means 'search for any one of the following:'

ac, abc, abbc, abbbc, ...etc

Notice that 'b*' includes a string of no characters; also, * applies only to the previous character - just the 'b' is repeated.

More useful would be the expression

/↑ *$/

which searches for a line that contains only blanks. Notice that a blank has been typed before the '*'.

'*' is most useful when used in conjunction with other special characters, particularly the period. Examples of the use of '*' with other special characters will follow.

-+*+-

3.41.30 MATCHING ANY CHARACTER QED: .

The period '.' is another character that QED uses in more than one way, with different meanings. Its use as 'dot', the current line has been discussed. This section describes its use as a 'match anything' character in regular expressions.

The precise definition of '.' in a regular expression is that it matches any single character. Thus

```
/x.y/
```

would match any one of the lines

```
x=y+1
8 = x+y
if (x<y) go to 10
a b c ... x y z
```

And of course it will match

```
x.y
```

combining the period with ↑ gives the expression

```
/↑...the/
```

which matches any line that starts with 3 characters followed by 'the'. This would include any of

```
on the other side of the coin
to the party
another time
```

but not

```
the other side
```

Probably the single most important use of '.' is in combination with '*' ; for instance

```
.*
```

means 'any string of zero or more characters on a line'. It is usually used to save typing a long string of characters; only a small part is typed, and the rest is expressed by '.*'. So, for example, if the buffer contains

to come to the aid of the party

then the commands

s/aid.*/party/p

will produce

to come to the party

The '.*' in this case matched all of the line after 'aid'.

Equally effective is

s/aid.*the //p

to get again

to come to the party

As a final example, the expression

/↑begin.*end$/

matches any line that starts with 'begin' and ends with 'end'.

-+*+-

3.41.31 WHAT WAS JUST MATCHED **QED: &**

The ampersand '&' is another shorthand symbol, which often saves typing. Suppose that the current line is

```
now is the time
```

and that parenthesis are required around it. One way would be to make the substitutions

```
s/↑/(/

s/$/)/
```

Another way is based on the ampersand; the following command has exactly the same effect.

```
s/.*/(&)/
```

This example defines '&' as a shorthand symbol for 'the text matched by the regular expression in the substitute command'. Whatever was to be replaced (ie. whatever was matched) is available by typing '&' in the replacement text. Consider this substitution on the original line:

```
s/.*/'&'? He answered, '&'./p
```

which returns

```
'now is the time'? He answered, 'Now is the time'.
```

The regular expression '.*' matched the whole line, so that is the 'value' of '&'.

It is not necessary to match the whole line. Suppose the buffer contains

```
the end of the world
```

A common abbreviated command sequence would be

```
/world/s//& is at hand/p
```

to produce

```
the end of the world is at hand
```

Observe this example carefully, for it illustrates QED's conciseness. The regular expression '/world/' found the desired line; the shorthand '//' found the same word in the line; and '&' saved retyping the text.

The '&' has special meaning only within the replacement text of a substitute command. To use ampersand within the replacement text, use two ampersands in a row. For example

```
s/ampersand/&&/
```

will convert the word 'ampersand' to the real symbol '&' in the current line. substitute command to separate out various parts of the string matched by the regular expression, for reference in the second half of the substitute command. This is an idea analogous to the '&', which represents the entire matched string.

Suppose that each line in the buffer contains a 5-character sequence number as its last five characters. Suppose want to move this information to the beginning of the line. This can all be accomplished in just one command.

```
*s/↑{.*}a{.....}b$/ba/
```

Examine this carefully! The first character after the closing brace is the label or tag for that portion of the string matched within the braces. These tags only have this special meaning within that substitute command and may be used any number of times.

If you do not have a full keyboard then the square brackets ("[" and "]") may be used by setting QED in upper case mode. (The upper case command is OU).

-+*+-

3.41.32 REGULAR EXPRESSION CONSIDERATIONS QED

If a regular expression can match several overlapping strings on a line, it will first match the leftmost (making it as long as possible) and will then find the next non-overlapping and longest string, until the entire expression is satisfied. Remember that regular expression will not match text spread over two or more lines.

-+*+-

3.41.33 SUMMARY OF COMMANDS AND LINE NUMBERS QED

The general form of QED commands is the command name, preceded by one or two line numbers, and perhaps followed by arguments. Commands may follow one another directly on the same line. Exceptions are 'that no commands may follow R or W or redirection commands' and 'Global executes all commands on its own line'.

-+*+-

COMMAND AND FUNCTION SUMMARY

3.41.34 COMMAND and FUNCTION SUMMARY **QED: summary**

A ADD	Add lines after the specified line number (else dot) Adding continues until "F is seen as first two characters on a line. Dot is set to last line added.
C Change	Change the lines specified to the new lines following the c command, up to "F. If no lines are specified, replace line at dot. Dot is set to last line changed.
D Delete	Delete the lines specified. If none specified, delete line dot. Dot is set to the next undeleted line.
E End	End the Edit session. The edit work file is saved.
= Facts	Give the file name of the last R or W command. Note that a W or R command immediately followed by a carriage return will use the file name which is displayed by the = command.
G Global	G/context search/ QED commands..... Execute the QED commands on all lines that satisfy the context search.
I Insert	Insert lines before the specified line (or dot) until a "F is typed on a new line. Dot is set to the last line inserted.
K Kopy	Copy the specified lines after dot. Dot will point to the last line copied in.
M move	Move the specified lines after dot. Dot will point to the last line moved in.
#	Display a number scale across the screen.
_	Display the next line.
OO Option Out	Turns OFF the use of regular expressions, and special character meanings in substitutes, context searches and tabbing. This is the default QED mode.

OI Option In Turns ON the use of regular expressions, the inverse of OO.

OA Option ASM Causes all lines added and written out to be tabbed by assembler conventions. A blank is used to separate the fields and a tabset character will tab to column 72 for continuation.

Normal tabbing is to columns 1,10,20,39,72 and continued lines begin in column 16. Use OO to turn tabbing off.

OT Option ASM Causes all lines added and written out to be tabbed by assembler conventions. A blank is used to separate the fields and a tabset character will tab to column 72 for continuation. Normal tabbing is to columns 1,10,16,40,72 and continued lines begin in column 16. Use OO to turn tabbing off.

OD Option ESCAPE Turns OFF the system escape feature of TIP. Use this if input lines may begin with the same character as the system escape character (usually @).

This command is a toggle; it flips the switch (ie. 1st on, 2nd off, etc.)

O# numbering Turns line numbering on or off. Every line displayed to the terminal will be preceded by its line number. To turn this off enter O# a second time.

OS scroll This command changes the way in which terminal output is handled. Scrolling or just carriage returns.

OX text QED will delete redundant blanks and permit word overflow from one line to the next.

OR RPG Sets RPG tabulation mode where columns 1-5 are not displayed. When the element is written out columns 1-5 are sequenced and the element name is placed in columns 75-80 as required by the RPG compiler.

OC COBOL Sets COBOL mode. Like "OR", only columns 1-6 are not displayed. Note the first character you enter on each line goes in column 7; also,in COBOL mode, tab-set characters will cause 4 blanks to be inserted for each tab-set entered beyond column 7.

COMMAND AND FUNCTION SUMMARY

OL Lower — All alphabetic characters input from the keyboard are changed to lower case.

OU Upper — All alphabetic characters input from the keyboard are changed to upper case.

ON Normal — No upper/lower case conversion of characters input from the keyboard. (What you typed is what you get).

P Print — Display specified lines on terminal; if none specified, print current line (dot).

Q Quit — Terminate the text editor program. The QED buffer (work file) is scratched

R Read — Add text from the specified file/elt to the end of the edit buffer unless otherwise specified.

"< file/elt — Used to redirect QEDs input from a element of a library file. QED will take its input (commands) from the named file/element until end of file is reached. This command must not be followed by any other QED command on the same line on the terminal.

Usually used in association with QED Exec Elements.

S Substitute — S/string1/string2/

Substitute characters 'string2' for 'string1' wherever 'string1' occurs in specified lines. If no line is specified, make substitution in line dot. Dot is set to last line in which a substitution took place. Note that the slashes can be replaced by any character which is not QED defined as significant.

A numeric modifier (S1/string1/string2/) can be used to just change the nth occurrence of 'string1' in a line.

V Version — Set the version number.

W Write — Write out buffer to a permanent file. A line range may be specified to limit the transfer. In any case, dot is changed to the last line written out.

ZL Zap Lower Change alphabetics in the specified lines to lower case. The current line (dot) will be set to the last line where a change of case was made.

ZU Zap Upper Identical to ZL command except that the change is to upper case.

ZSnn sort Sort the specified lines into ascending sequence.

ZBnn sort Sort the specified lines into descending sequence.

If ZS or ZB is followed immediately by a number (nn) the sort will be done from that column of each record to the end of the record.

= Display summary of edit buffer contents and options in effect

/----/ Context search. Search for next line which contains this string of characters and print it. Dot is set to line where the string is found.

Search starts at DOT+1, wraps around from $ to 1, and continues, if necessary, back to the starting point.

-/----/ Context search in reverse direction. Start search at DOT-1, scan to 1, wrap around to $.

!/----/ Context search for line that does NOT have this string on it. The ! may be used in conjunction with the reverse search direction '-' command.

-+*+-

3.41.35 LINE NUMBERS **QED**

.	Current line ("dot") - set by many commands, often to last line changed or referenced.
$	Last line in the edit buffer.
1,2,...	Absolute line numbers in edit buffer.
/xxxx/	Implicit context search - line number of next line that contains the string of characters.
-/xxxx/	Implicit context search in reverse direction.
*	All lines in the edit buffer Equivalent to 1,$
&	Special line range for CRT terminals. It is approximately equivalent to '.,+15' (16 lines). Approximately equivalent because it may adjust based on the specific terminal type in use.

-+*+-

EXERCISE 1: APPEND, QUIT, WRITE

3.41.36 EXERCISE 1: APPEND, QUIT, WRITE QED: Exercise 1

Enter QED and create some text using

```
   a
...text...
"f
```

Write it out using W. Then leave QED with the Q command. To check the results call QED with the file/elt name used to write the text out. When QED has read the elt in display the buffer contents using P.

-+*+-

3.41.37 EXERCISE 2: APPEND, PRINT **QED: Exercise 2**

As before, create some text using the append command and experiment with the P command. You will find, for example, that you can't print line 0 or a line beyond the end of the buffer and that attempts to print a buffer in reverse order by saying

```
3,1p
```

will result in QED inverting the line arguments before execution of the command.

-+*+-

3.41.38 EXERCISE 3: READ, PRINT, APPEND **QED: Exercise 3**

Experiment with the R command - try reading and printing various files. Try alternately reading and appending to see that they work similarly.

-+*+-

3.41.39 EXERCISE 4: ADD, READ, PRINT, WRITE QED: Exercise 4

Experiment with A, R, W, P, and D. Understand how dot, $, and the line numbers are used.

Try using line numbers with A, R, and W as well. Note that 'A' will append lines after the line number that you specify (not necessarily at the end); and that 'W' will write out exactly the lines specified, not necessarily the whole buffer. These variations are sometimes handy. For instance to insert an element at the beginning of a buffer use

```
      1I
"< filename/eltname
```

and to insert lines at the beginning of the buffer use

```
      1I
....text...
"F
```

-+*+-

3.41.40 EXERCISE 5: SUBSTITUTE **QED: Exercise 5**

Experiment with the substitute command. See what happens if you substitute for some word on a line with several occurrences of that word. For example, do this:

```
A
the other side of the coin
"F
S/the/on the/P
```

you will get

```
on the oon ther side of on the coin
```

Try it! Be sure you understand what's happening - that substitute changes all occurrences of the first string. Even experienced users make mistakes by forgetting this.

Try other characters instead of /'s to delimit the two sets of characters in the S command. Try several S commands (or others) all on one line.

If you recreate that same line again, but this time try

```
S1:the:on the:P
```

you will get, instead

```
on the other side of the coin
```

Notice the 1 immediately after the S. (the : were used for / just to show it could be done.) The 1 caused QED to only carry out the first substitute rather than all of them. In general, any number can be used instead of the one. It allows you to select a particular substitution to take place rather that every one on the line.

-+*+-

EXERCISE 6: CONTEXT SEARCHING

3.41.41 EXERCISE 6: CONTEXT SEARCHING **QED: Exercise 6**

Experiment with context searching. Try a body of text with several occurrences of the same string of characters, and scan through it using the same context search. (see section 1.9).

Try using context searches as line numbers for the substitute, print and delete commands. (They can also be used with R, W, and A.)

Try context searching using -/text/ instead of /text/. This scans lines in the buffer in reverse order rather than normal: sometimes useful if you go too far while looking for some string of characters. It's a fast way to back up.

-+*+-

3.41.42 EXERCISE 7: CHANGE **QED: Exercise 7**

"Change" is exactly the same as a combination of delete followed by insert. Experiment to verify that

```
start,end D
I
....text....
"F
```

is the same as

```
start,end C
....text....
"F
```

Experiment with A and I, to see that they are similar, but not the same. You will observe that

```
line-number A
....text....
"F
```

adds after the given line, while

```
line-number I
....text....
"F
```

inserts before it. Observe that if no line number is given, 'I' inserts before line dot, while 'A' appends after line dot.

-+*+-

3.42 RELOAD PROGRAM RELOAD

The first time a program is loaded from the TIPLOD library TIP/30 will move a copy of the load module to the TIP$SWAP file. Any subsequent requests for the load module will cause TIP to read the load module from the swap file copy. If the programmer has compiled the program, he may wish TIP/30 to get the new version from the TIPLOD library. To tell TIP/30 to do this the programmer must use the RELOAD transaction.

Syntax:

RELOAD loadm

Where:

loadm is the load module name.

Example:

RELOAD PAYUPD

Would display (for example):
PAYUPD cleared from loadr table.
PAYUPD cleared from reentrant control table.
Using PAYUPD as of 82/05/19 15:05:35 (C) ALLINSON-ROSS

Additional Considerations:

If the program is being used re-entrantly then TIP/30 must wait for all current users of the program to stop using it before a new version can be loaded.

RELOAD will have no effect on resident programs. A new version of a resident program can only take effect at TIP/30 initialization.

3.43 RPG EDITOR RPG

The RPG editor is an online program which was written to aid programmers in the creation and maintenance of programs written in the language 'RPG'. Using RPG, a programmer no longer has to worry about aligning fields in their columns. RPG has eight screen formats; one for each of the form types used in writing 'RPG' programs. The User only has to select the appropriate screen and enter the data on titled blank fields. RPG edits and aligns the data as if it were on a card.

In total there are 10 screen formats used in the RPG editor:

- Menu
- Record list
- Control card format
- File descriptor format
- File extension format
- Line counter format
- Telecommunications format
- Input format
- Output format
- Calculation format

where:

Menu — All commands are issued from the menu. All the remaining screen formats can be displayed using commands from the basic menu

Record list — Displays from one to fifteen records from the input file in card format

Others — The remaining screen formats are of the eight format types which correspond to the syntax of the 'RPG' language. These are invoked by entering the form type as the command in the menu; the corresponding screen format will be displayed. Each screen format contains the field names and data areas for the

corresponding form.

3.43.1 ENTERING RPG

After logging on to TIP, activate RPG by entering

```
RPG
```

to create a new element or

```
RPG   FILE/ELT
```

to update an existing element.

In the event of a system crash the edit buffer can be retrieved by entering

```
RPG   ELT
```

3.44 ERROR MESSAGES

RPG will respond to command and data entry errors by displaying a self-explanatory error message. In the case of data entry errors, in addition to the message the fields in error are set to blink.

3.44.1 DELETE

To delete a record from the edit buffer it must be displayed on its correct 'RPG' format screen. Once the record is displayed, it can be deleted by pressing function key 2. The line number of all records following the deleted record are decreased by one. The current line is the record which immediately followed the deleted record.

3.44.2 ADD A RECORD

The addition of records is also done from the formatted screen. If the screen displayed at the moment is not the correct 'RPG' format, the user must intercede by returning to the menu and selecting the correct form type format. Once the data has been entered, press transmit. The data is then validated. If all fields are valid, the record is added to the edit buffer and becomes the current line. In the case where the current line already pointed to a record, that record and any following records would have their line numbers increased by one.

3.44.3 UPDATE RECORDS

To update a record, display it on the CRT with its correct form type screen; make the necessary corrections to the data and press function key 1. The data fields are then edited. If they are all valid the old record is replaced by the new record and the user is given update confirmation. If the RPG validation fails, the fields in error are changed to blinking fields and the record is not updated. The user may correct the fields in error and re-submit or request another screen. Any fields which are numeric or blank only are edited by the screen formatter. Any record in the edit buffer can be updated as long as it is in one of the eight format displays. There are five ways to get a record into these displays:

- from the menu by entering '.' (current line)
- from the menu by entering '$' (the last line)
- from the menu by entering 'P' and the line number (specific line)
- from one of the eight format displays by pressing f3 (next record)
- from one of the eight format displays by pressing f4 (previous record)

The current line number is not altered by updates.

3.44.4 LIST LINES

To list part of the edit file: enter 'P' as the command on the menu and the beginning and end line numbers of the lines to be listed. The records are listed as they would appear on cards except that the line numbers and program identification are not shown. A maximum of 15 lines can be viewed at once. After these lines are listed, the next or previous 15 records may be viewed by pressing function key 1 or 2 respectively. The current line is the last line displayed on the terminal.

NOTE: If 'P' is entered without a line number, RPG editor assumes that the user wants the current line displayed in its card format.

3.44.5 GETTING OUT OF RPG

To terminate the session enter 'X' or'Q' as the command in the menu. Upon entering 'Q' the buffer is scratched whereas with 'X' it is retained. Before entering this command the user may wish to save his updated text by writing it to a permanent file using the 'W' command.

3.45 CURRENT LINE

To display the current line in its format display, enter '.' (dot) as the command on the menu.

3.45.1 LAST LINE

To display the last line in the edit file enter '$' (dollar sign) as the command on the menu. The record will be displayed in its corresponding format. The last line now becomes the current line.

3.45.2 LINE NUMBER OF CURRENT LINE

By entering '=' as the command on the menu the user is informed of the line number of the current line.

3.45.3 WRITING TEXT TO FILE

To write the contents of the edit file to a permanent file enter command 'W' and the 'filename/elt on the menu. A special comment can be inserted on the file header by entering it next to the 'comment' (maximum length is 20 characters). If this entry is left blank, your USER-ID will be used.

On a successful write to the library, the editor will respond with the number of lines copied. The edit file does not change as a result of the write command. It is important to remember that RPG works only with a copy of what is in the library file. The content of the library file does not change until the write command is issued and confirmed positively.

3.46 START OS/3 BATCH JOB RV

To start an OS/3 batch job, the TIP user may use the SYM program (see section on "SYM") and enter an OS/3 operator command via the SYM program. A more direct approach is the use of the RV program. The RV program is, in fact, a transaction-id that calls the SYM program. The SYM program detects that it has been called with a transaction-id of "RV" and reacts appropriately.

The RV program expects (on the command line) the parameters that the user would normally give to the OS/3 operator "RV" command. The user should keep in mind that OS/3 limits the length of a console command to a maximum of 60 characters.

Syntax:

RV parameters

Where:

parameters the parameters required by the RV command.

Example:

```
RV COB74(FRED),,E=TEST010
```

Would run a job stream named "COB74" from YJCS and cause the job name to be changed to "FRED". The keyword specification assigns a value to the job global "E".

Error Conditions:

The user may receive a security error if he does not have sufficient security to run the "RV" program.

3.47 SCRATCH A DYNAMIC FILE SCRATCH

The SCRATCH program is used to scratch a dynamic file that is currently assigned to the terminal. The SCRATCH program removes the entry for the dynamic file in the TIP/30 catalogue and releases the space currently used by the dynamic file in the TIP$RNDM file.

Syntax:

SCRATCH[,A] aft-name

Where:

"A" option used to indicate that all assigned files are to be scratched. Any OS/3 files assigned are FREE'd by this option. No file names need be specified.

aft-name is the active file name of the file to be scratched.

Example:

```
SCRATCH WORK1
SCRATCH,A
```

Error Conditions:
TIPFCS errors may be reported.

3.48 SET ATTRIBUTES FOR PROCESS SET

The SET program is a utility that allows the user to change various attributes of his own or other terminal processes.

Syntax:

SET [FOR term] [attributes]

Where:

FOR term	is the terminal name which is to be changed. If omitted then the calling terminal is used.
attributes	any of the following parameters:
U200	change terminal type to U200.
U400	change terminal type to U400.
Q310	change terminal type to Q310.
TTY	change terminal type to TTY (teletype).
LMON	turn TIP/30 line monitor on.
LMOFF	turn TIP/30 line monitor off.
LOGON=YES	terminal requires logon.
LOGON=NO	terminal is NOT required to logon.
DISABLE	terminal is disabled. No transactions will be allowed.
ENABLE	terminal is enabled.
DEBUG SYSTEM	all programs will run with storage protection.
DEBUG OFF	inverse of "DEBUG SYSTEM".
TEST ON	the terminal is set in test mode. File updates ignored.
TEST OFF	the terminal is cleared from test mode.

Example:

```
SET FOR T312 U200 LMOFF LOGON=YES.

SET U400 LOGON=NO.
```

Additional Considerations:

This program is intended to be used by systems programmers.

3.49 SPOOL FILE ENQUIRY SPL

The SPL program enables the user to examine sub-files in the OS/3 spool queues. A spool sub-file may be listed at the terminal, printed at a terminal printer, released for batch printing, or deleted.

The SPL program is able to read sub-files in the OS/3 spool queues. It has no provision for modification of data in the sub-file.

The OS/3 spool file is divided into two classes of sub-file:

- Held
- Not Held (queued)

Sub-files that are held are the usual (default) target of the SPL program. It is possible to direct SPL to examine sub-files that are not held, but the user should be aware that sub-files are queued only until the OS/3 output writer opens them for processing. There is, therefore, a potential race condition associated with queued files.

The OS/3 spool file is also divided (for each of the two classes described above) into the following queues:

LOG	job log.
PR	local print (default queue for SPL).
PU	local punch.
RDR	local reader.
RDR96	local 96 column reader.
SYSLOG	retained job logs.
RBPIN	remote reader (if configured).
RBPPR	remote print (if configured).
RBPPU	remote punch (if configured).

There are 18 (2 x 9) combinations of class and queue.

To examine or manipulate a spool sub-file entry, the user must always clearly establish both the class (default is HELD) and the queue (default is PR) of the desired sub-file.

SPOOL FILE ENQUIRY

The SPL program operates only in interactive mode. It makes no use whatsoever of command line parameters. To begin the SPL program simply enter the transaction name:

```
SPL
```

When SPL prompts the user for commands, the general syntax is as follows:

Syntax:

```
cmnd  [queue] [option] [keyword...keyword...keyword...]
```

Where:

cmnd A recognized SPL program command (eg: DELETE, PRINT, etc) as described in the next sections.

queue The OS/3 spool queue to be searched (default is PR).

option Optional additional information required by some commands as documented.

keyword Optional keywords that are used to qualify the selection of sub-files in the specified queue.

3.49.1 SPL SECURITY CONSIDERATIONS SPL: security

To maintain the security of the OS/3 spool file, the SPL program will display information from the spool queues according to the following rules:

* MASTER level users (ie: security 1 thru 9) are able to examine any spool queue sub-file;

* SYSTEM and PROGRAMMER users (ie: security 10 thru 29) are able to examine any spool queue entry with form name "STAND1";

* Other spool sub-files can be examined by a user if and only if:

  - userid, group one, group two, or terminal name (four characters) matches one of: FORM=, CART=, REMOTE=, FILE=, or ACCT= keyword specified.

Note that account number is the 4 character account number as given on the JOB statement of the job that created the spool sub-file.

-+*+-

3.49.2 SPL KEYWORDS — SPL: keywords

Following is a summary of the keywords that are recognized by the SPL program. Most keywords provide information that is used to specify the desired sub-file entry (ie: Jobname= etc).

Some keywords provide information to the SPL program that changes the behaviour of the SPL program (ie: PAge=).

Upper case characters in the keyword are required characters; lower case characters are noise characters for readability.

Acct= sub-files with this job account number.

The account number is a parameter on the // JOB statement and is restricted to 4 digits for SPL purposes.

Cart= sub-files with this print band name.

The cartridge name is a parameter on the // LCB statement

COlumn= specify leftmost column to display.

This keyword specifies the starting column to be used. Default is column one.

FIle= sub-files created with this LFD name.

This allows selection based on originating LFD name.

FOrm= sub-files that specify this form name.

This keyword allows selection based on orignal form name.

Hold= held or not-held class.

Indicates the class of spool queue (held or not held). Specified as "Y" or "N". Default "Y".

Job= sub-files with this job name.

JobNo= sub-files with this job number.

The summarize SPL command displays sub-file job numbers that may be referenced by this keyword.

Label= sub-files created with this label.

This keyword allows selection based on // LBL name.

Prog= sub-files created by this program name.

Selection by EXEC name.

PAge=nnn specify starting page number.

The summarize SPL command displays number of pages in the sub-file. This keyword allows user to begin processing at a specific page number.

Remote= sub-files for this remote destination.

The destination from the // DST statement.

STep= sub-files created by this step number.

The step number within a job.

USing=term,dvc route SPL output to alternate terminal.

SPL may be started up (asynchronously) on another terminal (to print using an attached printer).

If term is omitted, the issuing terminal is assumed.

If dvc is omitted, device AUX1 is assumed.

To route printout to AUX2 of your terminal, for example, USING=,AUX2 may be specified.

Several keywords may be specified to narrow the search as much as possible.

-+*+-

3.49.3 SPL PROGRAM OPERATION SPL: operation

Since the first sub-file that matches the specified criteria may not be the intended sub-file, the SPL program always prompts the user to determine if the found sub-file is to be processed.

When SPL finds the first sub-file (of the class and queue specified) that matches the criteria specified by the keyword information, it will display all known information about that sub-file. The sub-file that is found may not be the intended one - especially if the keyword information was too vague.

SPL then prompts the user for confirmation that the sub-file found is indeed the one wanted. If the user replies "Yes", the command will be carried out; if the reply is "No", the search will continue for the correct sub-file.

While a sub-file is listed at the user's terminal, the user may press MSG-WAIT to interrupt the display. The user is then prompted with a continuation prompt.

In response to the continuation prompt, the user may tab to the appropriate choice and press transmit.

The user may change page number (forward or backward) and/or may change the starting column number. To do this, specify:

```
>PAGE nnn [,ccc]
```

where nnn is the page number to proceed to and ccc is the new starting column number.

-+*+-

3.49.4 SPL FUNCTION KEY USE — SPL: fnkeys

The SPL program recognizes the following use of function keys:

MSG-WAIT Interrupt display on terminal.

User will be prompted with a continuation query.

F2 Re-display last command entered (can save some typing).

F3 Re-execute last command entered (can save some typing).

-+*+-

3.49.5 DELETE SPOOL SUB-FILE SPL: delete

This command enables the user to select spool sub-files to be deleted.

Syntax:

DELete [queue] [,ALL] [...keywords...]

Where:

queue Optional positional parameter which specifies the desired spool queue (default is PR).

ALL Optional positional parameter which indicates that ALL sub-files found that match keyword criteria are to be processed.

keywords see section 3.49.2

Example:

DEL ALL JOB=COB74

Would select ALL sub-files with jobname "COB74" in the held class for possible deletion.

Error Conditions:
None.

Additional Considerations:

The SPL program will display information about each sub-file in turn and prompt the user for delete verification.

-+*+-

3.49.6 END SPL PROGRAM SPL: end

This command will cause the SPL program to terminate normally.

Syntax:

End

Where:
No parameters required.

Error Conditions:
None.

-+*+-

3.49.7 DISPLAY SPL PROGRAM HELP SPL: help

This command will display on the terminal a summary of SPL program command syntax.

Syntax:

Help

Where:
No parameters requried.

Example:

HELP

Error Conditions:
The help information may not be available or may have been deleted.

-+*+-

3.49.8 LIST SPOOL FILE ON TERMINAL SPL: list

This command will list selected spool sub-files on the terminal. Since print lines (for example) are usually longer than the width of most terminals, the output from the list command may be "folded". This means that the display may span more than one line.

Syntax:

List [queue] [,ALL] [...keywords...]

Where:

queue Optional positional parameter which specifies the spool queue to be searched. Default is PR.

ALL Optional positional parameter which indicates that ALL sub-files found to match are to be processed.

keywords See section 3.49.2.

Example:

L JOB=COB74 PROG=LNKEDT

Would select for listing on the terminal any entry in the (held) PR queue that has a job name "COB74" and a program name "LNKEDT".

Error Conditions:
None.

-+*+-

3.49.9 LIST (SPACE SUPPRESSED) SPOOL FILE SPL: ls

This command is similar to the LIST command. Multiple spaces will be reduced to a single space, thus attempting to display more data per line on the terminal.

Syntax:

LS [queue] [,ALL] [...keywords...]

Where:

queue Optional positional parameter which specifies the spool queue to search. Default is PR.

ALL Optional positional parameter which indicates that ALL sub-files that match the selection criteria are to be processed.

keywords See section 3.49.2.

Example:

LS JOB=COB74 PAGE=10

Would list (with multiple space suppression) any entry in the (held) PR queue which has a job name "COB74". The PAGE=10 specification indicates that the listing is to start at page 10 of the file.

Error Conditions:
None.

-+*+-

3.49.10 LIST (TRUNCATED) SPOOL FILE SPL: lt

This command is similar to the LIST command. Output to the terminal will be truncated to the width of the lines on the terminal. The keyword COL= is very useful to specify the starting column number to display. By varying the starting column, the user can view either the left or right side of the spool data.

Syntax:

LT [queue] [,ALL] [...keywords...]

Where:

queue Optional positional parameter which specifies the spool queue to be searched. Default is PR.

ALL Optional positional parameter which indicates that ALL sub-files found to match are to be processed.

keywords See section 3.49.2.

Example:

LT JOB=COB74 COL=20

Would select for listing on the terminal any entry in the (held) PR queue that has a job name "COB74". The listing is to display (80) columns starting with column 20.

Error Conditions:
None.

-+*+-

3.49.11 PRINT SPOOL FILE — SPL: print

This command will print selected spool sub-files on the auxiliary printer attached to the terminal.

Syntax:

Print [queue] [,ALL] [...keywords...]

Where:

queue Optional positional parameter which specifies the spool queue to be searched. Default is PR.

ALL Optional positional parameter which indicates that ALL sub-files found to match are to be processed.

keywords See section 3.49.2.

Example:

P ALL JOB=COB74

Would select for printing on the AUX1 printer sub-files in the (held) PR queue that have job name "COB74".

Error Conditions:
None.

-+*+-

3.49.12 PRINT SPOOL FILE WITH TEST PAGE SPL: pt

This command is similar to the PRINT command. A test page (similar to the test page generated by the batch output writer) will be sent to the auxiliary printer. The user may find that this is preferable when printing forms that require delicate alignment.

Syntax:

PT [queue] [,ALL] [...keywords...]

Where:

queue Optional positional parameter which specifies the spool queue to be searched. Default is PR.

ALL Optional positional parameter which indicates that ALL sub-files found to match are to be processed.

keywords See section 3.49.2.

Example:

```
PT JOB=PAYROLL FORM=CHEX
```

Would select for printing on the auxiliary printer sub-files in the (held) PR queue that have job name "PAYROLL" and have form name "CHEX". Test (alignment) pages will be produced.

Error Conditions:
None.

-+*+-

3.49.13 END SPL PROGRAM AND LOGOFF — SPL: quit

This command will terminate the SPL program normally. If the SPL program was executing at program stack level one (ie: not called from another program) the user will be logged off TIP/30.

Syntax:

Quit

Where:
No parameters required.

Example:

Q

Error Conditions:
None.

-+*+-

3.49.14 RELEASE SPOOL FILE **SPL: release**

This command will release sub-files(s) for batch processing. This command is intended to be a mechanism to allow the user to release a held sub-file that is now to be printed.

Syntax:

Release [queue] [,ALL] [...keywords...]

Where:

queue Optional positional parameter which specifies the spool queue to be searched. Default is PR.

ALL Optional positional parameter which indicates that ALL sub-files found to match are to be processed.

keywords See section 3.49.2.

Example:

RE JOB=COB74

Would select for release any sub-file in the (held) PR queue that has a job name "COB74".

Error Conditions:
None.

-+*+-

3.49.15 SUMMARIZE SPOOL QUEUE CONTENTS SPL: summary

This command will list (on the terminal) the sub-files that exist in the specified class and queue which match the selection keywords.

By using this command the user can browse through the spool file to determine which spool sub-files exist.

Syntax:

S [queue] [,ALL] [...keywords...]

Where:

queue Optional positional parameter which specifies the spool queue to be searched. Default is PR.

ALL Optional positional parameter which indicates that ALL sub-files found to match are to be processed.

keywords See section 3.49.2.

Example:

S H=N

Would summarize the sub-files that are not held (queued) in the PR queue.

Error Conditions:
None.

-+*+-

3.49.16 WRITE SPOOL FILE TO EDIT BUFFER SPL: write

This command will select sub-files to be written to a TIP/30 edit buffer. The spool sub-file data will be copied to an edit buffer with the specified name.

Syntax:

Write [queue] [,buffer] [...keywords...]

Where:

queue Optional positional parameter which specifies the spool queue to be searched. Default is PR.

buffer Optional positional parameter which names the output edit buffer. Default is "SPOOL".

The edit buffer will be created with a group name equal to the user's group one specification.

keywords See section 3.49.2.

Example:

```
WR ,MYCOMP  JOB=COB74
```

Would create an edit buffer named "MYCOMP" containing the contents of a (held) print sub-file with job name "COB74".

Error Conditions:

None.

Additional Considerations:

This command writes 80 columns to the edit buffer. The COL= keyword may be used to some advantage.

-+*+-

3.49.17 WRITE SPOOL FILE TO FILE/ELEMENT SPL: wl

This command will write spool sub-files to a specified OS/3 library element.

Syntax:

WL [queue] [,file/elt] [...keywords...]

Where:

queue Optional positional parameter which specifies the spool queue to be searched. Default is PR.

file/elt Optional positional parameters which specify the output library and element name.

Default is RUN/SPOOL.

keywords See section 3.49.2.

Example:

```
WL ,TSTSRC/MYCOMP  JOB=COB74
```

Would write to library TSTSRC, element MYCOMP, sub-files in the (held) PR queue that have job name "COB74".

Error Conditions:

None.

Additional Considerations:

This command writes 128 columns to the specified element. The COL= keyword may be used to some advantage.

-+*+-

3.50 DISPLAY TIP/30 STATISTICS STATUS

The transaction 'STATUS' is a standard TIP/30 utility which is made available to the user. It is designed to give an insight into online system performance by probing internal tables and elapsed time counters maintained by TIP/30. By identifying disproportionate resource utilization, STATUS gives direction for action in tuning the system.

Syntax:

```
STATUS cmd
STATUS P cmd
STATUS PAUX cmd
```

Where:

STATUS is the catalogued transaction code for the STATUS program.

cmd is a command for status. Acceptable commands are:

```
A - produce all statistics
B - file buffer usage
D - disk device usage
F - fast load index
I - I/O summary
K - key holding table
R - program control tables
S - system statistics
T - terminal usage
```

P the report is spooled to the batch printer (PRNTR).

PAUX the report is printed on the auxiliary printer attached to your terminal (AUX1).

Additional Considerations:

If the STATUS program is specified as the system shutdown program (see section on TIP system generation), the STATUS program will perform a "P A" (print all statistics) function when TIP/30 is ended via "EOJ".

3.50.1 FILE BUFFER USAGE **STATUS: b**

Display the current occupant of each file buffer, and the number of swaps which have occurred.

Example:

TIP/30 BUFFER	SWAPS	FILE SIZE	BUFFER OCCUPANT	FILES	STATISTICS I/O'S	OUTPUTS
1	3	1,792	PRNTR	4	90	90
2	1	2,048	ISAM1	13	1,071	4
3	3	1,792	DOC	11	551	30
4	1	5,120	INITDTA	15	0	0
5	1	1,792	DDPOUT	6	0	0

Where:

BUFFER is the file buffer number.

SWAPS is the number of swaps done in this buffer.

SIZE is the size of the buffer in bytes.

OCCUPANT is the name of the file currently in the buffer.

FILES is the total number of files assigned to this buffer.

I/O'S is the number of logical input/output requests done using this buffer.

OUTPUTS is the number of update/add requests done using this buffer.

-+*+-

3.50.2 DISK DEVICE USAGE **STATUS: d**

For each disk volume which is assigned to TIP/30, a list of file names (LFD's) and I/O count for the file. For each disk the sum of I/O requests and the actual EXCP count is displayed.

Example:

```
I / O    S U M M A R Y   B Y   D V C
=====================================
I/O SUMMARY FOR REL071          TIP$CAT          198
                                TIP$RNDM         253
                                SYSGEN             0
                                SAM1               0

TOTAL (KNOWN) I/O FOR DVC ---                    451
EXCP'S                                         5,240

I/O SUMMARY FOR ARCSPL          TIPLOD             0
                                TSTSRC             0
                                ISAM1          1,071

TOTAL (KNOWN) I/O FOR DVC ---                  1,071
EXCP'S                                         3,836

I/O SUMMARY FOR ARCRUN          TIP$SWAP         248
                                TIP                0
                                DOC              551
                                MAC                0
                                INITDTA            0
                                DAM1               0
TOTAL (KNOWN) I/O FOR DVC ---                    799
EXCP'S                                         3,749
```

-+*+-

3.50.3 FAST LOAD INDEX STATUS: f

Display the programs currently in the fast load index and the memory blocks to which each has been relocated.

The fast load index is only used for non-re-entrant transaction programs. Its purpose is to improve the initial loading of such programs.

Example:

T I P / 3 0 F a s t L o a d T a b l e

Loadm	Page	Size	Loadm	Page	Size
TT$RUN	11	2K	TT$DOC	20	22K
TT$LIB	20	18K	TT$LIB	15	18K
TT$SPL	15	18K	TT$SYS	20	4K
TT$MAL	15	10K	TT$IDA	13	12K

Where:

Loadm is the load module name.

Page is the page number to which to program has been relocated.

Size is the size of the program (K=1024).

-+*+-

3.50.4 I/O SUMMARY **STATUS: i**

Display a summary of the OS/3 files which have been used by TIP/30; the number of programs currently assigned to a file; the sum of logical file accesses both for input and output; the file buffer associated with each file named.

Example:

```
T I P / 3 0   F I L E   S U M M A R Y
FILE     USERS    I/O'S  OUTPUTS  BUFFER
======== ===== ======== ========  ======
PRNTR        1       80       80       1
DOC                 551       30       3
ISAM1             1,071        4       2
========       ======== ========
*TOTAL*           1,702      114
```

Where:

FILE is the file name as generated into TIP/30.

USERS is the number of programs currently using the file.

I/O'S is the number of logical input/output requests.

OUTPUTS is the number of update/add requests. This number is included in the I/O'S figure.

BUFFER is the buffer number where the file resides.

Additional Considerations:

Files which have no current users and have zero I/O counts are not listed.

-+*+-

3.50.5 KEY HOLDING TABLE **STATUS: k**

Display the current contents of the key holding table within TIP/30.

Example:

```
T I P / 3 0   K e y   H o l d i n g   T a b l e
File     User-id  Term Key
======== ======== ==== ==========================
ARCUST   RJNORMAN ARC2 X'C1C4C5F0F1F4F0F6'
                       C'ADE01406'
```

Where:

File is the file name.

User-id is the name of the user who has the record held for update.

Term is the terminal name where the user is running.

Key is the key value displayed in hexadecimal and character.

-+*+-

3.50.6 RE-ENTRANT PROGRAM TABLE STATUS: r

Display the list of program control tables. The information displayed includes program size, current number of users, total number of times the program has been used.

Example:

T I P / 3 0 R E - E N T R A N T T A B L E

MODULE	LANGUAGE	TYPE	PAGE	SIZE	SWAPS	STATUS	USERS	USED
TT$STS	BAL	TIP	3	16K	3	IN	1	6
TT$TCP	BAL	TIP			1	RES	0	50
TT$LGN	BAL	TIP	3	4K	1	OUT	0	4
TT$LGF	BAL	TIP	13	4K	2	IN	0	2
TT$WHO	BAL	TIP			1	RES	0	1
TT$TV2	COBOL	IMS	1	6K	3	OUT	0	1
GETREC(SER)	BAL	TIP				RES	0	

Where:

MODULE is the load module name.

LANGUAGE is either COBOL or BAL.

TYPE is either TIP or IMS.

PAGE is the page number where the program is currently allocated.

SIZE is the size (in bytes) of the program (modulo 2K=2048).

SWAPS is the number of times that this load module has been read/written to/from TIP$SWAP.

STATUS IN if the program is in memory. RES if the program is permanently resident. OUT if the program is not in memory at this instant.

USERS is the number of users currently using the program.

USED is the number of times the program has been entered.

-+*+-

3.50.7 GENERAL STATISTICS STATUS: s

This command will show statistics accumulated overall in two columns. The first column is since TIP/30 was initiated. The second column is since some more recent time period. This provides a picture of what has happened since TIP/30 initialization and in the most recent time period.

The statistics are then averaged on a per input message basis. This information should present a transaction profile. ie: what happens (on the average) every time someone presses XMIT.

Example:

T I P / 3 0 S T A T U S R E P O R T

T O T A L S	SINCE 82/06/04 12:22	SINCE 82/06/04 13:00
# INPUT MESSAGES	92	57
# OUTPUT MESSAGES	449	109
PROGRAM LOAD REQUESTS	88	53
ACTUAL LIBRARY LOADS	10	3
M.C.S. FORMAT REQUESTS	4	2
M.C.S. FORMAT FILE I/O	2	1
CATALOG REQUESTS	382	192
CATALOG FILE I/O	196	109
SWAP FILE I/O'S (TIP$SWAP)	248	81
DYNAMIC FILE I/O'S (TIP$RNDM)	253	251
ALL TASKS WERE BUSY	0	0
# OF WAITING TERMINALS 5	0	0
# OF WAITING TERMINALS 10	0	
# OF WAITING TERMINALS 15	0	
DATA BASE OPENS FOR	0	0
DATA BASE I/O'S.	0	0

- P E R I N P U T M E S S A G E -		
RESPONSE TIME	0.834	0.772
TRANSACTION SCHEDULING TIME	0.531	0.500
INPUT NOTIFICATION TIME (ICAM)	0.447	0.458
CPU TIME USED	1.080	
SUPERVISOR CALLS (SVC)	405.2	
TRANSIENT CALLS	9.8	
EXCP'S	133.3	
PROGRAM LOAD REQUESTS	0.9	0.9
M.C.S. FORMAT REQUESTS	0.0	0.0
M.C.S. FORMAT FILE I/O	0.0	0.0
CATALOG REQUESTS	4.1	3.3
CATALOG FILE I/O	2.1	1.9
SWAP FILE I/O'S (TIP$SWAP)	2.6	1.4
DYNAMIC FILE I/O'S (TIP$RNDM)	2.7	4.4
INPUT MESSAGE LENGTH	51.4	76.6
OUTPUT MESSAGE LENGTH	72.9	194.4
LIBRARY FILE: RECORDS READ	5.6	
LIBRARY FILE: RECORDS WRITTEN	0.3	
DATA FILE: RECORDS READ	11.5	
DATA FILE: RECORDS WRITTEN	0.5	

-+*+-

3.50.8 TERMINAL USAGE **STATUS: t**

Display number of input and output messages for each terminal.

Example:

TERMINAL	SESSION: INPUT	OUTPUT	TODAY: INPUT	OUTPUT
T312	44	89	44	90
T313	13	20	38	251
ARC1	0	0	0	1
ARC2	0	0	21	122
TRM1	0	0	0	0

TIP/30 TERMINAL STATISTICS

Where:

TERMINAL is the ICAM terminal name.

SESSION: INPUT number of input messages since the current user logged on.

OUTPUT number of output messages since the current user logged on.

TODAY: INPUT number of input messages since TIP/30 initialization.

OUTPUT number of output messages since TIP/30 initialization.

-+*+-

3.51 IMMEDIATE TIP/30 SHUTDOWN STOP

This command will cause TIP/30 to shut down immediately. It will not wait for all users to log off.

Syntax:

STOP

Where:
No parameters required.

Example:

STOP

Error Conditions:
None.

Additional Considerations:

The system SHUTDOWN program will NOT be scheduled.

Under normal operating conditions, this command should only be issued after an "EOJ" command has been entered. "EOJ" is the preferred method of shutdown. Under certain conditions, a "STOP" command may be necessary to force off users that are running programs that do not recognize system shutdown requested.

3.52 SCHEDULE OS/3 SYMBIONT SYM

SYM is a utility program which interfaces with the OS/3 symbiont scheduler. It allows the user to submit requests to run symbionts in the same manner as the OS/3 console operator. Common commands include RV (run a program) PR (start an output writer) HO (hold an OS/3 queue) etc. An informational message is sent to the OS/3 operator console whenever a symbiont is scheduled by SYM. The message informs the operator that a symbiont command was issued and also shows the user name and terminal name of the submittor.

The SYM program may be run interactively or may be given a single command on the command line. If SYM is run interactively the user will be prompted for each command; if a command is provided on the command line SYM will attempt to execute that command and then terminate normally.

If the SYM program detects that it has been called via a transaction name other than "SYM" then it will assume that the transaction name is the desired command and will also assume that the parameters on the command line are associated with the transaction name. This composite command will be attempted and then SYM will terminate normally.

Syntax:

command parameters

Where:

command The two character name of the desired symbiont. The following symbiont names are supported:

BE CA CH DE D1...D9 HO PD PR PU RB RU RV.

Refer to OS/3 console operator documentation for details concerning the use of these commands. Also recognized are: "End" or any function key (end the SYM program) "Quit" (end the SYM program and logoff).

parameters The appropriate parameters for the requested symbiont.

Example:

```
PR BX,JOB=TIP30
```

This example would start a burst mode output writer to print any print spool files with a job name of "TIP30".

Additional Considerations:

The SYM program may be called from TIP/30 native mode programs [via the TIPSUB linkage mechanism (see TIP/30 PROGRAM MANAGEMENT ROUTINES)].

When invoked in this manner SYM expects the command and parameters in free format in the text area of the CDA (bytes 73 through 152).

If an error is detected, byte 73 of the CDA will be set to X'FF' otherwise byte 73 of the CDA will not be altered. This facility is extremely useful for submitting OS/3 commands from an on-line program.

SYM allows the user to invoke the cancel symbiont (CA) but will not allow any attempt to cancel the currently executing TIP30 job or any ICAM symbiont.

The distributed version of TIP/30 includes catalogue entries for a number of transactions that are in fact quick ways of calling SYM to perform a single function. For example, there is a transaction named "RV" which references the SYM load module. The existence of this transaction means that the "RV" transaction can have a low enough security to enable programmers to use it, but that the more powerful SYM transaction could have a higher security level and thus be unavailable to programmers. It is through this technique that the use of individual symbionts may be restricted.

3.53 SYSTEM STATUS SYS

SYS is a utility program which displays the current status of batch jobs in the OS/3 environment.

Syntax:

SYS opt

Where:

opt	is one of the item discussed below.
A	Similiar to "J" (see below) except that symbionts and shared code modules will also be listed.
End	End the SYS program normally.
J	Produces a list of the jobs which are currently running in batch. It details the decimal memory size, program, job step, job number, CPU seconds elapsed, base key priority and free memory regions.
Quit	End the SYS program normally and logoff TIP/30.
W	At 20 second intervals execute the "J" function.
WA	At 20 second intervals execute the "A" function.
Wait 'jobname'	Iteratively produces a list of the jobs which are currently running in batch until the job you have named starts and subsequently terminates.

Example:

```
SYS J           : display OS/3 job information
.SYS W COB74    : start background program to monitor
                  progress of job named "COB74".
```

Additional Considerations:

If SYS is run as a background program with the WAIT function (ie .SYS W jobname), then it will notify the initiating user with an unsolicited message when 'jobname' has started and when 'jobname' has terminated.

This allows the user to continue with other interactive activities SYS monitors the batch job asynchronously in the background.

When SYS is running in continuous display mode, press MSG-WAIT to interrupt the display.

If SYS is entered with no command then it will produce the 'Jobs' display and prompt you for another command.

3.54 TASK CONTROL BLOCK DISPLAY TCB

The transaction 'TCB' is a utility program which displays task control blocks that are attached to the OS/3 switch list. The program details job name, memory region in hex, size in hex, type, program executing, CPU time, account number, protect key, switch list and scheduling priority.

Priority numbers displayed are the actual displacement from the head of the switch list; hence the first user priority is 4. For transients and the supervisor overlay area (SOA), the number displayed in the account field is actually the transient, or SOA overlay ID. and the name in the program field is the overlay name.

Syntax:

TCB [wait]

Where:

wait Will instruct the TCB program to continuously display (at 20 second intervals) the OS/3 TCB information.

If wait is not specified, the TCB program will display the current OS/3 TCB information and terminate normally.

3.55 TIP FLAG MANIPULATION TIPFLG

The TIP/30 system has 32 flag bits that are accessible by all on-line programs. The 32 flag bits may be considered to be roughly analagous to the OS/3 job control UPSI bytes.

The utility program TIPFLG is provided as a transaction to interrogate or change the setting of any of the flag bits.

The flag bits may also be manipulated by an on-line native mode program (see section on the Program Control System), or by the console operator.

Before using this transaction in a cavalier fashion, the user is advised to check with the installation administrator. Some of the 32 bits may be used for specific scheduling purposes and should not be modified without careful consideration.

Syntax:

command [,bit1 ,bit2 ,bit3 ,bit4 ,bit5 ,bit6 ,bit7]

Where:

command The TIPFLG command chosen from the following list:

"WANYS" - wait for specified bits to be on

"WALLS" - wait for all to be on

"WSETC" - wait for specified bits to be on then set them off

"WANYC" - wait for specified bits to be off

"WCLRS" - wait for specified bits to be off then set them on

"SET " - set specified bits on

"CLEAR" - set specified bits off

"FLAGS" - display current flag bit status

bit1-7 Optional parameters where the user may specify up to 7 bits that are to be acted upon by the specified command.

Bits are numbered 0 through 31.

Example:

```
TIPFLG FLAGS              : display current bits status
TIPFLG CLEAR 0,1,2        : turn of bits 0, 1, and 2
TIPFLG WANYS              : wait for any bit to be set
```

Additional Considerations:

The TIPFLG program is NOT an interactive program. The required parameters are entered on the command line.

3.56 ON-LINE LIBRARIAN TLIB

TLIB is a utility program that provides on-line librarian facilities. The user may manipulate OS/3 library elements, QED edit buffers, and terminal auxiliary devices (cassette, diskette, printer).

TLIB will not create an edit buffer - but it will allow the user to specify an edit buffer as an input. TLIB will manipulate library elements that are type source (S) or macro (M) or proc (P); object modules and load modules may NOT be accessed via TLIB.

For certain commands, TLIB recognizes two pseudo types: directory "D" and fast directory "F". Directory implies the library header information including module name, module type, comments, date and time stamp (similar to a LIBS table of contents listing) whereas fast directory implies just the module name and module type.

TLIB recognizes the following commands:

BACK	- re-activate the previous version of an element
COPY	- copy an element or edit buffer to an element
DELETE	- delete a library element
END	- end TLIB interaction
HELP	- display help information on terminal
JOB	- submit an element or edit buffer to the remote batch reader queue
LIST	- list (on the terminal) an element or edit buffer
PRINT	- print a listing of an element or edit buffer
PUNCH	- punch an element or edit buffer
QUIT	- end TLIB interaction and logoff

TLIB may be used interactively or may be given a single command on the command line. If a single command is given on the command line TLIB will attempt only that command and terminate. When used interactively, TLIB will prompt the user for each command.

If TLIB detects that it has been called with a transaction name other than "TLIB", it will assume that the transaction code IS the command and not treat the first parameter as a command.

3.56.1 RE-ACTIVATE PREVIOUS VERSION TLIB: back

When an element of an OS/3 library is deleted, the module is not physically removed - the index entry for it is merely marked as logically deleted. The BACK command simply marks the currently active element as removed and finds the previous version and re-activates its directory entry. Elements that are marked as logically deleted are physically removed during a library pack operation. The BACK command may be issued several times in succession to go back a number of versions (if they still exist). If there is not a current active version of an element (for example, the user inadvertently deleted an element) then the user must first create a (dummy) current version before using a BACK command.

Syntax:

```
Back     file,element [,type]
```

Where:

file the catalogued logical file name of the OS/3 library

element the name of the desired element

type the type of the element (Source, Macro or Proc) default S

Example:

```
BACK     JCS/MYJOB
```

Will delete the current active element named "MYJOB" in the library "JCS" and re-activate the most recent previous version of that element.

Error Conditions:
The specified element may not currently exist or the file name may be invalid or it may not be possible to locate a "previous" version of the element.

-+*+-

COPY ELEMENT

3.56.2 COPY ELEMENT — TLIB: copy

This command will copy an existing library element or edit buffer to a specified output library element or auxiliary device. The number of lines copied is reported upon completion of the copy command.

Syntax:

Copy file [,elt] [,type] ,out-file [,out-elt] [,out-type]

Where:

file the catalogued logical file name of the input library file or edit buffer name or auxiliary device.

elt the input element name (not required if an edit buffer)

type the type of input element [default is source (S)]; must be "E" for edit buffer

out-file the catalogued logical file name of the output library file or an auxiliary device id (eg: AUX3 etc)

out-elt the name of the output element (default is same as the input element name)

out-type the output element type (default is same as input type)

Example:

```
COPY  JCS/TIP30,,TEST/TIP30BAK
```

This example illustrates copying the jcl for TIP/30 from the system YJCS library (assumed to be catalogued with a logical file name of "JCS") to a test library under the name "TIP30BAK".

Example:

```
COPY  JCS/TIP30,,AUX3
```

This example illustrates copying the jcl for TIP/30 to the issuing terminal AUX3 device (presumably a cassette or diskette style device).

Error Conditions:

The input file/element or edit buffer may not be found or the output file may not be available for use.

Additional Considerations:

Note that the output "file" may be an auxiliary device or may be an OS/3 queue such as "RDR" "RDR96" or "RBPIN". If the output file is specified as one of these queues, the output element name is taken as the LBL name of the queue element that is created. It is not possible to specify the same OS/3 library as both input and output.

-+*+-

3.56.3 DELETE ELEMENT — TLIB: delete

This command will delete an element from an OS/3 library. The element is marked "deleted" in the directory of the library; it is not physically removed from the file until such time as a pack operation is performed by the batch OS/3 librarian (LIBS). TLIB does not provide a facility for deleting edit buffers.

Syntax:

DELete file,element [,type]

Where:

file the catalogued logical file name of the OS/3 library

element the name of the element to be deleted

type the type of the selected element [default is source ("S")]

Example:

DELETE JCS/MYJOB

Will delete element "MYJOB" from library "JCS".

Error Conditions:
The specified element may not exist or the file cannot be accessed.

-+*+-

3.56.4 END TLIB PROGRAM TLIB: end

This command will cause TLIB to terminate normally.

Syntax:

End

Where:
No parameters required.

Error Conditions:
None.

-+*+-

DISPLAY HELP INFORMATION

3.56.5 DISPLAY HELP INFORMATION TLIB: help

This command will summarize the commands that are recognized by TLIB and the required parameter syntax.

Syntax:

Help

Where:
No parameters required.

Error Conditions:
The help information may be unavailable.

-+*+-

3.56.6 SUBMIT REMOTE BATCH JOB TLIB: job

This command will submit a library element or edit buffer to the remote batch reader queue. This command should only be issued if the OS/3 supervisor has been generated with support for remote spooling. If such is not the case, unpredictable results may occur (including the possibility of an unrecoverable HPR). After the element or edit buffer has been written in the remote batch reader queue TLIB will automatically call the "RB" symbiont to start the remote reader.

Syntax:

```
Job   file [,element] [,type]
```

Where:

file the catalogued logical file name of the library or the name of an edit buffer

element the name of a library element [not required if type is specified as edit buffer ("E")]

type the type of input [default is source ("S")]

Example:

```
J   RUN/QUIKJOB,s
```

Will submit a source element named "QUIKJOB" from library "RUN" to the remote batch reader and invoke the RB symbiont to process it.

Error Conditions:
The named element or edit buffer may not exist or the file cannot be accessed or the type may be invalid.

-+*+-

LIST ELEMENT ON TERMINAL

3.56.7 LIST ELEMENT ON TERMINAL TLIB: list

This command will list a library element or edit buffer at the terminal. The listing will be produced in "burst" mode; that is, it will continue as quickly as possible until completed or until the user presses the MSG WAIT key. If the user presses MSG WAIT, he will be notified that the listing has been halted and asked whether or not to continue listing. All 80 "columns" of the element or edit buffer will be displayed on the terminal.

Syntax:

List file [,element] [,type]

Where:

file the catalogued logical file name of a library or the name of an edit buffer.

element the name of an element in the library (may be omitted if type is specified as Edit buffer or Directory or Fast Directory).

type the type of input [default is source ("S")]; other choices include directory ("D") or fast directory ("F") of a library.

Example:

```
LIST  JCS,,D
```

Will list the directory of the file catalogued with logical file name "JCS".

Error Conditions:
The named element may not exist or the file cannot be accessed or the type may be incorrect.

-+*+-

3.56.8 PRINT HARD COPY LISTING TLIB: print

This command will create a hard copy printout of a library element or edit buffer or library directory at the site printer or an auxiliary print device. TLIB is aware of the declared format of a library element (ie: COBOL or Assembler or RPG etc) and will recognize COBOL page skip statements (a "/" in column 7) and assembler eject statements and the like and produce a printout that is somewhat more presentable than a simple list of the lines. Unless TLIB is advised otherwise, print files sent ot the site printer are preceded by a separator page to facilitate identification of the printout. Each TLIB print request to the site printer is breakpointed by TIP and may be printed by starting a burst mode output writer (ie: OS/3 operator command "PR BX,JOB=TIP30").

Syntax:

Print file [,element] [,type] [,printer] [,header] [,case]

Where:

file the catalogued logical file name of the library or the name of an edit buffer

element the name of a library element (must be omitted if type is edit buffer or directory or fast directory)

type the type of input [default is source ("S")]

printer the destination printer [default is the site printer (PRNTR)] other possibilities are (for example) AUX1 or AUX1*BYP etc.

header YES/NO choice of a header (separator) page. Default is "N" if the destination is an AUX printer, otherwise default is "Y".

case the choice of upper or lower case printing. Default is upper case ("U") if printer is the site printer otherwise default is lower case ("L").

PRINT HARD COPY LISTING

Example:

PR jcs/tip30,,aux1,n,U PR TX007,,E

Will print source element named "TIP30" from the library with catalogued logical file name "JCS" on the terminal auxiliary printer without a separator page (too noisy!) and with all alphabetic characters translated to upper case.

Error Conditions:
The specified element or edit buffer was not found or the file could not be accessed or the type is invalid.

-+*+-

PR TX007,,E Prints EDIT Buffer TX007

PR JCS,,D

PR JCS,,F

3.56.9 PUNCH ELEMENT TLIB: punch

This command will create a PUNCH file from a library element or edit buffer or library directory at the site punch.

Syntax:

PUnch file [,element] [,type] [,punch]

Where:

file the catalogued logical file name of the library or the name of an edit buffer

element the name of a library element (must be omitted if type is edit buffer or directory or fast directory)

type the type of input [default is source ("S")]

punch the destination punch [default is the site punch (PUNCH)].

Example:

PUN jcs/tip30

Will punch source element named "TIP30" from the library with catalogued logical file name "JCS" to the site punch.

Error Conditions:
The specified element or edit buffer was not found or the file could not be accessed or the type is invalid.

-+*+-

3.56.10 QUIT TLIB PROGRAM — TLIB: quit

This command will cause the TLIB program to discontinue prompting the user for more commands and will terminate the TLIB program normally. If the TLIB program was executing at stack level one (ie: TLIB was NOT called by another program) the user will be logged off the TIP/30 system.

Syntax:

Quit

Where:
No parameters required.

Error Conditions:
None.

-+*+-

3.57 ON-LINE 8080 CROSS ASSEMBLER UTSASM

UTSASM is an assembler that accepts the the INTEL 8080 assembler language as input. The UNIVAC MAC80 language is the same with the exception that macros have not yet been implemented in UTSASM.

The COPY psuedo-op has been added to enable the programmer to include other source modules.

The format of the 'COPY' statement follows:

```
COPY FILE/ELT
```

The program will prompt you for the input source file name and the output object file name; if no object file name is given then none is produced. The assembly listing is spooled out to the PRNTR file.

Note:

- To use the online 8080 cross assembler, you must specify a maximum program size of at least 32000 in the TIP/30 generation (ie: MAXPROG=32000).

3.58 DISK VOLUME TABLE OF CONTENTS VTOC

VTOC is a utility program that will display the volume table of contents of a disk. The selected disk must be one that is assigned to TIP/30 via job control; that is, the VTOC program cannot access any physical disk that is not allocated to the TIP/30 job. VTOC will also compute the available free space on a volume and indicate the size of the largest contiguous free area.

VTOC recognizes the following commands:

Display	-	display detailed file information
End	-	end VTOC program
Free	-	display available free space
Help	-	display command help information
List	-	list files on volume
Print	-	print vtoc listing
Quit	-	end VTOC program and logoff
Sort	-	display command (sorted by filename)
Volumes	-	display volumes allocated to TIP/30 job
Write	-	create library element of JCL statements

The VTOC program may be used interactively or may be given a single command on the command line. If used interactively, VTOC will prompt the user for each command. If a single command is entered on the command line, VTOC will attempt that command and then terminate normally.

3.58.1 DISPLAY FILE INFORMATION VTOC: display

This command will display (on the terminal) detailed information about selected files. The information includes record count, allocation, file type etc.

Syntax:

```
Display    volume    [,prefix]
```

Where:

volume the volume serial number of the selected disk (six characters).

prefix optional prefix of file names to select. If omitted all filenames will qualify.

Example:

```
D  ARCRES,!$Y$
```

Will display the files on the volume "ARCRES" that have a filename that does NOT begin with "Y".

Error Conditions:
The specified volume may not be mounted or may not be allocated to the TIP/30 job or there may not be any files found matching the specified file name prefix.

-+*+-

END VTOC PROGRAM

3.58.2 END VTOC PROGRAM — VTOC: end

This command will end interaction with the VTOC program and terminate the VTOC program normally.

Syntax:

End

Where:
no parameters required.

Example:

E

Error Conditions:
no error conditions known.

-+*+-

3.58.3 FREE SPACE ON VOLUME VTOC: free

This command will display (on the terminal) the free space available on a disk volume. The total free space and the size of the largest available contiguous area is given.

Syntax:

Free volume

Where:

volume the volume serial number of the desired disk. (six characters).

Example:

```
F  ARCRES
Will display the disk type of disk volume "ARCRES",
the total available free space on the volume,
and also display the size of the largest available
contiguous area.
```

Error Conditions:
The specified volume may not be mounted or may not be allocated to the TIP/30 job.

-+*+-

3.58.4 DISPLAY HELP INFORMATION VTOC: help

This command will display (on the terminal) help information which will summarize the command syntax recognized by the VTOC program.

Syntax:

Help

Where:
no parameters required.

Example:

HELP

Error Conditions:
No known error conditions.

-+*+-

3.58.5 LIST FILES ON VOLUME VTOC: list

The list command displays (on the terminal) a summary of information about the files on a selected disk. The information includes the LBL name of the file, the file organization, the block size and record size, the number of records etc. All files on a volume may be selected or a prefix may be given to select files by a 1 to 7 character prefix.

Syntax:

List volume [,prefix]

Where:

volume the volume serial number of the selected disk. (six characters).

prefix optional prefix to select file names. If omitted, all file names will qualify.

Example:

L ARCRES,*SG$

Will list information about files with names beginning with the prefix "SG$" from the volume "ARCRES".

Error Conditions:
The specified volume may not be mounted or may not be allocated to the TIP/30 job or there may not be any files that match the specified prefix.

-+*+-

3.58.6 PRINT VTOC — VTOC: print

This command will produce a printed VTOC listing. The VTOC information printed is similar to the information given by the LIST command, but the output may be directed to the site printer or a terminal auxiliary printer.

Syntax:

Print volume [,prefix] [,printer]

Where:

volume the volume serial number of the selected disk (six characters).

prefix optional file name prefix to select filenames by a 1 to 7 character prefix. If omitted, all files will qualify.

printer name of the printer to receive the output. Default is the site printer (PRNTR); other possibilities include: "AUX1" or "AUX1*BYP" etc.

Example:

PR ARCSPL,,AUX1

Will produce a VTOC listing on the executing terminal auxiliary printer of all files on the volume "ARCSPL".

Error Conditions:

The specified volume may not be mounted or may not be allocated to the TIP/30 job. The specified printer may not be available or no files exist which match the prefix specification.

-+*+-

3.58.7 END VTOC PROGRAM AND LOGOFF VTOC: quit

This command will end interaction with the VTOC program and, if the VTOC program is being executed at stack level one (ie: VTOC was NOT called by another program) the user will be logged off the TIP/30 system.

Syntax:

Quit

Where:
no parameters required.

Example:

Q

Error Conditions:
No error conditions known.

-+*+-

SORTED VTOC DISPLAY

3.58.8 SORTED VTOC DISPLAY VTOC: sort

This command will produce the same output as the "Display" command in sequence by file name.

Syntax:

Sort volume [,prefix]

Where:

volume the volume serial number of the selected disk. (six characters).

prefix optional file name prefix. If omitted, all file names on the selected disk will qualify.

Example:

S ARCRUN,!YRUN

Will produce (at the terminal) a display of VTOC information in file name sequence of all files on disk "ARCRUN" that do NOT begin with the prefix "YRUN".

Error Conditions:

The specified volume may not be mounted or may not be allocated to the TIP/30 job. There may not be any files on the volume that match the specified prefix.

-+*+-

3.58.9 LIST VOLUMES VTOC: volumes

This command will list the volumes currently mounted on the system. The display is similar to the display generated by the OS/3 operator command "MI VI". The display will show the volume name, the device address, and whether or not the volume is allocated (via JCL) to the TIP/30 job.

Syntax:

Volume

Where:
no parameters required.

Example:

V

Will display the volumes currently mounted on the OS/3 system.

Error Conditions:
No known error conditions.

-+*+-

3.58.10 CREATE JCL FOR FILES ON VOLUME VTOC: write

This command will create an OS/3 library element containing the JCL corresponding to the files selected on a disk volume. The element created will have an element name the same as the volume name and will be written to the TIP/30 YRUN library (catalogued logical file name "RUN"). The JCL written for each file includes DVC, VOL, EXT, LBL, SCR, and LFD statements. Once this element has been created, the user may use the TIP/30 editor (QED) to edit the JCL to suit his requirements. This process is very useful for creating backup/restore job control streams or for creating a job control stream to catalogue (in the OS/3 catalogue) selected files on a selected disk volume.

Syntax:

```
Write   volume  [,prefix]
```

Where:

volume the volume serial number of the selected disk. (six characters)

prefix optional file name prefix to select files on the disk by prefix. If omitted, all files on the volume specified will qualify.

Example:

```
WR  ARCRES,!$Y$
```

Will create RUN/ARCRES (element "ARCRES" in library "RUN") containing job control statements for all files on that volume that do not have a filename beginning with the prefix "Y".

Error Conditions:

The specified volume may not be mounted or allocated to the TIP/30 job or there may not be any files on the volume which have filenames that match the specified prefix.

-+*+-

3.59 DISPLAY ACTIVE USERS — WHOSON

The WHOSON utility displays on the terminal a list of active TIP/30 terminals and associated information.

Syntax:

WHOSON/[qual]

Where:

qual An optional qualifier. The qualifier may be one of: a terminal name, user-id, or active file name. The qualifier may also follow standard prefix notation (ie. *AR).

If the qualifier is omitted then a list of all active terminals is produced.

Example:

WHOSON/*TRM

User-Id	Terminal	Program	Lvl	In	Out	Resp	Uns
GEORGE	TRM1	WHOSON	3	7	16	.852	0
MARY	TRM2	VTOC	1	12	17	.652	1

Where:

User-id is the user currently using the terminal.

If an asterisk ("*") precedes the user-id, the terminal is in use without a user logged on (LOGON=NO).

Terminal Is the terminal name. This may be followed by '/DN' if the terminal is marked down by ICAM.

Program is the transaction code of the program currently running on that terminal.

If preceded by an asterisk ("*") the program is currently not in memory (swapped out).

Lvl is the program execution stack level.

In is the number of input messages since logon.

Out is the number of output messages since logon.

Resp average response time (seconds) observed at that terminal.

CPU CPU time (seconds) consumed at that terminal.

(Not available on release 7 and above).

Uns number of outstanding unsolicited messages waiting.

Error Conditions:

None.

Additional Considerations:

All columns are displayed for master level users. Other users receive a truncated display.

3.60 DISPLAY USER INFORMATION WMI

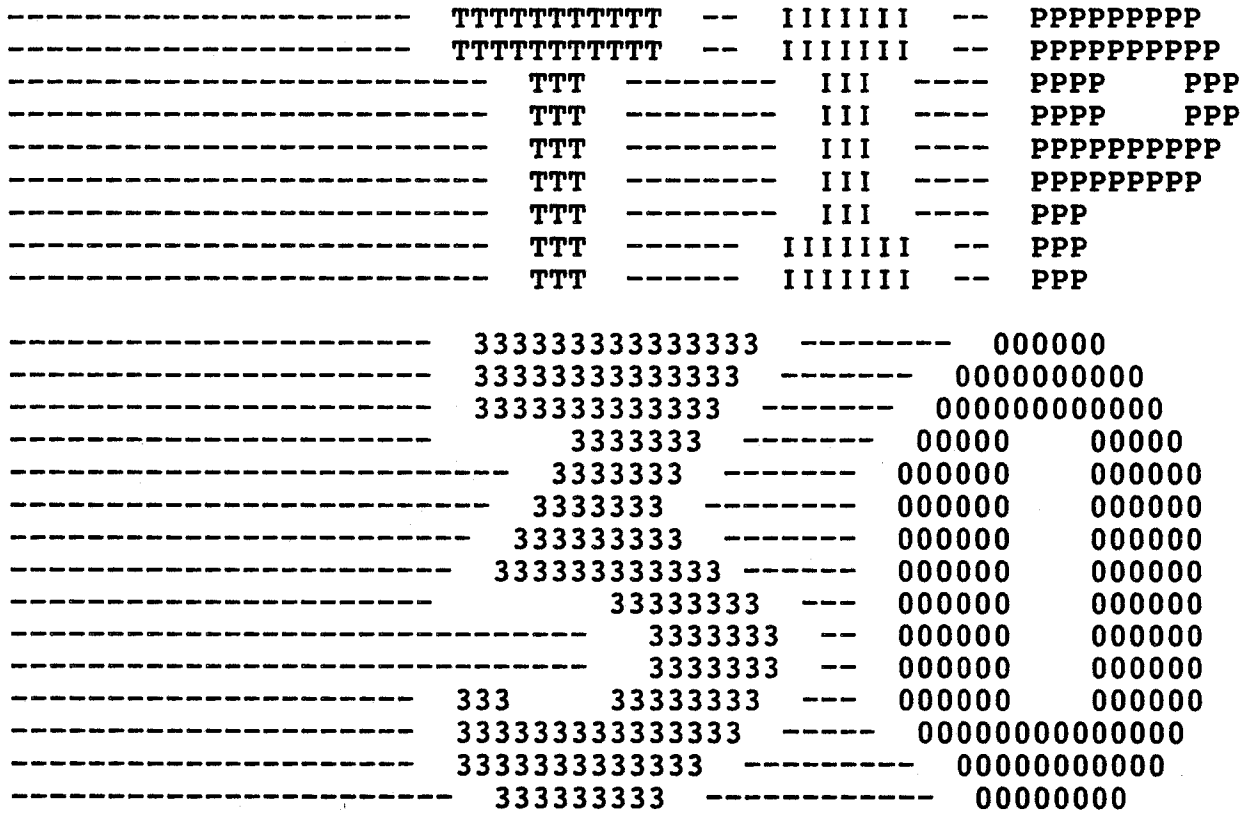
The WMI (who am I?) program displays information on the terminal showing the user-id of the user logged on the terminal, the terminal name (as defined to the system), the current date and time, the version of both TIP/30 and OS/3 that is in use, and the features of TIP/30 that are configured.

The WMI program requires no parameters. The user need only enter the transaction code ("WMI").

Example:
The following is sample output:

```
Hello GEORGE   on terminal T313 at site ABC-CORP

        Date: 82/06/18  Time: 14:13:39    TIP/30 Version: 2.5
ICAM network: NET1      OS/3:   7.1.0
  Attributes: SYSTEM/80 CDM DBMS OPEN DMS
```



TQL Reference Manual

Version 2.5R1 (83/06/01)

TD$TQL

A Product of:

Allinson-Ross Corporation
First Rexdale Place,
155 Rexdale Boulevard, Suite 906
REXDALE, Ontario
Canada M9W 5Z8
TEL: (416) 746-3388
TWX: (610) 491-1772

Allinson-Ross Corporation,
155 Rexdale Blvd, Suite 906,
Rexdale, Ontario,
Canada M9W 5Z8
Tel: (416) 746-3388

THIS DOCUMENT WAS PRODUCED USING THE
ALLINSON-ROSS CORPORATION DOCUMENT GENERATOR.

1. CHAPTER I - INTRODUCTION

1.1 PREFACE

This document is the reference manual for TIP/30 (Transaction Interface Processor), a software product developed by Allinson-Ross Corporation.

The names TIP and TIP/30 are used interchangeably in this manual.

Please direct any inquiries or requests for further information to:

Allinson-Ross Corporation
First Rexdale Place
155 Rexdale Blvd., Suite 906
Rexdale, Ontario
Canada M9W 5Z8
Tel. (416) 746-3388
TWX. (610) 491-1772

TABLE OF CONTENTS

1.2 TABLE OF CONTENTS **TOC**

- 1 -

1.3 THE TIP/30 QUERY LANGUAGE TQL

The TIP/30 Query Language (TQL) is an interactive facility that allows the user to create flexible and powerful 'query' programs. These programs have the capability to display, modify, enter, and report data from on-line files. The TQL system also allows the user to enter unstructured or 'ad hoc' requests. Ad hoc commands enable the user to request the retrieval of data in ways that may not have been explicitly anticipated by the programmer.

TQL allows access to standard Data Management files that are either direct access (DA) or indexed (ISAM, IRAM, MIRAM). In addition, the programmer may choose to make use of a very powerful facility: prototype files. A prototype file is a 'virtual' file that is maintained internally by TQL. The actual file is simulated by TQL. The programmer may alter the size and number of the fields within a record; all such changes take effect as soon as the TQL program is recompiled. The use of a prototype program mechanism allows the programmer to completely design, test and (if desired) implement an application without creating real files. The structure of the files (including any indices) may be altered as testing proceeds and new ideas materialize. When the application is completed, the application users can use the TQL programs until such time as a proper TIP native mode system can be written. Indeed, in some cases, it may be quite justifiable to leave certain limited applications in production although they are actually implemented using prototype files.

TQL programs are written in a language that is based on a subset of the COBOL-74 syntax with the addition of a number of extensions for use by TQL. The TQL program is compiled on-line and the output of the compiler is stored for later "execution" by the TQL run-time interpreter. The run-time interpreter provides an interface between the user and the TQL program. Ad hoc commands are interpreted and executed by the run-time interpreter - the TQL program need not concern itself with any aspect of such unstructured requests for data.

The TQL system is organized around a centralized data dictionary (or contol file). This control file is assumed to have the logical file name TQL$CTL. This file is initialized as part of the installation of TIP/30. The control file contains:

- the source for all existing TQL programs;
- the run-unit code (pseudo-object code) for all compiled TQL programs
- ANY pre-compiled record layouts
- ALL file descriptions and definitions

From this list of contents, we can draw the following conclusions:

- the source for TQL programs is stored in the TQL control file; other copies of this source may, of course, exist in normal OS/3 libraries;
- the 'executable' output of the TQL compiler is stored in the control file;
- record layouts MAY be pre-compiled and used (in common) by a number of TQL programs that need to access such records. This is a significant extension of the idea of using COPY books.
- all files that are to be accessed by TQL programs must be described (compiled) in the control file. The definition of a file (as will be shown) is a simple matter. This seemingly redundant definition of file characteristics is required to enable an implementation of a future batch interface to TQL.

The steps required to create a working TQL application are:

- ensure that all on-line files that are to be accessed are both generated into the TIP/30 system (unless they are prototype files) AND their characteristics have been compiled (defined) to TQL;

- either pre-compile record layouts for the files to be accessed OR include the record layouts in-line (explicitly) in the program;

- write and compile a TQL program that declares which files and record layouts are needed for the particular application AND defines displays and reports that are available to the user.

The following sections of this chapter describe:

- the rules of syntax for the specification of FILES, RECORDS, and PROGRAMS;

- the commands and use of the TQL Monitor Program (TQLMON) - a development environment for the TQL programmer;

- The commands and facilities available at run time to the user of a TQL program.

1.3.1 TQL EXPRESSIONS TQL: expr

TQL allows the programmer or run-time user to make use of arithmetic and relational expressions. These expressions may be used either as part of the TQL program proper or as part of a run-time command (eg: in the run-time "IF" command).

This section describes the syntax of the general TQL expression and contains several example expressions.

Syntax:

```
( field  oper  field  ) [ connector ] ( field  oper  field ) ...
    or          or                       or          or
  value       value                    value       value
```

Where:

( ) The use of parentheses may be necessary to force a specific order of evaluation of the expression or to nest expressions.

If parentheses are not used, standard operator precedence rules apply (multiplication and division before addition and subtraction etc).

field The name of a field that is defined in the TQL program. A list of available field names can be found (at run-time) by using the "SHOW" command (documented in a following section).

value A numeric or character value. Character values are normally enclosed in quotes. (Eg: 38 or 'JANUARY').

oper A relational or arithmetic operator (arithmetic operators may only be applied to numeric fields!). TQL supports the following operators:

OPERATOR	ALTERNATIVE NOTATION	DESCRIPTION
EQ	=	equal
NE	<>	not equal
GT	>	greater than
LT	<	less than
GE	>=	greater than or equal
LE	<=	less than or equal
BEGINS WITH	=*	begins with
DOES NOT BEGIN WITH	=!	does not begin with
CONTAINS	=:	contains
DOES NOT CONTAIN		does not contain
+		arithmetic addition
-		arithmetic subtraction
*		arithmetic multiplication
/		arithmetic division
%		arithmetic remainder

connector A (standard) logical connector. TQL supports the following connectors:

"AND" or "&" - logical "and" function

"OR" or "¦" - logical "or" function

"NOT" - logical negative

Example:

```
MOVE  INVENTORY-COUNT - 1 TO WORK-COUNT.

IF  (JOB-DESCRIPTION CONTAINS 'DEPUTY') AND GROSS-SAL > 25000
    AND GROSS-SAL <= 50000

IF  (NOT JOB-DESCRIPTION =: 'DEPUTY')

IF  0 = TOTAL-COUNT % 2
```

Additional Considerations:

Note that the field name "GROSS-SAL" had to be repeated for the comparison with 50000. This illustrates that TQL does not allow the subject of a comparison to be omitted (unlike COBOL-74).

Numeric fields must be entered without comma separators. (eg: 25000 rather than 25,000).

The remainder operator ("%") implies division, but the result is the remainder rather than the quotient. The example above compares 0 with the remainder when TOTAL-COUNT is divided by 2. If the remainder is zero, it implies that the field is evenly divisible by 2.

The result of a relational test (ie: A >= B) is considered to be equivalent to numeric 1 if the test was TRUE otherwise the value is 0 if FALSE. Such implied numeric values may be used in further computations if required.

-+*+-

1.3.2 FILE DEFINITION TQL: file

The programmer must define all required on-line files to TQL. In order to do this, it is necessary to create a source module [either using the standard TIP/30 text editor (QED) or by using the TQL source editor (described later)]. The source module may contain one or more FILE definitions. This source module is then compiled by TQL (the compilation process will be described in detail in a later section).

The definition of a FILE is similar to the specifications that are used in the TIP/30 generation process. The syntactical requirements are:

Syntax:

```
FILE  filename,filetype
        ACCESS=
        BLKSIZE=
        DELETE=
        INDSIZE=
        IORTN=
        KEYLEN=
        KEYLOC=
        KEY1=
        KEY2=
        KEY3=
        KEY4=
        KEY5=
        RECFORM=
        RECSIZE=
        . (period --> end the file definition)
```

Where:

filename the logical file name as specified in the TIP/30 catalogue.

filetype the type of file. Choose one of ISAM, MIRAM, DAM, DMIRAM, or PROTOTYPE.

ACCESS=opt the access option as described in the data management manual. Default is EXCR.

BLKSIZE=n the block size of the file.

FILE DEFINITION

DELETE=n the zero relative offset of the delete byte in the record. This is currently ignored by TQL (since TQL utilizes the TIP/30 file system) but may be required by a future batch interface to TQL.

INDSIZE=n is the INDEX AREA SIZE for this file. Default=256.

IORTN=name 'name' is the name of a user written I/O routine which is to be called by TQL to do all I/O for this file. This routine must be specified as a resident TIP/30 SUBPROGRAM. The name, therefore, must be the LOADM name of the suprogram.

TQL will call this routine via 'TIPSUBP' and pass the same parameters which would have been passed to 'TIPFCS'.

KEYLEN=n the length of the key for the file.

KEYLOC=n is the zero relative location of the key in the record. Default=0.

KEYLEN and KEYLOC do not have to be specified if the key information is provided by one or more of the keywords KEY1= thru KEY5=.

KEY1= (size,loc,NDUP,NCHG)

defines index 1.

'size' is the key length.

'loc' is the zero relative key location.

Note that TIP/30 does not allow KEY1 of a MIRAM file to change or have duplicates.

KEY2= (size,loc,DUP¦NDUP,CHG¦NCHG)

defines index 2

KEY3= (size,loc,DUP¦NDUP,CHG¦NCHG)

defines index 3

KEY4= (size,loc,DUP¦NDUP,CHG¦NCHG)

defines index 4

KEY5= (size,loc,DUP¦NDUP,CHG¦NCHG)

defines index 5

RECFORM= record format. Choose either FIXBLK or VARBLK. Default=FIXBLK.

Note that the first halfword of a variable length record is the record length. This field is available to the TQL program and the record definition must account for these two bytes (PIC 9(4) COMP-4). During an ADD of a record TQL will set the maximum record length.

RECSIZE=n is the length of the records in the file.

. the end of a file definition must be marked by a period. Other file definitions may follow in the same source module.

-+*+-

1.3.3 RECORD DEFINITION TQL: record

The programmer has several methods of handling record layouts:

- pre-compile the record definition and reference it by name in TQL programs that need to access such records;
- Use the COPY clause to include the record layout in (each of) the TQL programs that access the record;
- explicitly code the record definition in (each of) the TQL programs that access the record.

The first method (pre-compilation) is the most efficient and is highly recommended. Use of the COPY clause is clearly better than explicitly coding the record layout. The latter two methods are inferior. Pre-compilation ensures that all TQL programs use the same record layout and will be the basis for any future support of data dictionary schemes.

The record definition follows standard COBOL-74 record description conventions with the following exceptions:

- The COBOL special names: SPACES, ZEROES, HIGH-VALUES, LOW-VALUES are not recognized by TQL.
- COMPUTATIONAL-1 and COMPUTATIONAL-2 fields (short and long format floating point) are NOT supported by TQL.
- 66 level items (COBOL-74 RENAMES) are ignored.
- 77 level items are ignored.
- 88 level items are ignored.
- VALUE clauses are ignored.
- Only one level of subscripting is supported by TQL; that is, arrays may have only one dimension.

The record layout must be preceded by the (optional) "FOR" clause if it is to be separately pre-compiled. If the record is described explicitly in the program, record definitions immediately follow the associated FILE statement.

The COBOL record layout may contain or be followed by ALLOW CHANGE clauses, VERIFY clauses, ID clauses, ALLOW DELETE clause, ALLOW ADD clause.

Records are not allowed to change or be deleted unless such permission is explicitly granted through the appropriate clauses.

Example:

```
[ FOR filename. ]
RECORD PAYMST.
01  PAYMST.
    05  KEY.
        10  DEPT                PIC 99.
        10  NUMB                PIC 9(5) COMP-3.
    05  NAME                    PIC X(20).
    05  ADDRESS.
        10  LINE-1              PIC X(20).
        10  LINE-2              PIC X(20).
    05  SALARY                  PIC 9(4)V99.
    05  JOBS   OCCURS 4 TIMES.
        10  LOCATION            PIC X(8).
        10  NUMBER              PIC 9(4).
ID IS DEPT > 0.
ALLOW CHANGE ALL.
NO CHANGE DEPT NUMB.
VERIFY SALARY 6000 THRU 32000.
```

Additional Considerations:

The definition of the key field is critical to the operation of TQL at run time. The first definition of the key field(s) is taken as the way the key will be entered at the terminal when selecting records. This is a problem if the key is actually made up of several smaller fields. For example, if the key is defined as three (3) small fields then any key value must be entered at execution time as 3 separate items of the correct type. Numeric data is entered as a number, but alpha-numeric data must be entered in quotes('). If you prefer to enter the key data as one big field but still want to reference the sub-fields then code the record layout with one single field and then redefine it as the sub-fields. Since the single field definition would appear first, TQL would expect the key to to be entered as a single data item.

-+*+-

1.3.4 ALLOWING RECORDS/FIELDS TO CHANGE TQL: allow

Records are not allowed to be added, changed or deleted unless explicit permission is given in the TQL program. Fields within records cannot change unless permission is explicitly given. The ALLOW clause enables the programmer to specify what actions are permitted. The ALLOW clause may appear within a pre-compiled record layout, or within the DATA DIVISION of the TQL program. The program may specify multiple ALLOW clauses for a record.

Syntax:

```
ALLOW ADD.
ALLOW DELETE.
ALLOW CHANGE field-names.
ALLOW CHANGE ALL.
   NO CHANGE field-names.
   NO CHANGE ALL.
```

Where:

ALLOW ADD Indicates that records may be added to the file.

ALLOW DELETE Indicates that records may be deleted from the file.

ALLOW CHANGE defines which fields of the record may be changed when records are being updated.

field-names A list of field names involved. The names may be separated by commas or spaces and the statement should be terminated with a period.

ALL Indicates that all fields are implied.

NO CHANGE Defines fields which may not change.

Example:

```
ALLOW CHANGE ALL.
NO CHANGE SIN.
ALLOW CHANGE SALARY DEDUCTIONS.
```

-+*+-

1.3.5 FIELDS WHICH MUST BE ADDED — TQL: must add

If there are fields which MUST be entered, that is, the field may not have a value of zero if it is numeric or may not have a value of spaces if it is alpha-numeric the programmer may specify the following statements after the record definition. If this clause is not present for a record TQL assumes that the user may or may not enter a value for each field.

Syntax:

```
MUST ADD field-names.
MUST ADD ALL.
```

Where:

MUST ADD defines which fields of the record may not be omitted when a record is added or changed.

field-names is a list of field names involved. The names may be separated by commas or spaces and the statement should be terminated with a period.

ALL implies all field names.

Example:

```
MUST ADD ALL.
MUST ADD SALARY, DEDUCTIONS.
```

-+*+-

1.3.6 RECORD SELECTION TQL: id

Files often contain many different record types. Records may be selected by specifying the ID clause. The ID clause specifies to TQL which record types are to be selected.

Syntax:

ID IS <expression>

Where:

expression A relational expression which is a test for the inclusion of a record. TQL will evaluate the expression on every read or write of the record to determine whether the record is of the correct type.

Example:

```
ID IS REC-TYPE = 'HD'.
ID IS REC-TYPE NE 'HD' AND SAL > 25000.
```

Additional Considerations:

In the first example, the programmer has specified that the field "REC-TYPE" must be equal to the literal "HD". The second example requires that the field "REC-TYPE" is NOT equal to the literal "HD" and the field "SAL" must be greater than 25,000. If a record is read that does not satisfy the condition TQL will ignore that record and proceed to the next record.

-+*+-

1.3.7 FIELD VERIFICATION — TQL: verify

Whenever a record is added or updated, field verification is done by TQL as the data fields are moved from the screen display area to the record build area. Fields may be verified by specifying a list of possible values for each field to be verified. Such statements must follow the appropriate record definition.

Syntax:

```
VERIFY field  'string' THRU 'string'.
VERIFY field  'string', 'string'.
VERIFY field  'string'.
VERIFY field  number THRU number.
VERIFY field  number, number.
VERIFY field  number.
```

Where:

field	is the name of the data field to be verified.
'string'	is some alpha-numeric value
number	is some numeric value
THRU	is used to define a range check.

Example:

```
VERIFY SALARY 10000, 20000, 30000 THRU 55000.
VERIFY TITLE  'V.P.', 'MANAGER', 'GO-FOR'.
```

Additional Considerations:

A field may be tested for specific values and/or range(s) of values.

If the value of a field is found to not meet the required verification TQL will send back an error message which consists of the field name followed by question mark (?). The terminal operator must correct the field in error and continue.

-+*+-

1.3.8 SYSTEM FIELDS TQL: fields

There are several system data fields that are maintained by TQL that are available to the TQL program. They may be used in the same manner as record fields with the exception that ONLY the ERRCODE$ field may be assigned a value.

Field / Format	Definition
AUTHOR$ X(8)	user-id of person who wrote the program.
DD$ 9(2)	current day.
DESC$ X(30)	description of program from PROGRAM-ID clause.
DMY$ 9(6)	current date in DDMMYY format
ERRCODE$ X(1)	a status field [may be set by user].
HH$ 9(2)	current hour.
HHMM$ 9(4)	current time of day in HHMM format.
JUL$ 9(5)	current date in YYDDD (Julian) format.
LINE$ 9(3)	current line number (of report).
MIN$ 9(2)	current minute.
MON$ 9(2)	current month.
PAGE$ 9(5)	current page number (of report).
SITE$ X(12)	site name from TIP/30.
TID$ X(4)	terminal name running the TQL program.
TIME$ 9(6)	time of day in HHMMSS format.
UID$ X(8)	user-id of user running TQL program.
YMD$ 9(6)	current date in YYMMDD format.
YY$ 9(2)	current year.

-+*+-

1.3.9 TQL PROGRAM STRUCTURE — TQL: program

The general syntax of a TQL program is as follows:

```
IDENTIFICATION DIVISION.
PROGRAM-ID.

DATA DIVISION.

FILE <file-name-1>.
RECORD <record-name-1>.
 [ALLOW CHANGE, NO CHANGE, VERIFY, ID, clauses]
 [ALLOW DELETE.]     [NO DELETE.]
 [ALLOW ADD.]        [NO ADD.]
RECORD <record-name-n>.   ...etc...

FILE <file-name-n>.
RECORD <record-name-n>.   ...etc...

[WORKING-STORAGE SECTION.]

 [DECLARATIVES SECTION.]
*
* comments may be entered anywhere
* by entering an asterisk (*) in column 7
*
 [DISPLAY DIVISION.]

 [REPORT DIVISION.]
```

A program may specify as many files and records as are needed for the application. TQL programs may have any number of defined displays and/or reports. There should be at least one display or report in a TQL program.

-+*+-

1.3.10 IDENTIFICATION DIVISION TQL: id division

The IDENTIFICATION DIVISION of a TQL program must appear first and is required in all TQL programs. This division names the TQL program, may provide an informative description of the program and may restrict run-time access of the program to specific TIP/30 users.

Example:

```
IDENTIFICATION DIVISION.
PROGRAM-ID.    progname
               [ 'comments' ]
               [ GROUP=id ]
               [ PASSWORD PROTECT ]
               . (period indicates end of this DIVISION)
```

Where:

progname Up to eight characters (the first of which must be alphabetic) which uniquely identifies the program.

'comments' Up to thirty characters (enclosed in single quotes) which provide a description of the program.

This is the character string returned as the system field 'DESC$' (see previous section "SYSTEM FIELDS").

id An id that specifies which set of users may use the program. A user may use the program if this id matches either their user-id, a group to which they belong or their terminal name.

If no GROUP= clause is specified the program may be run by any user that has access to TQL.

PASSWORD PROTECT If this clause is specified, the programmer will be asked (at TQL compile-time) to supply a password. Whatever password is assigned by the programmer must be supplied by any user who attempts to run the TQL program. The only way to change the password is to recompile the program.

-+*+-

1.3.11 DATA DIVISION — TQL: data division

The DATA DIVISION of a TQL program is required and must follow the IDENTIFICATION DIVISION. The first section of the data division identifies the files and records that are required by the program. Subsequent (optional) sections define program work fields (WORKING-STORAGE SECTION) and exceptional event processing (DECLARATIVES SECTION).

Syntax:

```
DATA DIVISION.

FILE file-name.

[RECORD rec-name.]
[ ALLOW, VERIFY, ID, clause(s) ]

[01  name.     ]
[ 02 ...       ]
[    ...       ]
[ ALLOW, VERIFY, ID, clause(s) ]
```

Where:

file-name name of a pre-compiled file description.

rec-name name of a pre-compiled record description.

name record name of an explicitly defined record that is coded in-line.

Additional Considerations:

Records may be defined by either referring to the name of a pre-compiled record (that is the "RECORD rec-name" clause), or by actually coding the record description in place of the RECORD clause.

More than one record may be specified for a file; more than one file may be specified in a TQL program.

The "ALLOW, VERIFY, and ID" clauses may be specified in a pre-compiled record description, after the RECORD clause, or after the in-line record description.

DATA DIVISION

Example:

```
DATA DIVISION.

FILE PAYMAST.
RECORD PAY-HDR.
RECORD PAY-DETL.

FILE PAYTRANS.
RECORD PAYTRHDR.

01  PAYTRAN.
  05  FILLER                      PIC X(4).
  05  PAYTRAN-ID                  PIC X(2).
  05  PAYTRAN-DATA  OCCURS 12 TIMES.
    10  PAYTRAN-AMOUNT            PIC S9(7)V9(2).

ALLOW  CHANGE  ALL.
ALLOW  DELETE.                ALLOW ADD.
VERIFY    PAYTRAN-ID   'B3'.
MUST ADD  PAYTRAN-AMOUNT.
```

-+*+-

1.3.12 WORKING STORAGE SECTION — TQL: work fields

The WORKING-STORAGE SECTION of the DATA DIVISION of a TQL program is an optional section that may be included by the programmer to define work-fields that are used in computations or other data manipulations. The section must contain only a single 01 level. All fields must be subordinate to this group item. Since VALUE clauses are ignored by TQL, the fields are initialized by TQL to zero or spaces (as appropriate) upon each initial use of a display or report.

Example:

```
WORKING-STORAGE SECTION.

01  WORK-FIELDS.

  05  GRAND-TOTAL          PIC S9(7)V99   COMP-3.

  05  SUB-TOTAL            PIC S9(7)V99   COMP-3.

  05  FULL-ADDRESS.

    10  FULL-ADDRESS-1     PIC X(40).
    10  FULL-ADDRESS-2     PIC X(40).
    10  FULL-ADDRESS-3     PIC X(20).
```

-+*+-

1.3.13 DECLARATIVES SECTION — TQL: declaratives

The DECLARATIVES SECTION of the DATA DIVISION of a TQL program is an optional section that may be included by the programmer to define special processing that is to occur after a specified record is read or immediately before a specified record is written or added.

For example, the programmer may wish to timestamp all records which are written to a file. Rather than have the user enter the current date and time for each record (a tedious and error-prone procedure), the program could accomplish this by specifying the appropriate move statements in the declaratives section. The general syntax of the Declaratives Section is as follows:

Syntax:

```
DECLARATIVES SECTION.

[ ON READ  OF  <record-name>   statements. ]
[ ON WRITE OF  <record-name>   statements. ]
[ ON ADD   OF  <record-name>   statements. ]
```

Where:

ON READ OF clause indicating that the statements which follow are to be executed immediately AFTER any read of the specified record name.

An 'ON READ' clause for one record may contain 'READ' statements to get other records. This is the only way to select records (by using an IF clause at run time) based on the value of data in supplementary records.

ON WRITE OF clause indicating that the statements which follow are to be executed immediately BEFORE any write of the specified record name.

ON ADD OF clause indicating that the statements which follow are to be executed immediately before any new record is added to the file.

If both 'ON WRITE' and 'ON ADD' clauses exist for the same record, TQL will only execute the 'ON ADD' clause for added records. The 'ON WRITE' clause will then only be used when a record is updated (re-written).

<record-name> specifies the record name associated with this clause.

statements One or more statements which are to be executed at the indicated point in time.

The expression may contain a number of TQL statements (including (but not limited to) READ statements, MOVE, COMPUTE etc).

Valid statements for use in the DECLARATIVES SECTION are described as follows:

Syntax:

```
statement-list .

<-- statement-list  is one or more of the following -->

  ( statement-list )
  number ( statement-list )
  ADD expression TO field
  COMPUTE field = expression
  ERROR 'string'
  IF (expression) ( statement-list )
  IF (expression) ( statement-list ) ELSE ( statement-list )
  MOVE expression TO field
  NEXT RECORD
  ON ERROR 'string'
  READ record FROM field
  READ record VIA field
  SUBTRACT expression FROM field
  WHILE expression ( statement-list )
```

Where:

number(statement-list) indicates that the instructions appearing inside the parentheses are to be repeated the specified number of times. This is the simplest way to display more than one record. The "loop" will be exited early if a READ statement fails to retrieve a record from the file.

expression A standard TQL expression (see section 1.3.1).

DECLARATIVES SECTION

ADD the arithmetic expression is evaluated, added to the value of the 'field' and the result stored in the 'field'. Note that result fields as well as fields in the expression may be subscripted when appropirate by either a constant or subscript field.

COMPUTE the arithmetic expression is evaluated, and the result stored in the 'field'.

ERROR The specified string will be used as an error message and the system field "ERRCODE$" will be set to a non-blank value to indicate an error has occurred.

This clause may be used to signal an error condition

Eg: IF (COUNT LT 10) (ERROR 'not enough stock')

IF the relational expression immediately following the IF is evaluated. If it is found to be true then the code inside the parentheses will be executed. If it is found to be false then the code in parentheses following the ELSE will be executed. If no ELSE clause was given TQL continues with the next statement after the IF clause. Nested IF clauses are supported to a depth of 10.

It may be necessary to enclose the expression in parentheses to avoid confusion with subscripting.

MOVE The expression is evaluated, and the result is stored in the 'field'.

NEXT RECORD this will move an "S" to the field ERRCORD$, indicating that the current record is to be by-passed.

ON ERROR This clause may be used following a read statement to specify a string which will be displayed if an error occurred on the read. Expression usually is a literal (eg: ON ERROR 'No part info')

READ record Directs TQL to read the specified record at this point in the generation of the display.

VIA field 'field' is the name of a field which contains the key of the record to be read.

If the READ is being done because a record is about to be ADDed or UPDATEd then the READ is actually a read for update ('GETUP') and the record will be updated back to the file.

FROM field 'field' is the name of field holding (part of) the key for the secondary record. The file is read sequentially until this first portion of the key in the record no longer matches the value in 'field'.

SUBTRACT the arithmetic expression is evaluated, subtracted from the value of the 'field' and the result is stored in the field.

WHILE the relational expression immediately following the WHILE is evaluated. If it evaluates to be true, the code inside the parentheses will be executed. The code is executed repeatedly until the expression evaluates to be false.

. each ON statement list must be ended with a period.

DECLARATIVES SECTION

Example:

```
DECLARATIVES SECTION.

ON READ OF PAYMAST  ADD PAYMAST-SALARY TO WS-TOTAL-SALARY
                    ADD       1        TO WS-PAYMAST-COUNT.

ON WRITE OF PAYMAST MOVE TIME$ TO PAYMAST-TIME-WRITTEN
                    MOVE  YMD$ TO PAYMAST-DATE-WRITTEN.
```

In this example, every time a record named 'PAYMAST' is read, TQL will automatically execute the two ADD statements (which presumably modify some WORKING-STORAGE fields for later use).

Immediately before all writes of records named 'PAYMAST', TQL will automatically execute the two move statements (which take advantage of the system fields to move the current date and time to corresponding fields in the PAYMAST record).

Additional Considerations:

The statements that may be specified in the 'ON READ' or 'ON WRITE' statement can be arbitrarily complex and may include (for example) the usual 'IF' statements etc.

-+*+-

1.3.14 DISPLAY DIVISION

TQL: display

The DISPLAY DIVISION of a TQL program is a division that represents a TQL extension to standard COBOL-74. This division defines the display sets that are available at execution time. Each display set contains statements that specify the fields that are to be displayed. In addition, the display set contains VERBS that specify (to TQL) exactly which records to read. At execution time, the user of the TQL program uses the name of a display set to request the display of data according to the specifications of the display set.

Syntax:

```
name : display-list USING msg-name .

<-- display-list  is one or more of the following -->

  field
  ( display-list )
  number ( display-list )
  ADD expression TO field
  COMPUTE field = expression
  IF (expression) ( display-list )
  IF (expression) ( display-list ) ELSE ( display-list )
  MORE$
  MOVE expression TO field
  NL$
  READ record
  READ record FROM field
  READ record VIA field
  SUBTRACT expression FROM field
  WHILE expression ( display-list )
```

Where:

name is the display name. This name is required and must be unique within a TQL program. This name is used at execution time by the TQL user to request a particular display format.

msg-name This is the name of the TIP/30 screen format which is to be used to control the display format of the data.

field is the name of a data field. If the field is part of an OCCURS clause, it may be followed by the occurrence number such as PART-NUM(3). If no occurrence number is given then the first occurrence is assumed.

The field name may be subscripted by either a literal or another field. A field used as a subscript must be a binary halfword (ie. PIC 9(4) COMP-4). A subscript field may be part of a record structure or a working-storage field.

If the field named is a group item all subfields are processed with appropriate subscripting.

number (display-list) indicates that the instructions appearing inside the parentheses are to be repeated the specified number of times. This is the simplest way to display more than one record. The "loop" will be exited early if a READ statement fails to retrieve a record from the file.

expression A standard TQL expression (see section 1.3.1).

ADD the arithmetic expression is evaluated, added to the value of the 'field' and the result stored in the 'field'. Note that result fields as well as fields in the expression may be subscripted when appropirate by either a constant or subscript field.

COMPUTE the arithmetic expression is evaluated, and the result stored in the 'field'.

IF the relational expression immediately following the IF is evaluated. If it is found to be true then the code inside the parentheses will be executed. If it is found to be false then the code in parentheses following the ELSE will be executed. If no ELSE clause was given TQL continues with the next statement after the IF clause. Nested IF clauses are supported to a depth of 10.

It may be necessary to enclose the expression in parentheses to avoid confusion with subscripting.

MORE$ Marks the point from which the display is to be continued when more detail records are requested at execution-time. The TQL user can request "more" records by entering the "MORE" run-time command or by pressing function key 9.

See following section "Executing TQL programs".

MOVE The expression is evaluated, and the result is stored in the 'field'.

**NL$** This notation may be inserted to indicate (to the automatic screen generation process) that the screen format is to force a new line on the screen at this point. This NL$ specification has no other effect.

READ record Directs TQL to read the specified record at this point in the generation of the display.

VIA field 'field' is the name of a field which contains the key of the record to be read.

If no record could be read on a 'READ VIA' then TQL will skip to the end of the current repeat loop ('number (display-list)') or the to end of the display, whichever comes first.

If more information is to be displayed even if the READ VIA fails, then it would be necessary to include a (dummy) repeat loop such as: '1 (READ fileb VIA field)'.

FROM field 'field' is the name of field holding (part of) the key for the secondary record. The file is read sequentially until this first portion of the key in the record no longer matches the value in 'field'.

SUBTRACT the arithmetic expression is evaluated, subtracted from the value of the 'field' and the result is stored in the field.

WHILE the relational expression immediately following the WHILE is evaluated. If it evaluates to be true, the code inside the parentheses will be executed. The code is executed repeatedly until the expression evaluates to be false.

. each display must be ended with a period.

DISPLAY DIVISION

Example:

```
DEPLST:        READ DEPT-REC,    DEPT-NUM,   DEPT-NAME,
               MOVE 0 TO TOT-SAL,
  MORE$  19 (READ PAYREC FROM DEPT-NUM,
                    NAME, SIN, SALARY,  NL$
             ADD SALARY TO TOT-SAL,
            )  TOT-SAL USING PAYSCRN.
```

In the above example, TQL does the following for each display:

- read a department record (DEPT-REC) and display the fields DEPT-NUM and DEPT-NAME.
- collect up to 19 payroll records (PAYREC) which are in the selected department. The payroll file has the department number as the first part of the key of the record.
- for each PAYREC the fields NAME, SIN, and SALARY are displayed.
- SALARY is accumulated in the field TOT-SAL.
- TOT-SAL is the last field displayed on the screen
- The Message Control System (MCS) screen format name is PAYSCRN. Note that the inclusion of the NL$ notation forces the automatic screen generator to begin a new line in the screen format at that point.
- If more the 19 payroll records exist then the terminal operator may ask for more by pressing function key 9 on the terminal (or entering the run-time command "MORE"). TQL will continue from the point marked by the tag: "MORE$".

Example:

```
DEPSUM:  20 (READ PAY-REC, READ DEPT-REC VIA DEPT-NUM,
                 NAME, SIN, SALARY, DEPT-NAME    NL$
             ) USING PAYDEPT.
```

In the above example, TQL does the following for each display:

- read a payroll record (PAY-REC)
- then read from the department file (DEPT-REC) by using the field DEPT-NUM as a key. DEPT-NUM must be a field in the PAY-REC record.
- for each PAY-REC the fields NAME, SIN, SALARY, and DEPT-NAME (from department record) are displayed.
- The Message Control System (MCS) screen format name is PAYDEPT.
- Repeat up to 20 times (20 reads of PAY-REC).

-+*+-

1.3.15 REPORT DIVISION TQL: report

The REPORT DIVISION of a TQL program is a division that is a TQL extension to standard COBOL. This division defines one or more reports that are available at run-time to the TQL user. Each report has an assigned name that is used by the user to select the report. The report defines the contents of a "logical page" of the physical report. A logical page may consist of more than one physical page. The run-time TQL interpreter will generate the report by repeatedly generating the "logical page" until no more records are available.

The default destination of the report may be either the site printer (EG: PRNTR) or an auxiliary printer. The printout is actually routed by TQL via the TIP/30 printing facility (TIPPRINT).

The user may override the destination of the report at the time the report is requested.

Syntax:

```
name : report-list  ON print-file  [ AT END report-list ] .

<--- report-list  is one or more of the following --->

  field-names
  ( report-list )
  number ( report-list )
  ADD expression TO field
  COMPUTE field = expression
  HOME$
  IF (expression) ( report-list )
  IF (expression) ( report-list ) ELSE ( report-list )
  MOVE expression TO field
  NL$
  READ record
  READ record FROM field
  READ record VIA field
  SUBTRACT expression FROM field
  SKIP$(number)
  TAB$(number)
  WHILE expression ( report-list )
```

Where:

name The report name. This must be unique within a TQL program. At execution time the user will request the production of this report by referring to this report name.

print-file The default report destination. This may be the site printer which is called 'PRNTR', a communications printer such as 'AUX1' or even the name of a printer that data processing has generated into TIP/30 (eg: PRNTR2).

field the name of a data field. If the field is part of an OCCURS clause it may be followed by the occurrence number such as PART-NUM(3). If no occurrence number is given then the first occurrence is assumed. The field name may also be subscripted by some other field. A field used as a subscript must be a binary halfword (ie. PIC 9(4) COMP-4). A subscript field may be part of a record structure or working-storage field.

If the field named is a group item all subfields are processed with appropriate subscripting.

number (report-list) indicates that the instructions coded inside the parentheses are to be repeated the specified number of times. This is the simplest way to process several records. The "loop" will be exited early if a READ statement fails to retrieve a record from the file.

expression A standard TQL expression (see section 1.3.1).

ADD The arithmetic expression is evaluated, added to the value of the 'field' and the result is stored in the 'field'. Note that result fields as well as field involved in the expression may be subscripted (when appropriate) by either a number or subscript field.

COMPUTE The arithmetic expression is evaluated, and the result stored in the 'field'.

REPORT DIVISION

**HOME$** Force a skip to a new page (top of form). The system field "PAGE$" is incremented by one and the the system field "LINE$" is set to zero.

IF The relational expression immediately following the IF statement is evaluated. If the expression evaluates "TRUE" the code which follows in parentheses will be executed. If the expression evaluates "FALSE" the code in parentheses which follows the word "ELSE" will be executed. If no ELSE clause was given TQL continues with the next statement after the IF clause. Nested IF clauses are supported to a maximum depth of 10.

MOVE The arithmetic expression is evaluated and the result is stored in the 'field'.

**NL$** Force a new line. The current contents of the print line are printed. The system field "LINE$" is incremented by one.

READ record Read the specified record name.

VIA field 'field' is the name of the field containing the key of the desired record.

If a record cannot be read on a 'READ VIA' statement TQL will skip to the end of the current repeat loop ('number (report-list)') or to the end of the report, which ever comes first.

If more information is to be reported even though the READ VIA fails, then it would be necessary to include a (dummy) repeat loop such as: '1 (READ file VIA field)'.

FROM field 'field' is the name of the field containing (part of) the key for the secondary record. The file is read sequentially until this first portion of the key in the record no longer matches the value in 'field'.

SUBTRACT The arithmetic expression is evaluated, subtracted from the value of the 'field' and the result stored in the 'field'.

TAB$(number) TQL will position the output pointer into the print line to the exact column specifed by 'number'.

This statement may position the output pointer after the current column location OR before the current column location. The user is responsible for the results of overlapped fields.

SKIP$(number) TQL will advance the output pointer (horizontally) to the right by the number of columns indicated.

WHILE The statements in parentheses following the "WHILE" will be executed repeatedly until the relational expression is false.

AT END When all records have been processed the coding following the words "AT END" will be executed. This provides the capability to generate final totals or summary information of whatever kind.

This clause must appear as the last clause in a report definition.

. each report must be ended with a period.

REPORT DIVISION

Example:

```
REPORT DIVISION.

QOH:  HOME$
      TAB$(15) 'PART - QUANTITY ON HAND' TAB$(70) 'PAGE' PAGE$  NL$
      'PART NUMBER' TAB$(20) 'DESCRIPTION' TAB$(50) 'QUANTITY'  NL$

     50  (READ PARTFIL,
          PM-NUM   TAB$(20)   PM-DESC   TAB$(50)   PM-QTY   NL$
         ) ON AUX1.
```

For each logical page of this report the following is done:

- a new page is forced (HOME$)
- a two line page title is printed.
- up to 50 PARTFIL records are read.
- for each record the fields PM-NUM, PM-DESC and PM-QTY are printed on a separate line (note the NL$)
- the default destination of the report is AUX1. This may be overridden at execution time by the TQL user.

-+*+-

1.4 MAINTAINING THE TQL DICTIONARY TQLMON

The TQL monitor program (TQLMON) is a supplied utility that enables the PROGRAMMER to maintain the contents of the TQL dictionary (or control) file. TQLMON provides sub-functions which allow the programmer to create, edit or compile file or record definitions or TQL programs.

The programmer may use the standard TIP/30 system editor (QED) to create and maintain the source for file, record or program definitions. TQLMON also provides a screen-format oriented editor that is specifically designed for editing TQL source elements.

Each of the commands of the TQL monitor is described in the following sections. The TQLMON program is not normally used by non-programmers (users).

TQLMON COMMAND SUMMARY

C	- Compile file and record definitions
COMP	- Compile program (from dictionary)
CP	- Compile program (from library or edit buffer)
Delete	- Delete files and/or records
DP	- Delete program from dictionary
End	- End TQL monitor
Help	- Display help information
List	- List file and/or record compilation on terminal
LP	- List program compilation on terminal
Mcs	- Generate MCS screen format(s) for a program
N	- enter a new record definition
NF	- enter a new file definition
NP	- call editor to enter a new program
Print	- Print files and records on printer
PP	- Print program on printer
PURGE	- delete all records in a PROTOTYPE file
Q	- call QED then compile record definition
QP	- call QED then compile program
Run	- execute a TQL program (same as OPEN ...)
Summary	- Summary of files and records
SP	- Summary of programs
Update	- Update then compile record definition
UC	- Update control record
UF	- Update file characteristics
UP	- Update then compile program
Write	- Write records to library
WP	- Write program to library

1.4.1 COMPILE FILE/RECORD TQLMON: c

Syntax:

C file [,elt]

Where:

file The catalogued file name of a library file or the name of an edit buffer.

elt The name of the desired element from the file. This parameter should be omitted if the file or record is to be compiled from an edit buffer.

Example:

C SOURCE/PAYREC

Additional Considerations:

The input to this command (either an element of a library or an edit buffer) may contain one or more file definitions or record definitions.

Error Conditions:

Record already exists: <record-name>

- the record already exists and a new one with the same name is not allowed. The existing record must be deleted beforehand (see description of DELETE command following).

<file> not found <record-name> not posted.

- the file named for this record does not exist.

-+*+-

1.4.2 COMPILE PROGRAM — TQLMON: comp; cp

The COMP command causes a TQL program to be compiled directly from the TQL dictionary file (as opposed to compilation from a library element or edit buffer). This command is most often used when a record definition (or file definition) has been changed and all programs that refer to the record have to be recompiled.

The CP command compiles a TQL program from either an edit buffer or a library element.

Syntax:

```
COMP  program

CP    file  [,elt]
```

Where:

program The name of the TQL program to be recompiled. The program must already be in the control file. A programmer may wish to do this if some record or file definition which the program uses has changed.

file the catalogued name of the library containing the source for the program or the name of an edit buffer.

element the name of the element within the library. This parameter should be omitted if the compilation is from an edit buffer.

Example:

```
COMP  TQLTSP

CP    TIP/TQLTSP
```

Error Conditions:

DUPLICATE DISPLAY NAME: <name>

- a display set of that name is already defined in the program.

DUPLICATE REPORT NAME: <name>

- a report of that name is already defined in the program.

File not found: <name>

- the requested file does not exist.

Record not found: <name>

- the requested record does not exist.

EXPR: MISSING ')'

- a required right parenthesis is missing in an expression.

EXPR: INCOMPATIBLE DATA FIELDS

- attempting to compare numeric to non-numeric data or attempting arithmetic operations on non-numeric data.

-+*+-

1.4.3 DELETE FILE/RECORD — TQLMON: delete

The Delete command will delete a single record or single file definition from the TQL dictionary file.

Syntax:

Delete file [,record]

Where:

file the name of a file defined in the TQL dictionary.

record the name of a record of that file.

If 'record' is omitted the file definition is deleted.

Example:

Del PAYMST/PAYREC

Error Conditions:
A file definition cannot be deleted until all associated record definitions have been deleted.

-+*+-

1.4.4 DELETE PROGRAM — TQLMON: dp

The DP command will delete a single program from the TQL dictionary. The executable code will be deleted as well as the source for the program. Note that there is no recovery from this command (unless a backup copy of the source of the program has been stored in an element of a library).

Syntax:

DP progname

Where:

progname the name of a program defined in the dictionary.

Note that the name of the program is determined by the PROGRAM-ID clause in the TQL program.

Example:

DP PAYINQ

-+*+-

1.4.5 END TQLMON PROGRAM — TQLMON: end

The E command will terminate interaction with the TQL monitor program and return to the calling program or the TIP command line (whichever is appropriate).

Syntax:

End

-+*+-

DISPLAY HELP INFORMATION

1.4.6 DISPLAY HELP INFORMATION — TQLMON: help

The help command will display help information for the TQL monitor or will display help information for specific commands.

Syntax:

Help [command]

Where:

command the specific command for which help is required. If this is omitted, the help command will display a list of all available commands.

-+*+-

1.4.7 LIST FILE/RECORD — TQLMON: list

The LIST command will display (at the terminal) the compilation listing of a file or record definition. The listing may be interrupted by pressing the MSG-WAIT key on the terminal.

Syntax:

```
List  file  [,record]
```

Where:

file — the name of a file defined in the dictionary. This may be specified using standard prefix notation.

record — the name of a record defined for the named file. This may be specified using standard prefix notation. If this parameter is omitted, only the file specified will be listed.

Example:

```
L PAYMST *P
```

This example will list the compilation output for file "PAYMST" and all records with names beginning with "P".

-+*+-

1.4.8 LIST PROGRAM — TQLMON: lp

The LP command (list program) will display (at the terminal) the compilation output for a program. The listing may be interrupted by pressing the MSG-WAIT key at the terminal.

Syntax:

LP progname

Where:

progname the name of a program defined in the dictionary. This may be specified using standard prefix notation.

Example:

LP PAYINQ

-+*+-

1.4.9 CREATE SCREEN FORMATS — TQLMON: mcs

The MCS command will direct TQL to generate screen formats for the indicated display definitions in a named TQL program. The generated screen formats will have names as specified in the USING clause in the DISPLAY DIVISION of the TQL program.

The user may wish to later make enhancements to the machine generated screen formats by using the standard TIP/30 utility MSGDEF.

Syntax:

```
Mcs   program  [,display-name]
```

Where:

program — the name of a program defined in the dictionary. This may be specified using standard prefix notation.

display-name — the name of the display which describes the screen format(s) which are to be built. If this parameter is omitted, all screen formats will be built. This parameter may be specified using standard prefix notation.

Note that this name is NOT the name specified in the USING clause in the DISPLAY DIVISION.

Example:

```
M  PAYINQ  *P
```

This will build all message formaats for the displays that begin with the letter 'P' in the program 'PAYINQ'.

-+*+-

1.4.10 DEFINE NEW RECORD TQLMON: n

The N command (new record) allows the user to define a new record definition using the TQL Editor. The TQL Editor is described in detail in the following section.

When the N command is processed, TQLMON will call the TQL Editor to allow the user to complete the record definition process. The initial contents of the edit workspace includes a skeleton record format.

When the user ends the editor, TQLMON will automatically compile the record definition composed by the user.

Syntax:

```
N   [file]  [,record]
```

Where:

file name of the file which is associated with the record. If omitted, the user must include the appropriate "FOR" clause in the text that is subsequently prepared.

record The name of the record. If this parameter is given, the TQL Editor will automatically use it as the name of the 01 level item in the skeleton record layout.

Example:

```
N   PAYROLL,PAYREC
```

Error Conditions:
If errors occur during the compilation of the record, the user may return to editing the record definition by simply issuing the N command again (with the same parameters).

-+*+-

1.4.11 DEFINE NEW FILE — TQLMON: nf

The NF command (new file) will cause the TQL monitor to display a screen format which may be used to define the characteristics of a file. This method may be preferable to defining a file using keywords in a library element (as described in the previous section).

Syntax:

```
NF [filename]
```

Where:

filename The catalogued name of the file to be defined to TQL. If this parameter is provided TQLMON will copy it (as the first field) into the screen format which is to be displayed.

DEFINE NEW FILE

TQLMON will display the following screen format. The user is then able to fill in the information requested and press transmit (XMIT) to cause the file definition to be compiled. If the user does not press XMIT, (for example, presses MSG-WAIT), the new file definition process will be cancelled.

```
(TF$TQLFL)                    T Q L   File Definition

  File Name: ________      File Type: ________      Access: ____
 Block Size: ____          Delete flag: ____        Index Size: ____
I/O routine: ________      Record Format: ______    Record Size: _____

Key Information:
                        Location      Duplicates?    Changes?
                Length  rel to 0        (Y/N)          (Y/N)
                ~~~~~~  ~~~~~~~~      ~~~~~~~~~~     ~~~~~~~~
        Key 1:   ___     ____           -              -
        Key 2:   ___     ____           -              -
        Key 3:   ___     ____           -              -
        Key 4:   ___     ____           -              -
        Key 5:   ___     ____           -              -

                                Leave cursor here and press XMIT  (_)
```

Where:

The values that may be specified in the screen format are identical to those described in the previous section ("FILE DEFINITION").

-+*+-

1.4.12 DEFINE NEW PROGRAM — TQLMON: np

The NP command (new program) will cause the TQL monitor to call the TQL Editor to enable the user to enter the source for a new TQL program. This method may be used as an alternative to the standard TIP/30 text editor (QED). The TQL Editor will begin with a skeleton definition of a TQL program which may be modified by the user. When the TQL Editor is ended, the TQLMON program will automatically compile the program. The mechanics of the TQL Editor are described in the next section of this manual.

Syntax:

NP [progname]

Where:

progname The name of the new TQL program that is to be created. If this parameter is provided, it will appear as the PROGRAM-ID of the skeleton program that is used as the starting point for editing.

Example:

NP TESTPRO

Will create a skeleton program with PROGRAM-ID "TESTPRO".

Additional Considerations:

If errors are encountered when the program is compiled, the user may correct the errors merely by issuing the NP command again (with the appropriate program name). The TQL Editor will retrieve the source (as it was at compile time) and allow the user to resume editing.

-+*+-

1.4.13 PRINT FILE/RECORD — TQLMON: print

The PRINT command will generate a printed copy of the compilation output of a file or record definition. The printout may be routed to the site printer or an auxiliary print device. The printout is generated using TIPPRINT (the standard TIP/30 printing interface).

Syntax:

```
Print  file  [,record]  [,dest]
```

Where:

file the name of a file defined in the dictionary. This may be specified using standard prefix notation.

record the name of the record to be printed. This may be specified using standard prefix notation.

If record name is omitted, only the file compilation will be printed.

dest The desired destination of the printout. Default is PRNTR (the site printer). Other possibilities include: AUX1 etc.

Example:

```
P  PAYMST  *
```

This will print all record defninitions for the file PAYMST.

-+*+-

1.4.14 PRINT PROGRAM — TQLMON: pp

The PP command (print program) will generate a printed copy of the compilation output of a TQL program. The printout may be routed to the site printer or to an auxiliary printer. The printout is generated using the facilities of TIPPRINT (the standard TIP/30 printing interface).

Syntax:

```
PP   program   [,,dest]
```

Where:

program the name of a program defined in the dictionary. This may be specified using standard prefix notation.

dest The desired print destination. Default is PRNTR (the site printer). Other possibilities include: AUX1 etc.

Note that "dest" is the third parameter to the PP command. The second (omitted) parameter is reserved for future use and should normally be omitted.

Example:

```
PP  PAYINQ,,AUX1
```

-+*+-

PURGE PROTOTYPE FILE

1.4.15 PURGE PROTOTYPE FILE — TQLMON: purge

The PURGE command (purge prototype file) may be used to delete all records that are contained in a PROTOTYPE file. This command will only operate on PROTOTYPE files.

Syntax:

PURGE filename

Where:

filename The name of the prototype file that is to be purged.

Example:

PURGE TESTFILE

This delete all prototype record for the prototype file known as TESTFILE.

-+*+-

1.4.16 EDIT RECORD DEFINITION TQLMON: q

The Q command (QED record) allows the programmer to use the standard TIP/30 text editor (QED) to modify an existing record definition. TQLMON will call the text editor (QED) to enable editing of the record definition. When the user ends the interaction with QED with the QED "E" command, TQLMON will automatically recompile the record definition.

Syntax:

Q filename,record

Where:

filename name of the file with which the record is associated.

record name of the record which is to be edited.

Error Conditions:
If errors occur in the compilation of the record definition, the user may resume editing the record definition simply by issuing the Q command again (with the same parameters).

-+*+-

1.4.17 EDIT TQL PROGRAM — TQLMON: qp

The QP command (QED program) allows the programmer to use the standard TIP/30 text editor (QED) to modify the source of an existing TQL program. TQLMON will call the text editor (QED) to allow editing of the program source. When the user ends the interaction with QED with the QED "E" command, TQLMON will automatically recompile the TQL program.

Syntax:

QP progname

Where:

progname The name of the TQL program to be edited.

Error Conditions:
If errors occur in the compilation of the program, the programmer need only re-enter the QP command to resume editing the program source.

-+*+-

1.4.18 RUN PROGRAM

TQLMON: run, open

The Run command (run program) allows the programmer to execute a TQL program. This command is provided only to eliminate the need to end the TQL monitor and then use the OPEN transaction.

When the TQL program completes, control will return to the TQL monitor (TQLMON) program.

Syntax:

```
Run  program
OPEN  program
```

Where:

program The name of a TQL program defined in the dictionary.

Example:

```
R  PARTINQ
```

-+*+-

1.4.19 SUMMARIZE FILE/RECORD TQLMON: s

The Summarize command will display a list of existing file and/or record definitions that are presently in the TQL dictionary. File and/or record names may be selected by prefix. The listing may be interrupted by pressing MSG-WAIT.

Syntax:

```
S  file  [,record]
```

Where:

file the name of a file defined in the dictionary. This may be specified using standard prefix notation.

record the record name to be listed. This may be specified using standard prefix notation. If omitted, only file entries will be listed.

Example:

```
S  PAYMST *
```

Summarize all records for the file "PAYMST".

-+*+-

1.4.20 SUMMARIZE PROGRAMS — TQLMON: sp

The SP command (summarize programs) will display a list of program names that are presently in the TQL dictionary. The listing may be interrupted by pressing MSG-WAIT.

Syntax:

SP

Where:
There are no parameters to this command.

Example:

SP

Additional Considerations:

The programs that are in the dictionary are listed (along with their program description). The list is in the order that the programs were entered into the dictionary (chronological).

-+*+-

1.4.21 UPDATE RECORD DEFINITION TQLMON: u

The U command (update record) allows the user to change a record definition using the TQL Editor. The TQL Editor is described in detail in the following section.

When the U command is processed, TQLMON will call the TQL Editor to allow the user to modify the record definition. When the user ends the editor, TQLMON will automatically compile the modified record definition.

Syntax:

```
U   file,record
```

Where:

file Name of the file which is associated with the record.

record Name of the record.

Example:

```
N   PAYROLL,PAYREC
```

-+*+-

1.4.22 UPDATE CONTROL HEADER — TQLMON: uc

This command is used to maintain the control record in the TQL dictionary. The control record contains (an optional) READ and/or WRITE password to control dictionary access and a list of TIP/30 user-ids. The user-ids in the list are authorized to make changes to the contents of the TQL dictionary (ie: compile, delete etc).

Syntax:

UC

Where:
no parameters.

Example:

UC

UPDATE CONTROL HEADER

TQLMON will display the screen format shown below. The user may alter the list of user-ids in the fields provided and press XMIT to effect the change. Pressing MSG-WAIT will abort the "UC" command with no changes to the control record.

```
================================================================================
TIP/30      TQL DEFINITION FILE CONTROL RECORD                          (TF$QLC)
================================================================================

            Free Chain: ______ -->______
            File Chain: ______ -->______
         Session Chain: ______ -->______
       Directory Chain: ______ -->______
  Next Available Block: ______

Control Record Created: __/__/__ at __:__:__    Read Password [________]
                                                Update Password [________]

Users:
```

-+*+-

1.4.23 UPDATE FILE DEFINITION — TQLMON: uf

The UF command (update file) will cause the TQL monitor to display a screen format which may be used to change the definition of a file. This method may be preferable to defining a file using keywords in a library element (as described in the previous section).

Syntax:

UF filename

Where:

filename The name of the file which is to be changed.

UPDATE FILE DEFINITION

TQLMON will display the following screen format. The user is then able to alter the information shown and press transmit (XMIT) to cause the file definition to be compiled. If the user does not press XMIT, (for example, presses MSG-WAIT), the file change process will be cancelled.

```
(TF$TQLFL)                    T Q L   File Definition

  File Name:  ________        File Type:  ________        Access:  ____
  Block Size:  _____          Delete flag:  _____         Index Size:  ____
  I/O routine:  ________      Record Format:  ______      Record Size:  _____

Key Information:
                          Location      Duplicates?     Changes?
                Length    rel to 0        (Y/N)           (Y/N)
                ~~~~~~    ~~~~~~~~    ~~~~~~~~~~~     ~~~~~~~~
        Key 1:   ___       ____           -               -
        Key 2:   ___       ____           -               -
        Key 3:   ___       ____           -               -
        Key 4:   ___       ____           -               -
        Key 5:   ___       ____           -               -

                                  Leave cursor here and press XMIT  (_)
```

Where:

The values that may be specified in the screen format are identical to those described in the previous section ("FILE DEFINITION").

-+*+-

1.4.24 UPDATE PROGRAM — TQLMON: up

The UP command (update program) will cause the TQL monitor to call the TQL Editor to enable the user to modify the source for an existing TQL program. This method may be used as an alternative to the standard TIP/30 text editor (QED). When the TQL Editor is ended, the TQLMON program will compile the program. The mechanics of the TQL Editor are described in the next section of this manual.

Syntax:

UP progname

Where:

progname Name of the TQL program that is to be updated.

Example:

UP TESTPRO

-+*+-

1.4.25 WRITE FILE/RECORD — TQLMON: write

The WRITE command directs TQL to write the source (which is stored in the TQL dictionary) for a file or record to an OS/3 library element. This function may be performed as part of a backup scheme or to facilitate transporting TQL file or record definitions to other sites.

Syntax:

```
Write  file  [,record]  lib   elt
```

Where:

file the name of a file defined in the dictionary.

record the name of a record defined for the specified file. If omitted, only the file definition will be written.

lib the output library file name.

elt the output library element name.

Example:

```
W  PAYMST,PAYREC,SOURCE,SVREC
```

This example will write the source for record "PAYREC" of file "PAYMST" from the TQL dictionary to the library/element: SOURCE/SVREC

-+*+-

1.4.26 WRITE PROGRAM TO LIBRARY TQLMON: wp

This command will copy the source statements of a TQL program to a specified OS/3 library element. This function may be done as part of a backup scheme or to facilitate transporting TQL programs to other sites.

Syntax:

WP program lib elt

Where:

program the name of a program defined in the dictionary.

lib the output library file name.

elt the output library element name.

Example:

WP PAYINQ SOURCE SVINQ

This example will copy the source for the TQL program "PAYINQ" to the library element SOURCE/SVINQ.

-+*+-

1.5 THE TQL TEXT EDITOR TQLEDT

The TQL editor is a screen format oriented editor which has been designed specifically for use with TQL. TQLEDT will display a full screen of text (approx 17 lines). The user may directly alter the text which is displayed or may enter commands to display other portions of text or to move or copy text.

The command repertoire of TQLEDT is less extensive than the standard TIP/30 text editor, but is sufficient for TQL editing requirements.

There are no search or substite commands in the TQL editor. Searching is accomplished by using the "Forward Page" and the "Backward Page" function keys. Substitution is merely a matter of entering the desired text in place of the original text.

Lines in the edit work area are displayed with line numbers that are used as reference points by the various commands.

TQL Editor Commands

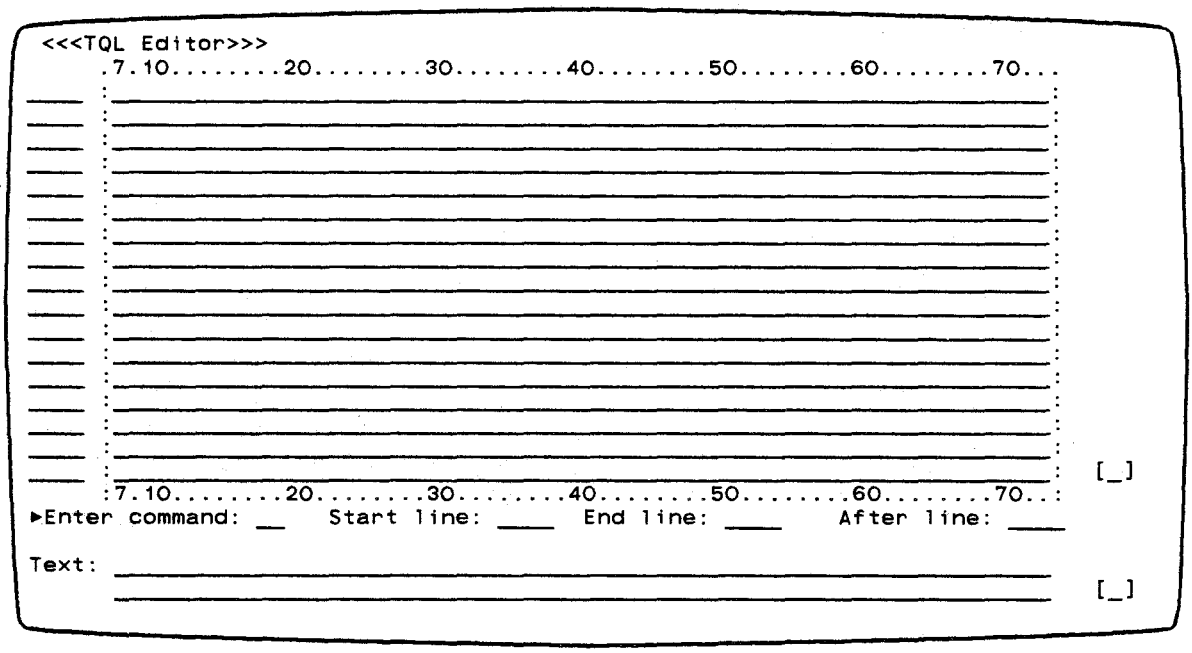
```
ADd       - ADD lines
COpy      - COPY lines
DElete    - DELETE lines
ENd       - END editing (and cause automatic compilation by TQLMON)
HElp      - HELP please!
MOve      - MOVE lines
PRint     - PRINT (display) lines on screen
QUit      - QUIT editing (and suppress automatic compilation)

F1/F5     - refresh screen display
F2/F6     - display next screen    (Forward Page)
F3/F7     - display previous screen  (Backward Page)

MSGWAIT - same as QUIT command
```

TQLEDT (initially) displays the screen format shown below (the lines will contain the first 17 lines of text). The user may enter explicit commands or simply modify the text that is displayed and press XMIT from the first cursor resting location.

Note that there is a field for entering a command, a starting line number, ending line number, or an "after" line number. Each command has specific requirements which are described in the following sections.



1.5.1 ADD LINES TQLEDT: ad

The ADd command allows the user to add new lines of text after a specific line number. The user should enter "AD" as the command and specify an "after" line number.

If the text to be added is two lines or less the user may enter them directly in the "Text" fields of the screen format and press transmit from the second cursor resting location.

If more than two lines are to be added, the user should leave the "Text" fields blank and simply specify the "after" line number. TQLEDT will respond by re-displaying the screen with the first line containing the contents of the specified "after" line (protected). The user may then enter any desired text below the first line and press transmit at the first cursor resting location.

Trailing lines which are entirely blank will not be added.

The "start line" and "end line" fields are ignored by the ADD command.

Pressing MSG-WAIT while in ADD mode will cancel the ADD command.

-+*+-

1.5.2 COPY LINES TQLEDT: co

The COpy command allows the user to copy a range of lines from one part of the edit work area to a point which is "after" another line.

The user should enter "CO" as the command and provide the starting line and ending line to be copied as well as the number of the line which is ahead of the desired location of the copied text.

For example, to copy lines 1 through 8 after line 17, the user would specify the command as "CO", the starting line as "1", the ending line as "8" and the after line as "17". TQLEDT will copy the lines after line 17 and ahead of the line which was line 18. The lines originally at lines 1 through 8 would remain unchanged.

-+*+-

1.5.3 DELETE LINES TQLEDT: de

The DElete command allows the user to delete a range of lines. As a precaution against fumble-finger syndrome, TQLEDT will not allow the deleting of lines that are not currently displayed in the upper portion of the display.

The user must enter "DE" as the command, the starting line number and the ending line number. The TQL Editor will delete the lines from the starting line number to the ending line number INCLUSIVELY.

If the range of lines specified is not contained entirely within the lines displayed in the upper portion of the display an error message will be displayed and the delete request will not be honoured.

-+*+-

1.5.4 END TQL EDITOR TQLEDT: en

The ENd command signals the TQL Editor that the user has completed all desired editing. The TQL Editor will terminate normally. The program which called the TQL Editor (normally the TQL Monitor - TQLMON) will (by default) immediately begin compiling the contents of the edit workspace. (See previous section on the TQL Monitor - in particular the description of the N, NP, U, UP commands).

If errors occur during that compilation process, the user can simply re-enter the appropriate TQLMON command and correct the errors.

-+*+-

1.5.5 HELP FOR TQL EDITOR — TQLEDT: he

The HElp command will display a screen containing the current help information for the TQL Editor. The following screen format is representative of what is displayed:

```
            T I P / 3 0   Q u e r y   L a n g u a g e   E d i t o r


Command         Function
=======         ============================================================
 AD             add text after 'Start line'
 CO             copy 'Start line' thru 'End line' after 'To line'.
 DE             delete 'Start line' thru 'End line'.
 EN             end this editor. (Module will then be compiled)
 MO             move 'Start line' thru 'End line' after 'To line'
 PR             display from 'Start line'
 QU             quit this editor. (Module will not be compiled)
 RE             read an element in at 'Start line' (default is last line)
 WR             write 'Start line' thru 'End line' to an element


Press XMIT to continue: _
```

-+*+-

1.5.6 MOVE LINES TQLEDT: mo

The MOve command is identical to the COPY command (see previous section) with the exception that the moved lines are NOT left in their previous location.

The move command requires the starting, ending and after line numbers be specified.

-+*+-

1.5.7 PRINT (DISPLAY) LINES TQLEDT: pr

The PRint command will display a range of line numbers in the upper portion of the screen. The user must enter the "PR" command in the command field and then must specify a starting line number.

If an ending line number is not specified the TQL editor will display as many lines as possible (to a maximum of 17) starting with the specified starting line number.

-+*+-

1.5.8 QUIT TQL EDITOR

TQLEDT: qu

The QUit command causes the TQL editor to abort the editing session. The calling program (normally TQLMON) will NOT attempt an automatic compilation of the contents of the edit workspace. The contents of the work space will be lost.

If changes had been made to the contents of the workspace, the TQL editor will warn the user and ask for confirmation of the QUIT command.

The QUIT command is normally used only if the contents have been damaged by a poor choice of previous commands.

-+*+-

1.5.9 TQL EDITOR FUNCTION KEYS — TQLEDT: fkeys

The TQL Editor recognizes certain function keys as special commands. Invalid function keys will result in an error message displayed on the screen.

F1 / F5 Function key 1 (or 5) causes TQLEDT to resend the last output screen. This can be necessary if the screen display was altered unintentionally or by the recpetion of an unsolicited message.

F2 / F6 Function key 2 (or 6) is the "Forward Page" key. When this function key is pressed TQLEDT will display the next set of source lines. This is equivalent to advancing the display by 17 lines.

F3 / F7 Function key 3 (or 7) is the "Backward Page" key. When this function key is pressed TQLEDT will display the previous set of source lines. This is equivalent to displaying the previous 17 lines.

MSG-WAIT Pressing MSG-WAIT will signal the TQLEDT program that the user wishes to abort the edit session (equivalent to the QUIT command). If changes had been made, the user will receive a warning and be given a chance to reconsider.

-+*+-

1.6 RUNNING A TQL PROGRAM TQL: open

The supplied transaction code "OPEN" is used to begin execution of a TQL program. The TQL user may choose to OPEN a particular (TQL) program or may choose to view a menu of available TQL programs and make a selection from the menu. In either case, once a particular program has been selected, TQL will display a standard TQL command screen.

The commands available include capabilities to:

- request the generation of a pre-defined report
- request the display of data using a pre-defined display format
- list selected fields (at the terminal using a free-format display)
- print selected fields (at the site printer or an auxiliary printer)
- display or report data according to constraints (IF field = ...)

The user may select certain subsets of the available data by including in his command certain conditions that must be met before data is to be displayed.

To specify the conditions the user would normally use the "IF" statement. The "IF" statement contains field comparisons and/or other constraints that the data must meet before being included in a particular display.

The following sections describe the initial execution of a TQL program and the various commands that are available to the TQL user.

1.6.1 TQL PROGRAM EXECUTION — TQL: open

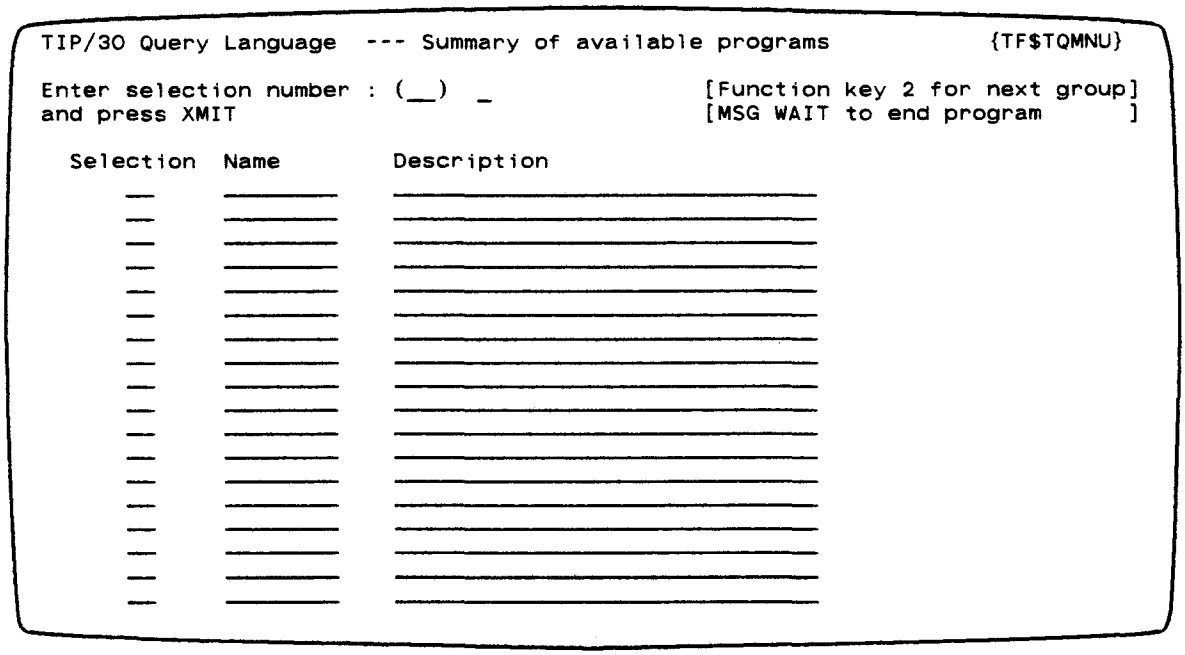
The OPEN transaction causes the TQL interpreter to "execute" a TQL program. All TQL programs operate interactively to allow the user to enter commands which are processed by the TQL program.

Syntax:

OPEN [progname [,initial command]]

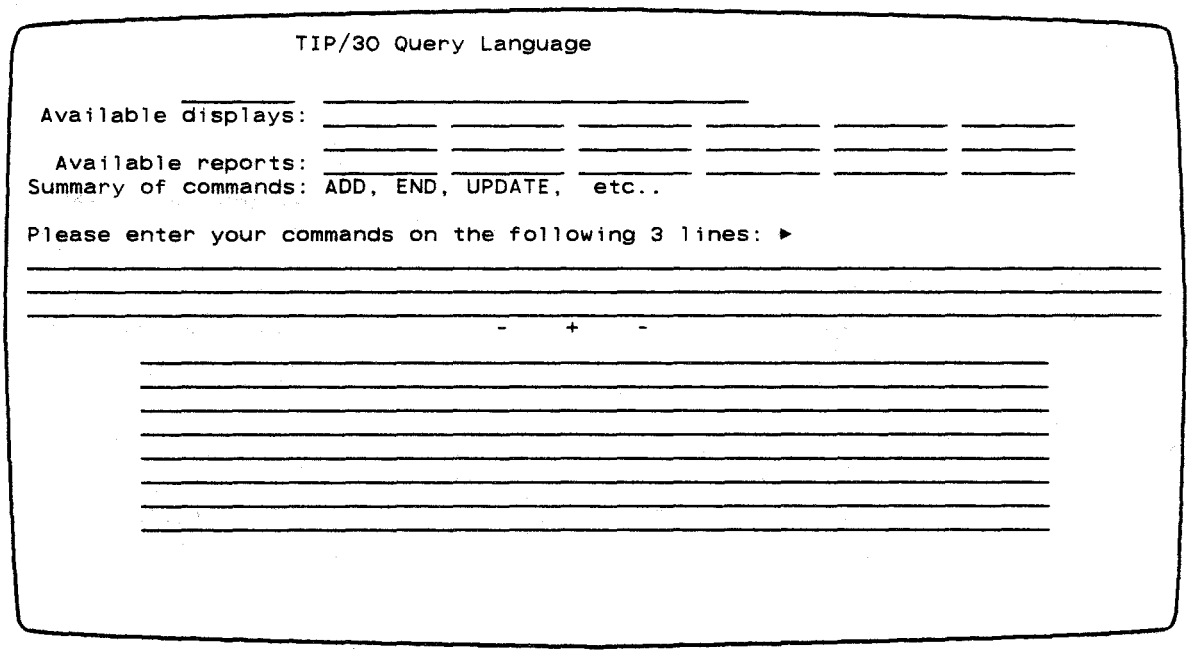
Where:

progname Name of the TQL program to run. If this program name is not specified, TQL will react by displaying a menu of available programs. The user may enter the selection number desired and press transmit (XMIT) or press MSG-WAIT to terminate the OPEN transaction.



command This optional initial command may be specified if the user wishes to execute ONLY this one command. The inclusion of this initial command merely bypasses the display of the standard TQL command screen (since the command is already known). When this command is completed (successfully) the OPEN transaction will terminate normally.

If the program selection is valid the following menu screen is displayed. This screen format is used to enter all interactive TQL commands.



-+*+-

1.6.2 PREDEFINED DATA DISPLAY — TQL: display

The TQL user may request that data be displayed according to a pre-defined display format. Each pre-defined display format has a name which was assigned by the programmer. The display format effectively describes which fields will be displayed and the visual format of the display. To request a particular display, the user must enter a command of the following format.

Syntax:

```
display-name  [ IF expr    ]
              [ BY field   ]
              [ FROM key   ]
              [ TO key     ]
              [ SUM field field ... ]
```

Where:

displayname The name of the desired pre-defined display. A list of available display names that have been programmed is given at the top of the TQL command screen.

IF expr The IF clause may be included to qualify the data to be displayed. The expression which follows the word "IF" may include field comparisons (Eg: IF PRICE > 500) and/or computations (Eg: PRICE * QUANTITY < 100000).

The complete description of TQL expressions is described in section 1.3.1.

BY field Indicates that the display is to be produced in ascending order by the specified field name. The field name specified must be defined as a "key" for the file that is being accessed.

If this clause is included, it must precede any use of the "FROM" or "TO" clauses.

FROM key Indicates that the display is to begin with the first record which has a key greater than or equal to the key given.

TO key Indicates that the display is not to go beyond records which have a key greater than or equal to the specified key.

SUM field Specifies one (or more) fields which are to be summed by TQL. At the end of the display (or upon returning prematurely to the TQL command screen) the total, average and count of each specified field will be shown. Fields specified must be numeric fields.

Example:

```
DISP1 IF INVOICE-TOTAL > 5000  SUM INVOICE-AMT  UNIT-PRICE
```

This example would request a pre-defined display named "DISP1". The display would only be produced if the field "INVOICE-TOTAL" has a value greater than 5,000. At the end of displaying all data, the total and average of both "INVOICE-AMT" and "UNIT-PRICE" will be displayed. The count of the number of items that was used to compute the average will also be shown. Note that the total of UNIT-PRICE might be rather meaningless in practice, but the average may be very interesting.

-+*+-

1.6.3 ADD RECORD TQL: add

The ADD command allows the user to place new records in the file. It will display an empty screen which must be filled in and transmitted to the program.

Syntax:

ADD <display-name>

Where:

display-name name of display statment in program.

Additional Considerations:

The screen is displayed with no initial data; the user must enter the appropriate data and press XMIT.

When TQL receives the data it will verify it according to any VERIFY clauses and ON ADD/ON WRITE clauses. If errors are detected the terminal operator will be notified. The terminal operator should correct the data in error and try again.

-+*+-

1.6.4 COUNT RECORDS

TQL: count

The COUNT command will count records based on the <expr>, starting position, and ending position in the file.

Syntax:

```
COUNT record-name [IF <expr>]
      [BY keyn] [FROM key] [TO key]
      [SUM field]
```

Where:
Parameters are the same as previously described.

Example:

```
COUNT PAY-REC IF TIMES-RUN > 5
      SUM BASIC-CHRG TOTAL-CHRG
```

-+*+-

DELETE RECORD

1.6.5 DELETE RECORD TQL: delete

The DELETE command will display the selected record. The informational message "Press F2 to delete record" will also appear on the screen. The terminal operator should verify that the displayed record is indeed the record to be deleted. To delete the displayed record Function key 2 (F2) must be pressed. If any other key is pressed, TQL will NOT delete the displayed record and return the user to the main prompt screen.

Syntax:

```
DELETE display-name  key
```

Where:
Same parameters as before.

Example:

```
DELETE CUST 'AEI00020'
```

-+*+-

1.6.6 ENTER RECORDS

TQL: enter

The ENTER command is equivalent to multiple uses of the ADD command. TQL will repeatedly display the specified display-name so that the terminal operator may enter new records to the file. To terminate the ENTER command, the terminal operator must press MSG-WAIT. This causes TQL to return to the main prompt screen. The message "record not added" signals that the ENTER operation has been completed (the last screen - at the time of MSG-WAIT - was not added to the file).

Syntax:

ENTER <display-name>

Where:

display-name name of display statment in program.

Additional Considerations:

The screen is displayed with no initial data. The user must enter the data and press transmit.

When TQL receives the data it will verify it according to any VERIFY clauses and ON ADD/ON WRITE coding. If errors are detected the terminal operator will be notified. The terminal operator should correct the data in error and try again.

-+*+-

END SESSION

1.6.7 END SESSION TQL: end/close

The END command terminates the current session. The optional 'CLOSE' command is equivalent.

Syntax:

```
CLOSE
END
```

-+*+-

1.6.8 TQL HELP

TQL: help

The HELP command will display a screen with a summary of available commands.

Syntax:

HELP <session-name>

Where:

The following screen is displayed.

```
                TIP/30 Query Language ... run time commands
                ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
display-name [IF expression] [FROM key] [TO key] [BY field]
             [SUM fields]

ADD    display-name            : add new record
DELETE display-name            : delete record
END                            : end program
HELP                           : display this screen
OPEN   program-name            : open new TQL program
PRINT  report-name [ON file]   : print report
SHOW   display-name            : display field names used
UPDATE display-name key        : update a record
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
   Current program: ________   ____________________________
Available displays: ________ ________ ________ ________ ________ ________
                    ________ ________ ________ ________ ________ ________
 Available reports: ________ ________ ________ ________ ________ ________

  Enter your command and press XMIT ►
```

-+*+-

1.6.9 FREE FORMAT LIST — TQL: list

The LIST command may be used to produce an ad hoc display selecting records based on the <expr>, starting position, and ending position in the file. The selected records are displayed in groups of 4 on the terminal. The listing will be truncated (if necessary) at 80 columns.

Syntax:

```
LIST (field-names) [IF <expr>]
       [BY keyn] [FROM key] [TO key]
       [SUM field]
```

Where:

(field-names) a list of field names enclosed in parentheses and separated by either commas or spaces. The first field name is used to determine which record is to be read.

Example:

```
LIST (CUST-NAME CUST-DATE CUST-DUE)
            IF TIMES-RUN = 0
```

-+*+-

1.6.10 DISPLAY NEXT SCREENFULL

TQL: next

The NEXT command continues displaying records from the file from the last record displayed. Function key 2 (F2) equivalent to the NEXT command.

Syntax:

NEXT

-+*+-

OPEN NEW PROGRAM

1.6.11 OPEN NEW PROGRAM — TQL: open

The OPEN command will END the current TQL program and execute the TQL program specified as a parameter to the OPEN command.

Syntax:

OPEN <program-name>

Where:

program-name The name of the TQL program to execute.

-+*+-

1.6.12 PRINT A REPORT — TQL: print

The PRINT command will generate the pre-defined report. Data may be selected based on the <expr>, starting position, and ending position position in the file. This command will continue to print all records from the file which satisfy the <expr>. The user may overide the default print destination by using the "ON" clause.

Syntax:

```
PRINT  report-name  [IF <expr>]
       [BY keyn]  [FROM key]  [TO key]  [ON file]
```

Where:
Parameters are the same as previously described.

ON file a valid printer file name. This may be the name of a printer generated into the TIP/30 system, PRNTR or AUX1.

Example:

```
PRINT REC-A IF TIMES-RUN = 0 ON AUX1
```

-+*+-

1.6.13 FREE FORMAT PRINT — TQL: print

The PRINT command may also be used to produce an ad hoc report selecting records based on the <expr>, starting position, and ending position in the file. The user specifies which fields are to be printed (instead of specifying a pre-defined report name). The report will continue with all records that satisfy the given expression. The destination of the printed report may be changed (from the default of the site printer) by using the "ON" clause.

Syntax:

```
PRINT (field-names) [IF <expr>]
       [BY keyn] [FROM key] [TO key] [ON file]
       [SUM field]
```

Where:

(field-names) A list of field names enclosed in parentheses and separated by either commas or spaces. The printed fields must be able to fit on one print line (132 characters) otherwise truncation will occur. The first field named is used to determine which record is read.

ON file a valid printer file name. This may be the name of a printer generated into the TIP/30 system, PRNTR or AUX1.

Example:

```
PRINT (CUST-NAME CUST-DUE CUST-DATE)
            IF TIMES-RUN = 0 ON AUX1
```

-+*+-

1.6.14 SHOW FIELD NAMES

TQL: show

The SHOW command allows the user to get a list of all field names in a given display.

Syntax:

SHOW display-name

Where:

display-name The name of a pre-defined display. The names of the fields that are referenced by this display will be shown. Field names that have the suffix ":9" are numeric fields (the ":9" is not part of the field name - merely a notation).

-+*+-

UPDATE RECORD

1.6.15 UPDATE RECORD TQL: update

The UPDATE command will display the record selected. The user may then update the information on the screen and transmit. The updated record is then written to the file and the main prompt screen is displayed for the next command.

Syntax:

UPDATE <display-name> [<from-clause>] [IF <expression>]

Where:
Same parameters as before.

Example:

```
UPDATE CUST IF AMOUNT-DUE > 5000
UPDATE CUST FROM 'AEI00020'
UPDATE CUST 'AEI00020'
```

Additional Considerations:

Function key 4 (F4) may be pressed after a record is displayed. This will re-display the same record for update.

If you decide not to proceed with the update press MSG-WAIT to cancel the update.

-+*+-

1.6.16 USE OF FUNCTION KEYS

TQL: fn keys

F1, F5 will re-display the current screen.

F2, F6 will proceed to the next screen of data.

F3 will return to the menu screen

F4 will prepare to update the last displayed record.

F9 get more detail records.

-+*+-

1.7 CALLING TQL FROM TIP PROGRAM — TQL: call tql

A TQL program may be invoked by another transaction program by using the TIPSUB or TIPXCTL subroutine to transfer control to the transaction code "TQL".

TQL expects the CDA to contain the following information (the structure of the CDA is assumed to be that described by the copy book "TC-CDA" in library "TIP" - refer to the description of this layout in the PCS section of this manual):

CDA-PARAM (1) The name of the TQL program to execute

CDA-TEXT (optional) single initial TQL command

Eg: REP1 IF PART-NO = '123XV'

Example:

```
MOVE 'TQL'        TO PIB-TRID.

MOVE SPACES       TO CDA.

MOVE 'PARTINQ'    TO CDA-PARAM (1).

CALL 'TIPSUB'.

IF NOT PIB-GOOD
   GO TO ERROR-CALLING-TQL.
```

1.8 RESERVED WORDS TQL: words

The folowing words have reserved meaning to TQL and may not be used as data field names.

ACCEPT	ADD	ADD	ALL
ALLOW	AND	AT	
BY			
CALL	CASE	CASEOF	CHANGE
CLOSE	COMPUTE	COPY	COUNT
DATA	DATE	DAY	DEFAULT
DELETE	DISPLAY	DIVIDE	DIVISION
DECLARATIVES			
ELSE	END	EQUAL	EQUALS
ENTER	ERROR	EXIT	
FILE	FOR	FROM	
GIVING	GO	GOTO	GREATER
HIGH-VALUE	HIGH-VALUES	HOME$	
ID	IDENTIFICATION	IDENTIFIER	IF
IN	INSPECT	INTO	INVOKE
IS			
KEY			
LESS	LIST	LOW-VALUE	LOW-VALUES
MINUS	MOVE	MULTIPLY	MUST
MORE$			

RESERVED WORDS

NEGATIVE	NEXT	NL$	NO
NOT	NUMERIC		
OF	ON	OPEN	OR
PERFORM	PLUS	POSITIVE	PROCEDURE
PRINT			
QUOTE	QUOTES		
RANGE	READ	RECORD	REMAINDER
REPLACING	REPORT	RETURN	ROUNDED
SEARCH	SECTION	SECURITY	SELECT
SENTENCE	SET	SHOW	SKIP$
SPACE	SPACES	SUBTRACT	SUM
TAB$	THAN	THEN	THROUGH
THRU	TIME	TIMES	TO
UNTIL	UP	UPON	USING
VARYING	VERIFY	VIA	
WHEN	WHERE	WRITE	
ZERO	ZEROES	ZEROS	

1.9 INITIALIZING TQL DICTIONARY TQLINT

This section describes the maintenance of the TQL dictionary (control) file. It is of interest primarily to the systems programmer.

The dictionary file is known by the logical file name TQL$CTL. It is a direct access file. It may be created by the user as DAM, or direct MIRAM (DMIRAM). It should be allocated at least 5 cylinders with a three cylinder increment. This file is generated into TIP/30 as follows.

```
FILE TQL$CTL,DMIRAM BLKSIZE=512 RECSIZE=512 HOLD=UP.
```

The file is formatted using an online program called TQLINT. This program should be catalogued for master use only. It will prompt for a password to control access to the file. If a password is supplied, the password is then required whenever the dictionary file is updated (compilation etc...).

The user who initializes the control file (by executing the transaction TQLINT) is entered as the only user who may update the control file. If other users (programmers) are to be allowed to modify the control file, then the TQL monitor program (TQLMON) "UC" command may be used by the initializing user to add other user-ids to the list of those authorized to modify the control file.

If TQL is to be heavily used you may wish to make the file resident to reduce TIP/30 swapping load.

One technique for testing new TQL programs is to have a second control file catalogued in the programming group:

In TIPGEN:

```
FILE TQL$TST,DMIRAM BLKSIZE=512 RECSIZE=512 HOLD=UP.
```

In TIP Catalogue:

```
FILE TQL$CTL LFD=TQL$TST SECUR=PROG GROUP=EDP.
```

1.10 LISTING THE TQL DICTIONARY FILE QB$LST

The TQL dictionary file (TQL$CTL) may be listed by a batch utility program supplied by Allinson-Ross. This batch program is capable of listing record definitions, file definitions, and TQL programs.

The program accepts control stream options that may be used to select programs by group affiliation or records by file membership.

If no control stream options are specified the program assumes that the entire TQL dictionary is to be listed.

If control stream options are given dictionary information will be listed only for the specified group(s) or file name(s).

Syntax:

```
//      JOB   TQLPRINT,,C000
//      TIPFILES
//      LBL   TIP$TQL
//      DD    ACCESS=SRD
//      LFD   TQL$CTL
//      EXEC  QB$LST,TIP
/$
        GROUPS=grp(s)
        FILES=file(s)
/*
/&
```

Where:

grp(s) A list of group names (separated by a comma). If this keyword is used only programs which may be accessed by one of these groups will be listed.

Standard prefix notation may be employed to specify a group name. Eg: *P means all groups with names starting with "P".

file(s) A list of file names (separated by a comma). If this keyword is used record and file definitions will be listed only for the files specified.

Standard prefix notation may be employed to specify a file name.

Example:

```
GROUPS=ARC,EDP
```

This group specification directs the QB$LST program to list only programs that are available to groups "ARC" and/or "EDP".

1.11 REORGANIZING THE TQL DICTIONARY FILE QB$DMP

The TQL dictionary file (TQL$CTL) may be reorganized by using the supplied utility program QB$DMP. This program can "dump" the TQL dictionary file to either tape or disk and perform the corresponding "restore" operation. The "restore" operation will have the effect of condensing the TQL dictionary file.

For a DUMP operation, the program requires an output file with an LFD name of "TQL$DMP". This file may be either tape or disk (sequential or non-indexed MIRAM).

For a RESTORE operation, the file "TQL$DMP" (that was produced by a prior DUMP operation) becomes the input.

The type of operation (DUMP or RESTORE) and the type of device used for the TQL$DMP file must be specified by control stream parameters.

Syntax:

```
//        JOB        TQLREORG,,10000
//        GBL        TYPE,MEDIUM
//        TIPFILES
//        LBL        TQL$CTL
//        DD         ACCESS=EXCR
//        LFD        TQL$CTL
//        DVC        90
//        VOL        ??????
//        LBL        TQL$DMP
//        LFD        TQL$DMP
//        OPTION     SCAN,SUB
//        EXEC       QB$DMP,TIP
/$
          TYPE=&TYPE
          MEDIUM=&MEDIUM
/*
/&
```

Where:

&type The type of operation to be performed. Choose either "DUMP" or "RESTORE". Eg: TYPE=DUMP.

This keyword is required.

&medium The device type of the TQL$DMP file. Choose either "TAPE" or "DISK". Eg: MEDIUM=TAPE.

This keyword is required.

Additional Considerations:

The job log for the program gives the count of each type of dictionary record that has been processed. There may be discrepancies between the number input and output for a given type of record (on a DUMP operation). Unless there are explicit error messages, such discrepancies are not a problem. The TQL$DMP file is a sequential file.

1.12 TQL PROTOTYPING TQL$PRO

File prototyping is a feature provided by TQL to facilitate the design of new applications. TQL allows the programmer to define files with a type "PROTOTYPE". Such a file is mapped into a real file, known by logical file name TQL$PRO. One TQL$PRO file may contain many logical prototype files. All data is stored in an internal format. This allows the programmer to change the size or position of data fields without affecting the logical file or the data already stored in a logical file.

Once the file design is complete the programmer may then allocate disk space for the file, define the file to TIPGEN, and update the TIP/30 JCL for the new file(s). Then update the file definition in TQL changing the type to MIRAM or ISAM as appropriate. Now re-compile any TQL programs developed using the COMP command of TQLMON. Your existing TQL program will now work on the real file.

There is no interface between user written COBOL programs and TQL prototype files.

To remove data from a prototype file there is a command PURGE available in TQLMON. Enter 'PURGE filename'. You will be prompted if it is OK to proceed. When you reply YES all data for the named prototype file is deleted from TQL$PRO. Note that you should periodically un-load and re-load TQL$PRO to recover space from deleted records.

Just as one TQL$PRO file may contain many logical prototype files, so your system may have several TQL$PRO files by taking advantage of the TIP/30 catalogue.

The TQL$PRO file must be generated in your TIP system as follows:

```
FILE  TQL$PRO,MIRAM BLKSIZE=2048   RECSIZE=2047   JOURNAL=NO
                    INDSIZE=768    DELETE=RCB     HOLD=UP
                    KEY1=(49,001,NDUP,NCHG)
                    KEY2=(49,050,DUP,CHG)
                    KEY3=(49,099,DUP,CHG)
                    KEY4=(49,148,DUP,CHG)
                    KEY5=(49,197,DUP,CHG).
```

1.13 TQL EXAMPLE TQL Example

This section illustrates many of the features of TQL. The example shows the file and record definitions for a simple inventory file and associated order file. The programs illustrated provide the capability to maintain the inventory file (INV) and the order file (ORD) and to enter orders, change orders, display orders, print orders etc, while keeping track of inventory.

The inventory file has a logical file name of "INV" in the TIP/30 catalogue and has the following characteristics (obtainable from the TIP/30 generation parameters):

```
      FILE INV,MIRAM BLKSIZE=500
                     RECSIZE=50
[1]                  KEY1=(4,0,NDUP,NCHG)
                     KEY2=(16,4,DUP,CHG)
                     KEY3=(2,20,DUP,CHG)
                     ACCESS=EXCR.

      RECORD  INVREC
      01  INVREC.
[2]       05  INV-PART          PICTURE 9(4).
          05  INV-DESC          PICTURE X(16).
          05  INV-LOC           PICTURE 99.
          05  INV-QOH           PICTURE 9(5).
          05  INV-PRICE         PICTURE 9(5)V99.

      ALLOW CHANGE ALL.
      ALLOW ADD.
      ALLOW DELETE.
```

[1] The primary key of a MIRAM file must not allow duplicates or changes (TIP/30 restriction).

[2] The primary key is the inventory part number.

TQL EXAMPLE

This example system also makes use of an order file (logical file name "ORD") which has the following characteristics:

```
FILE ORD,MIRAM BLKSIZE=1000
               RECSIZE=100
               KEY1=(16,0,NDUP,NCHG)
               ACCESS=EXCR.
```

The order file contains two types of records. The first type is a header record (one per order). The second type is a detail record (one or more per order - representing items ordered):

```
RECORD  ORDHDR
01  ORDHDR.
    05  HDR-KEY.
       10  HDR-ORD.
          15  HDR-CUST  PICTURE X(8).
          15  HDR-NUM   PICTURE 9(4).
       10  HDR-LINE     PICTURE 9(4).
    05  HDR-PO-NUM      PICTURE X(8).
    05  HDR-LAST-LINE   PICTURE 9(4).
[1] ID IS HDR-LINE = 0.
    ALLOW CHANGE ALL.
    ALLOW ADD.
    ALLOW DELETE.
```

[1] A header record is distinguished by the field HDR-LINE equal to zero.

TQL EXAMPLE

```
        RECORD  ORDDTL
        01  ORDDTL.
            05  ORD-KEY.
                10 ORD-CUST      PICTURE X(8).
                10 ORD-NUM       PICTURE 9(4).
                10 ORD-LINE      PICTURE 9(4).
            05  ORD-PART         PICTURE 9(4).
            05  ORD-QTY          PICTURE 9(4).
[1]     ID IS ORD-LINE > 0.
        MUST ADD ORD-QTY.
        ALLOW CHANGE ALL.
        ALLOW ADD.
        ALLOW DELETE.
```

[1] A detail order record (representing on item ordered) is distinguished by the field ORD-LINE greater than 0. The field is incremented by one for each item in the order (items 1 through last item).

TQL EXAMPLE

The following TQL program was written to provide maintenance capabilities for the inventory file. Two pre-defined displays are defined by the program:

- "PART " - display (all fields) in a single inventory (part) record

- "PARTS" - display (all fields) in 5 inventory records.

The screen formats "DEMOINV1" and "DEMOINV2" are shown following the program source.

```
IDENTIFICATION DIVISION.
PROGRAM-ID.  INV          'INVENTORY UPDATE'.

DATA DIVISION.

FILE   INV.
RECORD INVREC.

DISPLAY DIVISION.

PART:  READ INVREC
        INVREC
       USING DEMOINV1.

PARTS: 5 ( READ INVREC
           INVREC)
         USING DEMOINV2.
```

TQL EXAMPLE

```
DEMOINV1
 INV-PART  INV-DESC          INV-LOC  INV-QOH  INV-PRICE
     ____  ____________________    __   _____   _____.__ _


F1/5:Refresh screen  F2/6:Next screen  F4:Update  Msg-wait:Menu
```

TQL EXAMPLE

```
DEMOINV2
 INV-PART  INV-DESC          INV-LOC  INV-QOH  INV-PRICE
     ____  ________________     __     _____    _____.__
     ____  ________________     __     _____    _____.__
     ____  ________________     __     _____    _____.__
     ____  ________________     __     _____    _____.__
     ____  ________________     __     _____    _____.__

F1/5:Refresh screen  F2/6:Next screen  F4:Update  Msg-wait:Menu
```

TQL EXAMPLE

The main processing program (shown following) is used to enter new orders, perform maintenance operations on existing orders and (in all cases) adjust the quantity on hand in the inventory file according to the number of items ordered or returned.

```
IDENTIFICATION DIVISION.
PROGRAM-ID.  ORD.

DATA DIVISION.

FILE   ORD.
RECORD ORDHDR.
RECORD ORDDTL.

FILE INV.
RECORD INVREC.

FILE TSPFILE.
RECORD TQLTSPR.

WORKING-STORAGE SECTION.
01  WORK-AREA.
    05  TOT-PRICE  PIC 9(5)V99.
    05  PREV-QTY   PIC 9(4).
```

TQL EXAMPLE

```
       DECLARATIVES SECTION.

       ON READ OF ORDHDR
          READ TQLTSPR VIA HDR-CUST.

[1]    ON READ OF ORDDTL
          MOVE ORD-QTY  TO PREV-QTY
          READ INVREC VIA ORD-PART.

       ON WRITE OF ORDHDR
        READ TQLTSPR VIA HDR-CUST
          ON ERROR 'INVALID CUSTOMER #'.

[2]    ON WRITE OF ORDDTL
        READ INVREC VIA ORD-PART
          ON ERROR 'BAD PART NUM'
        READ TQLTSPR VIA ORD-CUST
          ON ERROR 'INVALID CUST #'
        MOVE ORD-CUST TO HDR-CUST
        MOVE ORD-NUM  TO HDR-NUM
        MOVE 0        TO HDR-LINE
        READ ORDHDR VIA HDR-KEY
          ON ERROR 'MISSING HEADER RECORD'
        IF INV-QOH < ORD-QTY
           ERROR 'NOT ENOUGH GOODS'
        COMPUTE INV-QOH = INV-QOH + PREV-QTY - ORD-QTY
        MOVE ORD-LINE TO HDR-LAST-LINE.
```

TQL EXAMPLE

```
      DISPLAY DIVISION.

[3]   NEWORDER: READ ORDHDR
                 HDR-CUST HDR-NUM HDR-PO-NUM
                 USING DEMOORD1  ON  ENTER  ORDER.

[4]   ORDER:    MOVE HDR-CUST           TO ORD-CUST
                MOVE HDR-NUM            TO ORD-NUM
                MOVE HDR-LAST-LINE + 1 TO ORD-LINE
                READ ORDDTL
                 ORDDTL
                 USING DEMOORD2.

[5]   ORDDISP:  READ ORDHDR
                 HDR-CUST HDR-NUM  HDR-PO-NUM  CM-COMPANY  NL$
              8 ( READ ORDDTL  FROM  HDR-ORD
                   ORD-LINE ORD-PART  INV-DESC  ORD-QTY  INV-PRICE
                   COMPUTE TOT-PRICE = INV-PRICE * ORD-QTY
                   TOT-PRICE  NL$
                ) USING DEMOORD3.
```

TQL EXAMPLE

Notes to "ORD" program:

[1] Whenever an order detail record is read, the number of items ordered (ORD-QTY) is saved in working-storage field "PREV-QTY". This is done so that the quantity on hand in inventory can be recomputed if the detail item is updated (or deleted).

[2] Whenever an order detail record is written this coding validates the part number and the cutomer number according to the data in other files.

It also verifies that there is an existing header record for this detail record.

If the quantity-on-hand in the INV file (INV-QOH) is not sufficient an error message is produced ("NOT ENOUGH GOODS")

Finally, the inventory quantity on hand is recomputed and the inventory file is updated too.

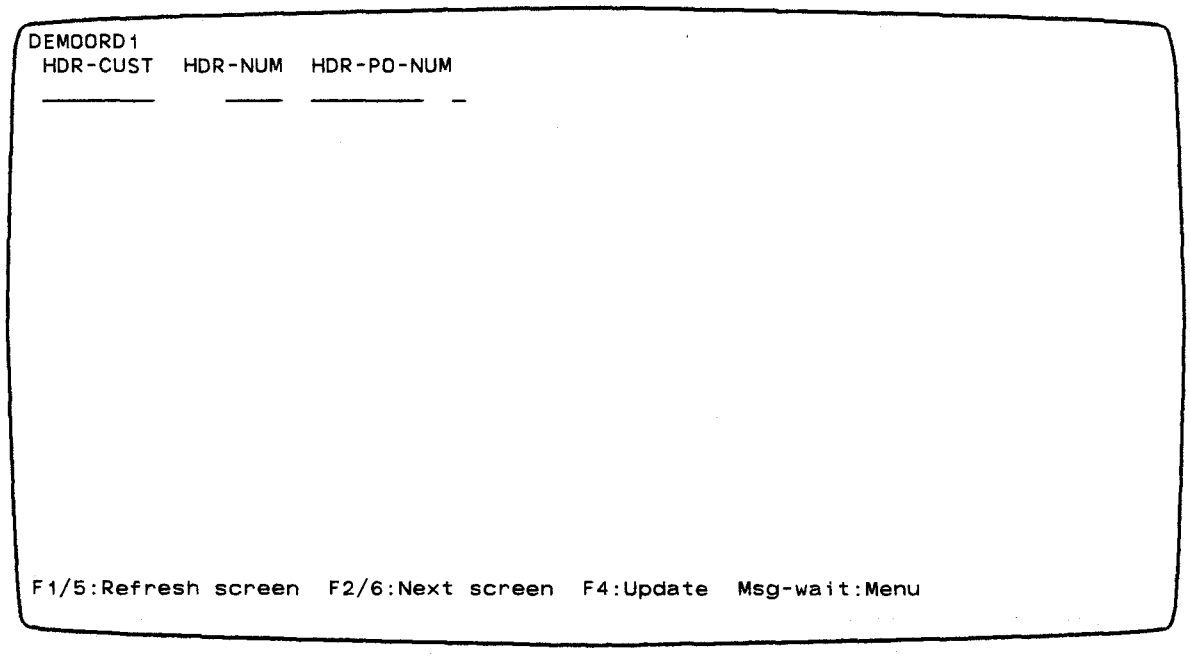
[3] The display "NEWORDER" is used to enter a new order. The "ON ENTER" clause specifies that when the user has entered the data in screen "DEMOORD1" he/she is to be taken (in data entry mode) to pre-defined display "ORDER".

The display "ORDER" therefore, is chained to the entry of a new order.

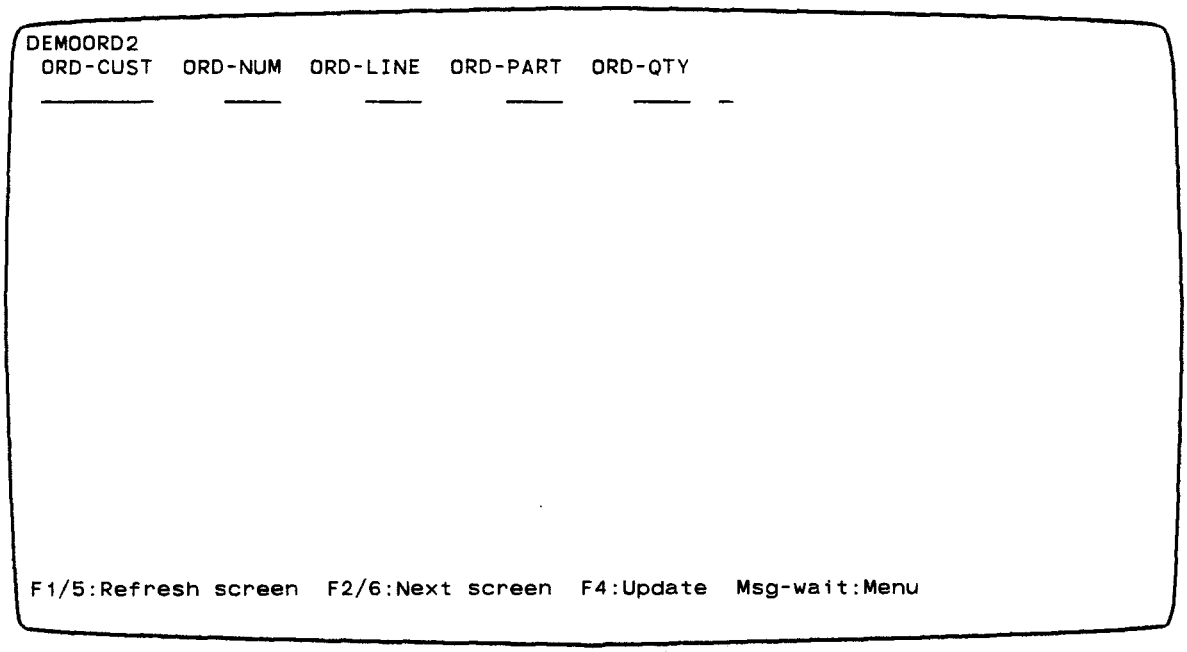
[4] The display "ORDER" is used as described above (as a secondary activity of order entry). It may also be used directly to perform maintenance activities on order detail records.

[5] The display "ORDDISP" displays the header information for an order and displays (on the same screen) up to 8 order detail records.

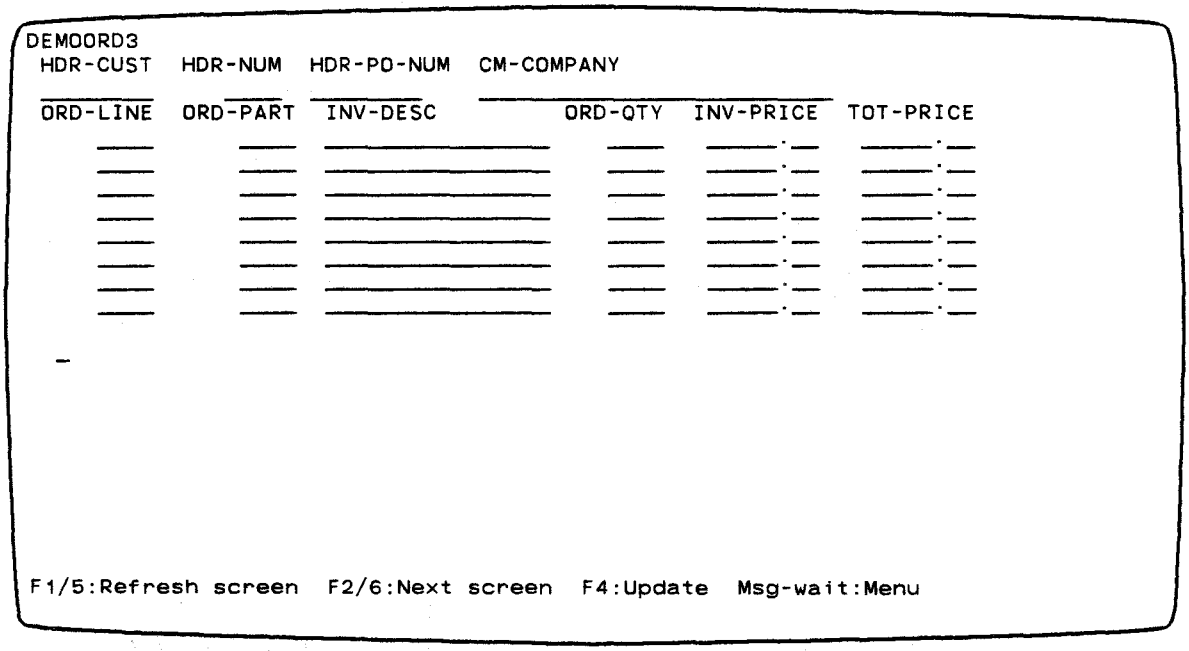
TQL EXAMPLE


DEMOORD1
HDR-CUST
HDR-NUM
HDR-PO-NUM
F1/5:Refresh screen
F2/6:Next screen
F4:Update
Msg-wait:Menu

TQL EXAMPLE


DEMOORD2
ORD-CUST
ORD-NUM
ORD-LINE
ORD-PART
ORD-QTY
F1/5:Refresh screen
F2/6:Next screen
F4:Update
Msg-wait:Menu

TQL EXAMPLE


DEMOORD3
HDR-CUST
HDR-NUM
HDR-PO-NUM
CM-COMPANY
ORD-LINE
ORD-PART
INV-DESC
ORD-QTY
INV-PRICE
TOT-PRICE
F1/5:Refresh screen
F2/6:Next screen
F4:Update
Msg-wait:Menu

TQL EXAMPLE

The following TQL program was written to generate invoices from the orders in the order file. A number of sample invoices produced by this program (using test data) are shown following the program source.

```
IDENTIFICATION DIVISION.
PROGRAM-ID.  INVOICE.

DATA DIVISION.
FILE   ORD.
RECORD ORDHDR.
RECORD ORDDTL.

FILE INV.
RECORD INVREC.

FILE TSPFILE.
RECORD TQLTSPR.

WORKING-STORAGE SECTION.
01  WORK-AREA.
    05  TOT-PRICE        PIC 9(5)V99.
    05  GRAND            PIC 9(6)V99.
    05  TAX              PIC 9(6)V99.
    05  FINAL            PIC 9(6)V99.
    05  SUM-TAX          PIC 9(7)V99.
    05  SUM-DUE          PIC 9(7)V99.
    05  SUM-GOODS        PIC 9(7)V99.

DECLARATIVES SECTION.
ON READ OF ORDHDR
   READ TQLTSPR VIA HDR-CUST.

ON READ OF ORDDTL
   READ INVREC VIA ORD-PART.
```

TQL EXAMPLE

```
REPORT DIVISION.

INVOICE:  READ ORDHDR  HOME$
         TAB$(10) 'SAMPLE ORDER INVOICE'
           SKIP$(4) YY$'/'MON$'/'DD$   NL$ NL$
         'CUST #    ORD#  P.O.#     COMPANY NAME' NL$
          HDR-CUST ' ' HDR-NUM ' ' HDR-PO-NUM ' ' CM-COMPANY
 NL$ NL$
    '  LINE  PART#   DESCRIPTION'
   TAB$(34) 'QUANTITY     PRICE       TOTAL'

 NL$   50 (READ ORDDTL FROM HDR-ORD
            ORD-LINE  ' '
            ORD-PART  ' '
            INV-DESC  ' '
            ORD-QTY   ' '
            INV-PRICE '   '
            COMPUTE TOT-PRICE = INV-PRICE * ORD-QTY
            TOT-PRICE
            ADD TOT-PRICE TO GRAND NL$)  NL$
   COMPUTE TAX = GRAND * 0.07
   COMPUTE FINAL = GRAND + TAX
   TAB$(41) 'TOTAL PRICE ' GRAND  NL$
   TAB$(41) '  SALES TAX '   TAX  NL$
   TAB$(41) ' AMOUNT DUE ' FINAL  NL$
   ADD TAX   TO SUM-TAX
   ADD FINAL TO SUM-DUE
   ADD GRAND TO SUM-GOODS
   ON PRNTR

   AT END HOME$ NL$ NL$
   'TOTAL VALUE OF GOODS SOLD' SUM-GOODS NL$
   ' TOTAL TAX DUE GOVERNMENT' SUM-TAX   NL$
   '  TOTAL AMOUNT TO COLLECT' SUM-DUE   NL$.
```

TQL EXAMPLE

The following report was produced by the program INVOICE.

SAMPLE ORDER INVOICE 83/06/01

CUST # ORD# P.O.# COMPANY NAME
COA00000 1 BILL CITY OF ARVADA

LINE	PART#	DESCRIPTION	QUANTITY	PRICE	TOTAL
1	3	RED SHIRT	4	22.50	90.00
2	1	WHITE SHIRT	1	11.95	11.95
3	3	RED SHIRT	4	22.50	90.00
4	7	THIN TIE	3	2.00	6.00
5	11	NEHRU JACKETS	3	1.95	5.85

TOTAL PRICE	203.80
SALES TAX	14.27
AMOUNT DUE	218.07

TQL EXAMPLE

SAMPLE ORDER INVOICE 83/06/01

CUST #	ORD#	P.O.#	COMPANY NAME
GLO00000	1	DAVID	GENERAL LAND OFFICE

LINE	PART#	DESCRIPTION	QUANTITY	PRICE	TOTAL
1	3	RED SHIRT	5	22.50	112.50
2	5	WIDE TIE	8	6.50	52.00
3	7	THIN TIE	3	2.00	6.00

TOTAL PRICE	374.30
SALES TAX	26.20
AMOUNT DUE	400.50

TQL EXAMPLE

SAMPLE ORDER INVOICE 83/06/01

CUST #	ORD#	P.O.#	COMPANY NAME
GLO00000	56	XYZ	GENERAL LAND OFFICE

LINE	PART#	DESCRIPTION	QUANTITY	PRICE	TOTAL
1	5	WIDE TIE	4	6.50	26.00
2	1	WHITE SHIRT	7	11.95	83.65

TOTAL PRICE	483.95
SALES TAX	33.88
AMOUNT DUE	517.83

TQL EXAMPLE

```
TOTAL VALUE OF GOODS SOLD     1062.05
 TOTAL TAX DUE GOVERNMENT       74.35
  TOTAL AMOUNT TO COLLECT     1136.40
```



```
JDE** FORMS=ALROS1,END;
```

2. KWIC INDEX INDEX

- E -

- F -

- H -

- I -

- K -

- L -

- M -

- N -

- O -

- P -

- T -

- U -

- V -

- W -

- E n d O f D o c u m e n t -

5. CHAPTER V - PROGRAM CONTROL SYSTEM PCS

This chapter of the TIP/30 reference manual describes the facilities of the Program Control System (PCS).

PCS is the component of TIP/30 that provides monitor-level functions for transaction programs. All transaction programs are executed under the control of PCS.

PCS provides such services as:

- inter-program linkage
- timer facilities
- run-time debugging facilities.

5.1 PROGRAM CONTROL SYSTEM PCS

TIP/30 provides the following methods of transferring control between programs:

TIPSUB - "CALL" another program

TIPSUBP - "CALL" a subroutine

TIPXCTL - "GOTO" another program

TIPFORK - create asynchronous process

The "TIPSUB" facility allows a program to "perform" another program and receive control when that program is finished.

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

The "TIPXCTL" facility allows a program to "GO TO" another program with no return of control.

The "TIPFORK" facility allows a program to "start up" another program as an asynchronous task, and thus create an independently executing program.

These facilities allow the programmer writing on-line programs to use control structures that are familiar and indeed taken for granted in typical batch programs. All TIP/30 programs (irrespective 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 be treated either as a subtask or as a main task without the need for special code in the program.

5.1.1 ON-LINE PROGRAM STRUCTURE PCS

On-line 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 example is illustrated using COBOL74 syntax.

TIP/30 passes a transaction program up to five parameters (in the following fixed order!):

- "PIB" - Process Information Block
- "CDA" - Continuity Data Area
- "MCS" - Message Control System workarea
- "WORK" - Workarea
- "GDA" - Global Data Area [TIP generation option]

-+*+-

5.1.2 PROCESS INFORMATION BLOCK PCS: pib

The PIB is a fixed size and fixed format area that contains information about the transaction that is executing. The layout of the PIB is given by the COBOL copy book "TC-PIB" supplied in the TIP library:

```
000001*
000002* TIP/30  -  PROCESS INFORMATION BLOCK
000003*
000004      05  PIB-TRID                          PICTURE X(8).
000005      05  PIB-UID                           PICTURE X(8).
000006      05  PIB-TID                           PICTURE X(4).
000007      05  PIB-STATUS                        PICTURE X.
000008*****************************************************************
000009*  THE FOLLOWING 88-LEVEL DATA ITEMS ARE SET BY TIPFCS          *
000010*             FCS ERROR RETURN CODES                            *
000011*****************************************************************
000012          88  PIB-GOOD                      VALUE ' '.
000013          88  PIB-DUP-AFT-NAME              VALUE 'C'.
000014          88  PIB-DUP-KEY                   VALUE 'D'.
000015          88  PIB-EOF                       VALUE 'E'.
000016          88  PIB-ACTIVE                    VALUE 'H'.
000017          88  PIB-IO-ERROR                  VALUE 'F'.
000018          88  PIB-FUNCTION                  VALUE 'G'.
000019          88  PIB-LOCKED                    VALUE 'L'.
000020          88  PIB-NOT-FOUND                 VALUE 'N'.
000021          88  PIB-NOT-HELD                  VALUE 'X'.
000022          88  PIB-SECURITY                  VALUE 'K'.
000023          88  PIB-HELD                      VALUE 'Y'.
000024          88  PIB-WRONG-MODE                VALUE 'W'.
000025          88  PIB-PROG-ABEND                VALUE 'A'.
000026      05  PIB-SYSTEM                        PICTURE X.
000027          88  PIB-EOJ-PENDING               VALUE 'E'.
```

PIB-TRID This eight byte field contains the transaction name 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 (eg: TIPSUB) may require information moved into this field.

PIB-UID This eight byte field contains the user-id of the user that is executing the program. This field will contain "TP" if the user had not logged on TIP/30.

This field will contain "BACK$nnn" [where nnn represents 3 digits] if the program is being executed in the background.

This field will contain "tttt*BYP" if the program is executing on a "bypass" terminal [where tttt is the terminal name of the originating terminal].

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

Certain TIP/30 subroutine calls (eg: TIPFORK) may require information moved into this field.

PIB-STATUS This one byte field contains the status returned as a result of <u>any</u> call to a TIP/30 subroutine. A number of 88 level items are defined in the copy book for the convenience of the programmer. The programmer <u>MUST</u> interrogate this status field after every call to a TIP/30 subroutine (eg: TIPFCS, TIPSUB, etc).

A value of PIB-GOOD indicates that the subroutine call was successful as far as TIP/30 is concerned. Any other value is an error.

PIB-SYSTEM This one byte field is set to the 'PIB-EOJ-PENDING' value if and only if TIP/30 has been given a shutdown command (EOJ). This mechanism allows TIP/30 native mode programs to detect that an EOJ is requested. When a program detects this condition, it should attempt to issue a call to the TIPRTN subroutine as soon as practical.

-+*+-

5.1.3 CONTINUITY DATA AREA PCS: cda

The continuity data area is an area of storage that is provided by TIP/30 for transaction programs. It is the only area that is copied to and from programs during inter-program linkage. The size and format of this area is determined by the programmer. The size of the area is given in the TIP/30 catalogue entry for the transaction.

If a transaction program is called from the TIP/30 command line and the transaction was catalogued with CML=YES in the TIP/30 catalogue, TIP/30 will parameterize the command line information into the CDA. The COBOL copy book TC-CDA in the TIP library provides the format for this particular use of the CDA:

```
000001*
000002*        TIP/30  -  COMMAND LINE FORMAT OF CDA
000003*
000004      05  CDA-PARAMETERS.
000005          10  CDA-PARAM      OCCURS 8 TIMES     PICTURE X(8).
000006
000007      05  CDA-OPTIONS.
000008          10  CDA-OPTION     OCCURS 8 TIMES     PICTURE X.
000009
000010      05  CDA-TEXT                              PICTURE X(80).
```

CDA-PARAMS Up to eight command line parameters will be parameterized into these fields. Strictly numeric parameters will be right justified and leading zero filled. Non-numeric parameters will be left justified and trailing blank filled.

CDA-OPTIONS This field will contain the command line option information (the options immediately follow the transaction name and are separated from the transaction name by a comma or a slash).

If no options were given, this field contains spaces.

CDA-TEXT This field contains the command line parameters in the same format they were entered (ie: not parameterized).

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

-+*+-

5.1.4 MESSAGE CONTROL SYSTEM WORKAREA PCS: mcs

The MCS area is an optional area that will be reserved by TIP/30 for the transaction program. This area is normally used by the transaction program as a screen format I/O area. The size of this area must be given in the TIP/30 catalogue entry for the transaction program.

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

The COBOL copy book TC-MCS defines the layout of the MCS packet prefix that is required to interface with the Message Control System. This prefix is described in the section of the reference manual describing the Message Control System (MCS).

```
000001*
000002*        TIP/30  -  MESSAGE  CONTROL  SYSTEM  PACKET
000003*
000004      02  MCS-NAME                          PICTURE X(8).
000005      02  MCS-TERM                          PICTURE X(4).
000006      02  MCS-FUNCTION                      PICTURE X.
000007*
000008*       'A' - READ FULL SCREEN ON TIPMSGI
000009*       'D' - SEND DATA ONLY (NO HEADINGS DATA)
000010*           - LOW VALUE FIELDS ARE NOT SENT
000011*       'M' - SEND MESSAGE AS UNSOLICTED
000012*       'P' - CAUSE TERMINAL TO PRINT NON-TRANSPARENT
000013*       'R' - CAUSE TIPMSGE TO REFRESH ALL FCC'S
000014*       'S' - STOP SENDING HEADING DATA
000015*             WHEN MCS-COUNT REACHES ZERO
000016*
000017      02  MCS-HOLD                          PICTURE X.
000018*
000019*         'H' - TIPMSGI NOT TO RELEASE RECORD LOCK(S)
000020*
000021      02  MCS-SIZE                          PICTURE S9(4) COMP-4 SYNC.
000022      02  MCS-STATUS                        PICTURE X.
000023          88  MCS-GOOD                        VALUE ' '.
000024          88  MCS-XMIT                        VALUE ' '.
000025          88  MCS-MSG-WAIT                    VALUE '0'.
000026          88  MCS-FKEY1                       VALUE '1'.
000027          88  MCS-FKEY2                       VALUE '2'.
000028          88  MCS-FKEY3                       VALUE '3'.
000029          88  MCS-FKEY4                       VALUE '4'.
000030          88  MCS-FKEY5                       VALUE '5'.
000031          88  MCS-FKEY6                       VALUE '6'.
000032          88  MCS-FKEY7                       VALUE '7'.
000033          88  MCS-FKEY8                       VALUE '8'.
000034          88  MCS-FKEY9                       VALUE '9'.
000035          88  MCS-FKEY10                      VALUE 'A'.
```

MESSAGE CONTROL SYSTEM WORKAREA

```
000036          88  MCS-FKEY11                    VALUE 'B'.
000037          88  MCS-FKEY12                    VALUE 'C'.
000038          88  MCS-FKEY13                    VALUE 'D'.
000039          88  MCS-FKEY14                    VALUE 'E'.
000040          88  MCS-FKEY15                    VALUE 'F'.
000041          88  MCS-FKEY16                    VALUE 'G'.
000042          88  MCS-FKEY17                    VALUE 'H'.
000043          88  MCS-FKEY18                    VALUE 'I'.
000044          88  MCS-FKEY19                    VALUE 'J'.
000045          88  MCS-FKEY20                    VALUE 'K'.
000046          88  MCS-FKEY21                    VALUE 'L'.
000047          88  MCS-FKEY22                    VALUE 'M'.
000048          88  MCS-F-REBUILD                 VALUE '1' '5'.
000049          88  MCS-F-NEXT                    VALUE '2' '6'.
000050          88  MCS-F-UPDATE                  VALUE '4' '8'.
000051      02  MCS-FILLER                      PICTURE X.
000052*
000053*         VALID FILLER VALUES ARE ' ', '_', OR '*'
000054*
000055      02  MCS-COUNT                       PICTURE S9(4) COMP-4 SYNC.
000056/
000057      02  MCS-DATA.
000058*
000059*     USER SUPPLIED RECORD LAYOUT FOR MCS SCREEN FOLLOWS HERE
000060*
```

-+*+-

5.1.5 WORK AREA PCS: workarea

The WORKAREA is an optional area that will be reserved by TIP/30 for the transaction program. The size and layout of the workarea is entirely at the discretion of the programmer. The size of the workarea must be given in the TIP/30 catalogue entry for the transaction program.

The workarea is used by native mode programs as an area containing fields which 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 IMSCOD=YES option.

The workarea is also the proper place for the various record areas for files that are manipulated on-line.

The workarea is set to low values (X'00') by TIP/30 before the transaction program is called.

-+*+-

5.1.6 GLOBAL DATA AREA PCS: gda

The GLOBAL DATA AREA is an optional area that may be specified 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 available to all TIP/30 programs.

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

One use of the GDA might be (for example) the storage of a common table that is referenced by many on-line programs. Instead of having each program that needs to reference the table read the table into the program's work area, the GDA could be initialized by a system startup program. Thereafter, all programs that need to refer to the table could reference it in the GDA.

The GDA is a serial resource; modification of this area might involve race conditions. The TIP/30 flags (see discussion of TIPFLAG subroutine) may be useful in queueing requests to modify the contents of the GDA.

-+*+-

5.2 USER PROGRAM ABORT TRAP TIPABRT

Normally when a user program aborts, TIP/30 will react by calling the Post Mortem Dump Analysis program (PMDA) on behalf of the program in error. If the user wants to get control in his program when an abort condition occurs, the program must issue a call to TIPABRT to set up an abnormal termination entry point (island code) in the user program. (This facility is normally used by assembler language programs.

Syntax:

```
CALL TIPABRT,(savearea)
```

Where:

savearea The first parameter on the call is the location in which the PSW and registers are stored after an abnormal condition has occured. TIP/30 will enter the program at the first location beyond the psw/register save area. Upon entry register 15 contains the address of the first instruction to be executed, and can be used as the cover register for the abort routine. Once the abort routine has been entered, any further abnormal conditions will result in loading PMDA unless the user program calls TIPABRT again to re-establish the abnormal termination entry point.

Example:

```
            CALL TIPABRT,(ABTERM) . SETUP ABTERM ENTRY POINT
            .
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
```

Error Conditions:
The save area must be fullword aligned.

5.2.1 CONVERT 32 BYTES TO 32 BITS TIPBITS

This subroutine is supplied as an aid for COBOL language programmers that are manipulating the TIP/30 Flag bits. (see section on TIPFLAG following).

This subroutine will convert a string of 32 bytes (each containing a "0" or "1" value) into a fullword [S9(6) COMP-4 SYNC] with the corresponding bits in the fullword set to a zero or one.

Note that the bits in a fullword are numbered from 0 to 31 from right to left.

Syntax:

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

Where:

BIT-SWITCHES The receiving field defined as PIC S9(6) COMP-4 SYNCHRONIZED.

BYTE-SWITCHES The values to set the bits in the receiving field.

Each byte must contain a zero or one (X'F0' or X'F1').

Additional Considerations:

A copy element named: TIP/TC-BITS is provided which contains the required definition of the two parameters in the above syntax description. See section on TIPFLAG subroutine.

-+*+-

5.2.2 CONVERT 32 BITS TO 32 BYTES TIPBYTES

This subroutine is supplied as an aid for COBOL language programmers that are manipulating the TIP/30 Flag bits. (see section on TIPFLAG following).

This subroutine will convert a fullword [S9(6) COMP-4 SYNC] into a string of 32 bytes with each byte containing a 0 or 1 depending on the value in the corresponding bits in the fullword.

Note that the bits in a fullword are numbered from 0 to 31 from right to left.

Syntax:

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

Where:

BIT-SWITCHES The fullword field defined as PIC S9(6) COMP-4 SYNCHRONIZED.

BYTE-SWITCHES The bytes to be set to a one or zero corresponding to the setting of each bit in the field BIT-SWITCHES.

Additional Considerations:

A copy element named: TIP/TC-BITS is provided which contains the required definition of the two parameters in the above syntax description. See section on TIPFLAG subroutine.

-+*+-

5.3 TODAY'S DATE TIPDATE

This routine returns the current date in the format 'DAY MONTH DD 19YY'.

Syntax:

```
CALL 'TIPDATE' USING DATE-AREA
```

Where:

DATE-AREA is a 30 character field to receive the date.

Eg: "WEDNESDAY AUGUST 18 1982 "

Example:

```
05  TODAYS-DATE                PIC X(30).
     ...
     ...
     ...
CALL 'TIPDATE' USING TODAYS-DATE.
```

Error Conditions:

None.

5.4 FORCE PROGRAM DUMP TIPDUMP

This subroutine may be called to force a program dump at a specific point in the processing.

Syntax:

```
CALL 'TIPDUMP'
```

Where:
No parameters.

Error Conditions:
None.

Additional Considerations:

The dump is caused by executing an illegal machine instruction. (X'00DEAD00'). Program register 14 will hold the address of the call to TIPDUMP.

5.5 FILE ERROR EDIT TIPFCER

This subroutine provides the user with a standard method of converting the error code returned in the FCS file name packet (9th byte) into a literal error message. The error message built by this routine has the following format:

FCS Error=A, File=logical, Meaning='description of error'.

Syntax:

CALL 'TIPFCER' USING FILE-NAME, ERROR-LINE.

Where:

FILE-NAME is the file name packet which was used when the error was detected.

ERROR-LINE is an 80 character field to receive the literal error message.

Example:

```
05  PAYMASTER.
    10  AFT-NAME                    PIC X(8).
    10  FILE-STATUS                 PIC X.

05  ERROR-LINE                      PIC X(80).

                .....

CALL 'TIPFCER' USING PAYMASTER, ERROR-LINE
```

Error Conditions:
None.

5.6 FLAG SERVICES TIPFLAG

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

Syntax:

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

Where:

FUNCTION is a character code representing the function to be performed by TIPFLAG:

0 - wait for any of the flags in the mask to be set.

1 - wait for all of the flags in the mask to be set.

2 - wait for any of the flags in the mask to be set then clear the flags indicated by the mask.

3 - wait for all of the flags in the mask to be set then clear the flags indicated by the mask.

4 - wait for any of the flags in the mask to be clear.

5 - wait for all of the flags in the mask to be clear.

6 - wait for any of the flags in the mask to be clear then set the flags indicated by the mask.

7 - wait for all of the flags in the mask to be clear then set the flags indicated by the mask.

8 - set the flags indicated by the mask.

9 - clear the flags indicated by the mask.

MASK is the fullword used to identify those flags to be acted upon by flag services (each bit represents a flag).

RESULT is used to receive the flag word after the indicated function has been performed (the result field is only used for function codes 8 and 9).

Example:

```
       77  FLAG-BIT-THREE VALUE 4    PIC 9(7) COMP-4 SYNC.

       CALL 'TIPFLAG' WAIT-ALL-CLEAR-SET, FLAG-BIT-THREE.
```

Error Conditions:

None.

Additional Considerations:

The COBOL copy element (TC-FLAG) provides a complete set of TIPFLAG function codes:

```
000001*****************************************************************
000002*  USED AS FUNCTION CODES TO DIRECT TIP FLAG SERVICES            *
000003*****************************************************************
000004    05  WAIT-ANY-SET                        PICTURE X VALUE '0'.
000005    05  WAIT-ALL-SET                        PICTURE X VALUE '1'.
000006    05  WAIT-ANY-SET-CLEAR                  PICTURE X VALUE '2'.
000007    05  WAIT-ALL-SET-CLEAR                  PICTURE X VALUE '3'.
000008    05  WAIT-ANY-CLEAR                      PICTURE X VALUE '4'.
000009    05  WAIT-ALL-CLEAR                      PICTURE X VALUE '5'.
000010    05  WAIT-ANY-CLEAR-SET                  PICTURE X VALUE '6'.
000011    05  WAIT-ALL-CLEAR-SET                  PICTURE X VALUE '7'.
000012    05  SET-ON                              PICTURE X VALUE '8'.
000013    05  SET-OFF                             PICTURE X VALUE '9'.
```

The COBOL copy element TIP/TC-BITS describes work areas that may be useful to the COBOL programmer that is manipulating TIPFLAGs.

This copy book is used in conjunction with the subroutines "TIPBITS" and "TIPBYTES" (as described earlier).

```
000001*****************************************************************
000002*              DEFINE THE 32 SWITCHES USED BY TIPFLAG           *
000003*****************************************************************
000004*
000005      05  BIT-SWITCHES                        PICTURE 9(6)
000006          COMPUTATIONAL-4 SYNCHRONIZED.
000007*
000008      05  BYTE-SWITCHES.
000009          10  SWITCH-31                       PICTURE 9.
000010              88  SWITCH-31-OFF               VALUE '0'.
000011              88  SWITCH-31-ON                VALUE '1'.
000012          10  SWITCH-30                       PICTURE 9.
000013              88  SWITCH-30-OFF               VALUE '0'.
000014              88  SWITCH-30-ON                VALUE '1'.
000015          10  SWITCH-29                       PICTURE 9.
000016              88  SWITCH-29-OFF               VALUE '0'.
000017              88  SWITCH-29-ON                VALUE '1'.
000018          10  SWITCH-28                       PICTURE 9.
000019              88  SWITCH-28-OFF               VALUE '0'.
000020              88  SWITCH-28-ON                VALUE '1'.
000021          10  SWITCH-27                       PICTURE 9.
000022              88  SWITCH-27-OFF               VALUE '0'.
000023              88  SWITCH-27-ON                VALUE '1'.
000024          10  SWITCH-26                       PICTURE 9.
000025              88  SWITCH-26-OFF               VALUE '0'.
000026              88  SWITCH-26-ON                VALUE '1'.
000027          10  SWITCH-25                       PICTURE 9.
000028              88  SWITCH-25-OFF               VALUE '0'.
000029              88  SWITCH-25-ON                VALUE '1'.
000030          10  SWITCH-24                       PICTURE 9.
000031              88  SWITCH-24-OFF               VALUE '0'.
000032              88  SWITCH-24-ON                VALUE '1'.
000033          10  SWITCH-23                       PICTURE 9.
000034              88  SWITCH-23-OFF               VALUE '0'.
000035              88  SWITCH-23-ON                VALUE '1'.
000036          10  SWITCH-22                       PICTURE 9.
000037              88  SWITCH-22-OFF               VALUE '0'.
000038              88  SWITCH-22-ON                VALUE '1'.
000039          10  SWITCH-21                       PICTURE 9.
000040              88  SWITCH-21-OFF               VALUE '0'.
000041              88  SWITCH-21-ON                VALUE '1'.
000042          10  SWITCH-20                       PICTURE 9.
```

```
000043              88  SWITCH-20-OFF           VALUE '0'.
000044              88  SWITCH-20-ON            VALUE '1'.
000045          10  SWITCH-19                  PICTURE 9.
000046              88  SWITCH-19-OFF           VALUE '0'.
000047              88  SWITCH-19-ON            VALUE '1'.
000048          10  SWITCH-18                  PICTURE 9.
000049              88  SWITCH-18-OFF           VALUE '0'.
000050              88  SWITCH-18-ON            VALUE '1'.
000051          10  SWITCH-17                  PICTURE 9.
000052              88  SWITCH-17-OFF           VALUE '0'.
000053              88  SWITCH-17-ON            VALUE '1'.
000054          10  SWITCH-16                  PICTURE 9.
000055              88  SWITCH-16-OFF           VALUE '0'.
000056              88  SWITCH-16-ON            VALUE '1'.
000057          10  SWITCH-15                  PICTURE 9.
000058              88  SWITCH-15-OFF           VALUE '0'.
000059              88  SWITCH-15-ON            VALUE '1'.
000060          10  SWITCH-14                  PICTURE 9.
000061              88  SWITCH-14-OFF           VALUE '0'.
000062              88  SWITCH-14-ON            VALUE '1'.
000063          10  SWITCH-13                  PICTURE 9.
000064              88  SWITCH-13-OFF           VALUE '0'.
000065              88  SWITCH-13-ON            VALUE '1'.
000066          10  SWITCH-12                  PICTURE 9.
000067              88  SWITCH-12-OFF           VALUE '0'.
000068              88  SWITCH-12-ON            VALUE '1'.
000069          10  SWITCH-11                  PICTURE 9.
000070              88  SWITCH-11-OFF           VALUE '0'.
000071              88  SWITCH-11-ON            VALUE '1'.
000072          10  SWITCH-10                  PICTURE 9.
000073              88  SWITCH-10-OFF           VALUE '0'.
000074              88  SWITCH-10-ON            VALUE '1'.
000075          10  SWITCH-09                  PICTURE 9.
000076              88  SWITCH-09-OFF           VALUE '0'.
000077              88  SWITCH-09-ON            VALUE '1'.
000078          10  SWITCH-08                  PICTURE 9.
000079              88  SWITCH-08-OFF           VALUE '0'.
000080              88  SWITCH-08-ON            VALUE '1'.
000081          10  SWITCH-07                  PICTURE 9.
000082              88  SWITCH-07-OFF           VALUE '0'.
000083              88  SWITCH-07-ON            VALUE '1'.
000084          10  SWITCH-06                  PICTURE 9.
000085              88  SWITCH-06-OFF           VALUE '0'.
000086              88  SWITCH-06-ON            VALUE '1'.
000087          10  SWITCH-05                  PICTURE 9.
000088              88  SWITCH-05-OFF           VALUE '0'.
000089              88  SWITCH-05-ON            VALUE '1'.
000090          10  SWITCH-04                  PICTURE 9.
000091              88  SWITCH-04-OFF           VALUE '0'.
000092              88  SWITCH-04-ON            VALUE '1'.
```

```
000093          10  SWITCH-03                       PICTURE 9.
000094              88  SWITCH-03-OFF               VALUE '0'.
000095              88  SWITCH-03-ON                VALUE '1'.
000096          10  SWITCH-02                       PICTURE 9.
000097              88  SWITCH-02-OFF               VALUE '0'.
000098              88  SWITCH-02-ON                VALUE '1'.
000099          10  SWITCH-01                       PICTURE 9.
000100              88  SWITCH-01-OFF               VALUE '0'.
000101              88  SWITCH-01-ON                VALUE '1'.
000102          10  SWITCH-00                       PICTURE 9.
000103              88  SWITCH-00-OFF               VALUE '0'.
000104              88  SWITCH-00-ON                VALUE '1'.
000105*
000106*************************************************************************
000107*     TO COMPRESS BYTE-SWITCHES INTO BIT-SWITCHES FOR TIPFLAG.          *
000108*                                                                       *
000109* CALL 'TIPBITS' USING BIT-SWITCHES, BYTE-SWITCHES.                     *
000110*                                                                       *
000111*************************************************************************
000112*     TO EXPAND BIT-SWITCHES TO BYTE-SWITCHES FOR PROGRAM USE.          *
000113*                                                                       *
000114* CALL 'TIPBYTES' USING BIT-SWITCHES, BYTE-SWITCHES.                    *
000115*                                                                       *
000116*************************************************************************
```

5.7 CREATE AN ASYNCHRONOUS PROCESS TIPFORK

This call is used to start a program as an asynchronous process. After this call is issued, the calling program will continue to execute while the called program begins execution. This call may be used to start a background process.

As a background process the program may perform all TIP/30 functions except solicit input from a terminal. A background process may be useful for time consuming file processing operations, for which the user does not require an immediate response.

This call may also be used to start a program running interactively on another terminal.

To use this facility, the user must move the transaction-id of the program to be called to the field PIB-TRID and then issue the call. The CDA is copied to the called program.

Syntax:

```
CALL 'TIPFORK'
```

Where:

No parameters required.

If you wish to start up a new process, but attached directly to another terminal (ie. not as background), you should set PIB-TID to the destination terminal before calling TIPFORK.

For instance the destination terminal could be '*BYP' to start a new process running directly on the bypass terminal. Such a process may receive input messages from the bypass terminal.

Example:

```
MOVE 'PRINTQ' TO PIB-TRID.
MOVE 'TRMP'   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.
PIB-LOCKED	there is already a program running on the terminal.
PIB-SECURITY	the user running the initiating program does not have a high enough security to run the requested program or the requested program is locked due to time of day.

5.8 END ONLINE PROGRAM TIPRTN

This call is used to terminate a user program. When a program is terminated, control will resume in the calling program. IE. control returns to the previous level on the program stack.

Syntax:

```
CALL 'TIPRTN'
```

Where:
No parameters.

Error Conditions:
None.

Additional Considerations:

The CDA will be copied back to the calling program.

5.9 SNAP MEMORY TIPSNAP

This is a method for a program to produce snap dumps of various sections of memory.

Syntax:

```
CALL 'TIPSNAP' USING bgn-1,end-1,...bgn-n,end-n.
```

Where:
each parameter pair represents the start and ending location of an area of memory which is to be snap dumped.

Example:

```
CALL 'TIPSNAP' USING WORK-AREA, END-WORK,
                     MCS,       END-MCS.
```

Error Conditions:
None.

Additional Considerations:

This may be useful when debugging programs but you should remove such calls when placing program in production. Excessive number of calls to TIPSNAP will degrade TIP/30 performance.

5.10 PROGRAM LINKAGE TIPSUB

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. To use this facility, the calling program must move the catalogued transaction-id of the program to be called to PIB-TRID. The CDA is copied to the called program.

Syntax:

```
CALL 'TIPSUB'
```

Where:
No parameters.

Example:

```
MOVE 'ADDLNE' TO PIB-TRID.
CALL 'TIPSUB'.
IF NOT PIB-GOOD
    PERFORM ERR-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, WORD, MCS etc) is too large.

PIB-SECURITY the user running the initiating program does not have a high enough security to run the requested program or the requested program is locked due to the time of day.

PIB-PROG-ABEND the sub program aborted (program checked) during execution. In this case, PMDA would have been called on behalf of the sub program and PMDA would return control to the calling program with this error status.

5.11 SUB-ROUTINE LINKAGE TIPSUBP

This is a method for calling user written sub-routines which have been separately compiled and linked. Only parameters given by the calling program are passed to the sub-routine. The calling program must move the 'load module' name to PIB-TRID to identify the desired subroutine.

Syntax:

```
CALL 'TIPSUBP' USING param-1 ... param-n
```

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

Example:

```
MOVE 'CHKDGT00' TO PIB-TRID.

CALL 'TIPSUBP' USING PART-NUM, DIGIT.

IF NOT PIB-GOOD
    PERFORM ERROR-ON-SUBP-CALL.
```

Error Conditions:

PIB-NOT-FOUND the sub-routine is not resident.

Additional Considerations:

If the sub-routine is not written as re-entrant assembly language then it must be cataloged as USAGE=REUSE.

A sub-routine may do most TIP/30 calls with the exception of TIPSUB, TIPSUBP, TIPXCTL.

If the calling program is native mode the sub-routine may only do file I/O by calling TIPFCS.

If the calling program is an emulated IMS/90 program the sub-routine may only do IMS/90 style file I/O calls.

COBOL subprograms should end by using the "EXIT PROGRAM" statement.

Assembler subprograms should end by using the "RETURN" macro.

All sub-routines MUST be made resident via the TIP/30 job control stream.

5.12 TIMER SERVICES TIPTIMER

This function allows the user program to pause for a specific length of time. This is useful when a program must wait as in the case where an STS-HELD status has been received. Queue driven programs would normally call TIPTIMER when their input work queue was empty.

Syntax:

```
CALL 'TIPTIMER' USING TIME [,TIME-STATUS [,PREVIEW]]
```

Where:

TIME a fullword holding the time is seconds that the program is to be suspended.

The program will be re-activated when the time has elapsed, or an input message is available.

TIME-STATUS is a one byte status code which will receive the reason TIP/30 re-scheduled the program.

'M' indicates that a message is available although the requested time has not elapsed.

'T' indicates that the time period has elapsed or that the computer operator has requested TIP/30 to shut down.

PREVIEW is a 12-byte field into which TIP/30 will place a preview of the input message.

Example:

```
05  TIMER          PIC 9(7) COMP-4 SYNC.
05  STATUS         PIC X.
05  PREVIEW        PIC X(12).

MOVE 60 TO TIMER.
CALL 'TIPTIMER' USING TIMER, STATUS, PREVIEW.
```

In the example, TIP/30 will suspend execution of the program for 60 seconds or until an input message is available. If a message had arrived, STATUS would hold 'M' and PREVIEW would hold the first 12 text characters (in upper case) of the input message.

5.13 TRANSFER CONTROL TIPXCTL

This call is used to transfer control explicitly to another program on the same program stack level. The calling program must move the catalogued transaction-id to PIB-TRID.

Syntax:

```
CALL 'TIPXCTL'
```

Where:
No parameters.

Example:

```
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.

PIB-SECURITY the user running the initiating program does not have a high enough security to run the requested program or the requested program is locked due to time of day.

Additional Considerations:

The CDA will be copied to the next program.

6. CHAPTER VI - FILE CONTROL SYSTEM FCS

This chapter of the TIP/30 reference manual describes the facilities provided by the TIP File Control System (FCS).

FCS is the component of TIP/30 that provides the interface between transaction programs and data files.

This file interface allows transaction programs to access standard OS/3 files, standard OS/3 libraries, TIP/30 dynamic files (temporary scratch-pad type files) and, TIP/30 edit buffers.

The interface is implemented at the subroutine call level; that is, all requests for file services are provided by calling a supplied subroutine with appropriate parameters describing the required information.

6.1 FILE CONTROL SYSTEM FCS

The TIP/30 File Control System (FCS) is the interface between transaction programs and on-line files. FCS provides services at the program "CALL" level.

Programs call one subroutine (TIPFCS), and provide parameters which select the desired function to be performed and relevant file and record information.

Programs refer to files by referencing a Logical File Name (LFN). This LFN is the name by which the file is known to the system. The LFN need not be the same as the actual physical file name (LFD) as supplied in the Job Control for TIP/30.

The LFN is connected to the actual physical file by an entry in the TIP/30 catalogue. All on-line 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 required security level to access the file.

Programs may access a file only if it has been assigned to the program (either by an explicit OPEN request to FCS or an implicit OPEN requested in the program's catalogue entry).

Once a file is assigned to a program, an entry for it is placed in the Active File Table (AFT) for the terminal. Files which have entries in the terminal AFT are available (by reference to the LFN) to all programs that are run at the terminal until they are unassigned.

The following file organizations are fully supported by the TIP/30 File Control System:

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

TIP/30 FCS provides the ability to access OS/3 library elements, the capability of creating (on demand) TIP/30 Dynamic Files, and the ability to access Edit Buffers (a specific type of dynamic file) that are manipulated by the TIP/30 text editor (QED).

File integrity is maintained by the record locking feature. Records that are to be updated are locked when read. Records that are locked are unavailable to other processes until released.

A generalized resource locking facility is provided which enables a program to enter (or remove) a value in a key-holding table. Once a given value has been entered in the table, any process which attempts to enter the same value will be given a "locked" status. This essentially provides a generalized queueing mechanism that may be used to prevent concurrent access to any resource (ie: a file, a group of records etc).

By exercising a TIP/30 generation option, on-line files may be journalled. This facility allows either before images or after images (or both) to be written to the TIP/30 journal file.

"Before" images can be 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"). Dynamic files may be created and erased as needed, by programs. Many applications require temporary auxiliary storage and can make use of this facility.

6.2 TIPFCS AND THE TIP/30 CATALOGUE FCS

The files that a transaction program accesses may be specified in the program entry in the TIP/30 catalogue. This technique is highly recommended.

By utilizing this feature of the TIP/30 catalogue, the program does not need to explicitly open or close such files.

Refer to the section on the TIP/30 catalogue manager program (CAT) for detailed information.

6.3 RECORD AND FILE LOCKING FCS

It is generally accepted that two JOBS should not update the same file at the same time. Similarly, two online users, although they may be processing the same file should be protected from 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.

- JOE displays record 500 intending to update it.
- JOE is interrupted for a moment and TOM reads record 500, changes it at his terminal and re-writes the record in the file.
- JOE then changes the record and re-writes it in the file, overlaying TOM's update and perhaps causing problems which may not appear until much later.

With the record locking capability provided by FCS, this situation cannot occur; the updating process is assured that its logical integrity will be maintained.

6.3.1 SIMPLE RECORD HOLDING HOLD=YES

To lock a record the user program issues a CALL TIPFCS using the FCS-GETUP function code and supplies the KEY of the record. The function status of every CALL TIPFCS is stored in the ninth byte of the file-pkt (Logical File Name Packet) and in the PIB-STATUS field of the program information block. This status must be checked after each call.

When a record is being held for update, the key of the record is stored in a key holding table. If the function status after an FCS-GETUP is PIB-HELD then another user has the record held and all other records held for the program which just received the PIB-HELD are automatically released and must be re-acquired before updating.

If a user program receives a function status of PIB-HELD in response to an FCS-GETUP, meaning the record is locked for someone else then FCS automatically pauses the caller for one second. The program may try the GETUP again or CALL TIPTIMER to wait a little longer.

If the program issues an FCS-PUT without locking the record via an FCS-GETUP then the function status will be PIB-NOT-HELD and the FCS-PUT will be ignored.

To enter a value in the Key Holding Table, which could be used for locking a file or a section of a file, use FCS-HOLD and FCS-RELEASE.

-+*+-

6.3.2 RECORD HOLDING FOR THE TRANSACTION HOLD=TR

TIP/30 defines a transaction as the events which take place from the time an input message arrives until the request for the next input message is issued by the program.

Files may be defined with HOLD=TR. This means that updates are are to be held for the entire transaction. This is different from HOLD=YES where records are only held until they are updated and then removed from the holding table.

For programs which will access the same records of a file at the same time, option HOLD=TR requires more processing than HOLD=YES; however, the BEFORE images of all updated records are saved in the Journal file (or the Before file) and will be automatically rolled back by PMDA (Post Mortem Dump Analyzer) if the program aborts before the transaction completes.

A user program itself, may issue the CALL FCS-BACK to roll back updates on a file basis for HOLD=TR files.

To prevent deadlock situations for HOLD=TR files, programs which hold a record from several files should always acquire the records in the same order. (ie. File-A, File-B, File-C etc). If the program decides to roll back updates, it should issue FCS-BACK to the files in the reverse order in which the records where held.

For HOLD=YES, deadlock can never occur because the programs are allowed to re-try their FCS-GETUP.

-+*+-

6.3.3 RECORD HOLDING FOR THE UPDATE HOLD=UP

You may also specify HOLD=UP. Which allows the program to hold several records within one file until the record is updated. This does not provide for online roll-back or 'QUICK' recovery facility.

-+*+-

6.3.4 RECORD HOLDING SUMMARY FCS

GEN PARAM	ROLL BACK CAPABILITY	MULTI HELD PER FILE	RECORDS RLSED UPDT OR TREN	RELEASE ON HELD	POSSIBLE DEADLOCK
HOLD=YES	NO	NO	UPDT or GETUP	YES	NO
HOLD=TR	YES	YES	TREN	NO	YES
HOLD=UP	NO	YES	UPDT	NO	YES

-+*+-

6.3.5 FCS DEADLOCK CONSIDERATIONS FCS

Different user programs may be sharing the same memory region within TIP/30. It is therefore very important to remember not to lock a record and solicit terminal input, (ie. TIPMSGI or T@GET) since you may be swapped out leaving the record locked and the other program is loaded and may be trying to access the same record.

Similarly, do not set an indexed file in sequential mode and solicit terminal input. An indexed file set in sequential mode (via FCS-SETL) is effectively locked to all other users of the file. If terminal input is solicited, any other users of the file must wait at least until the input message is received. This causes unnecessary delay and overhead, and may seriously degrade the performance of the system.

Always set an indexed file that is in sequential mode back to random mode (via FCS-ESETL) before doing any terminal I/O.

-+*+-

6.4 SUMMARY OF FCS CALLS FCS: summary

All file processing requests are issued in the form of a call to the subroutine TIPFCS with an appropriate function code and associated parameters.

Following is a summary of the CALLs to the File Control System:

```
CALL 'TIPFCS' USING FCS-ACCESS,  file-pkt, FILE-DESCRIPTOR.  *
CALL 'TIPFCS' USING FCS-ADD,     file-pkt, WORK, KEY.
CALL 'TIPFCS' USING FCS-ASSIGN,  file-pkt, FILE-DESCRIPTOR.  *
CALL 'TIPFCS' USING FCS-BACK,    file-pkt.
CALL 'TIPFCS' USING FCS-CLOSE,   file-pkt.
CALL 'TIPFCS' USING FCS-CREATE,  file-pkt, FILE-DESCRIPTOR.  *
CALL 'TIPFCS' USING FCS-DELETE,  file-pkt, WORK, KEY.
CALL 'TIPFCS' USING FCS-ESETL,   file-pkt.
CALL 'TIPFCS' USING FCS-FLUSH,   file-pkt.
CALL 'TIPFCS' USING FCS-GET,     file-pkt, WORK [,KEY].
CALL 'TIPFCS' USING FCS-GETUP,   file-pkt, WORK [,KEY].
CALL 'TIPFCS' USING FCS-HOLD,    file-pkt, KEY.
CALL 'TIPFCS' USING FCS-JOURNAL, file-pkt, WORK.
CALL 'TIPFCS' USING FCS-NEXT,    file-pkt, WORK.
CALL 'TIPFCS' USING FCS-NOUP,    file-pkt  [,KEY].
CALL 'TIPFCS' USING FCS-OPEN,    file-pkt, FILE-DESCRIPTOR.
CALL 'TIPFCS' USING FCS-PUT,     file-pkt, WORK [,KEY].
CALL 'TIPFCS' USING FCS-RELEASE, file-pkt, KEY.
CALL 'TIPFCS' USING FCS-SCRATCH, file-pkt, FILE-DESCRIPTOR.  *
CALL 'TIPFCS' USING FCS-SETL,    file-pkt, KEY.
CALL 'TIPFCS' USING FCS-TREN,    file-pkt.
```

* This function supported for FCS DYNAMIC FILES only.

6.5 SUPPORTED FILE TYPES FCS: types

Many files types are supported by the File Control System. Certain file organizations are considered to be generic equivalents.

Supported file types are summarized below:

ISAM Indexed Sequential Access (one index)

MIRAM Similiar to ISAM but allows up to 5 indices.

DAM Direct MIRAM, memory, FCS dynamic files.

SAM Sequential MIRAM, PRNTR, PUNCH, TAPE.

All printer files have the same format. The output records must be standard variable length format and the first byte of data is taken to be the spacing control character. Refer to your data management manual for the 'device independent' (DI) printer spacing control codes.

6.6 CALL TIPFCS - COMMON PARAMETERS TIPFCS: params

Most calls to TIPFCS have the following format:

Syntax:

```
CALL 'TIPFCS' USING function,
                    file-pkt,
                    [record-area],
                    [key-value],
                    [index-num]
```

Where:

function is the TIPFCS function code. (Eg. FCS-GET).

file-pkt is the logical file name packet. This holds the logical name of the file (as it appears in the active file table).

record-area is a record work area. This will contain the record contents after a successful input request. This must also hold the record for output functions.

This record layout must define the record format as it is stored in the file.

key-value For an indexed file this will hold the record key.

May be omitted (as documented) for some function codes.

For a direct file this is a binary fullword which holds the relative record number (ie. PIC 9(6) COMP-4 SYNC).

index-num For multi-indexed files such as MIRAM, this will hodl the index number. It must be a binary halfword. (ie. PIC 9 COMP-4 SYNC).

If omitted, index 1 will be assumed.

6.7 FILE CONTROL SYSTEM INTERFACE PACKETS

There are two packets which are used to control processing of files through FCS:

- LOGICAL FILE NAME PACKET
- FILE DESCRIPTOR PACKET

6.7.1 LOGICAL FILE NAME PACKET — FCS: file-pkt

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

The first is the 8-byte Logical File Name assigned to the file for the program in use. Each file in use by a program must have a name unique to the associated terminal.

The second field in the File Name packet is a 1-byte status field where FCS stores the completion status of every file processing request made for the file.

The status codes are the same as those returned in the PIB.

This field is required to maintain downward compatability with earlier releases of TIP/30.

It is recommended that users reference the status as returned in the field PIB-STATUS.

Example:

```
05  PART-FILE.
    10  FILE-NAME              PICTURE X(8).
    10  PART-FILE-STATUS       PICTURE X.
```

-+*+-

6.7.2 FILE DESCRIPTOR PACKET — FCS: descriptor

This packet is used during a call to FCS using the OPEN function code. It establishes the relationship between a logical file name (LFN) and the real file to which I/O is to be done.

If opening a data management file with the logical file name the same as the real file name this packet may be omitted from the open request to TIPFCS.

```
        02 FILE-DESCRIPTOR.  COPY TC-FDES OF TIP.
000001*  TC$FDES COPY ELEMENT FOR TIP/30 FILE CONTROL INTERFACE
000002*
000003*************************************************************
000004*  THE FOLLOWING DATA ITEMS ARE REFERRED TO AS              *
000005*            THE FCS FILE DESCRIPTOR PACKET                 *
000006*************************************************************
000007      05  FDES-USER-ID                    PICTURE X(8).
000008      05  FDES-CATALOG                    PICTURE X(8).
000009      05  FDES-FILE-NAME                  PICTURE X(8).
000010      05  FDES-PASSWORD                   PICTURE X(8).
000011      05  FDES-FCS-CLASS                  PICTURE X.
000012      05  FDES-FCS-TYPE                   PICTURE X.
000013      05  FDES-FCS-PERM                   PICTURE X.
000014      05  FDES-FCS-LOCK                   PICTURE X.
```

Where:

FDES-USER-ID may contain the USER-ID or Group name to which the file belongs. If opened for read a complete search of the catalogue is done. If opened for output the specified value is used. If creating a dynamic file, this will be set to the callers USER-ID.

FDES-CATALOG additional level of naming provided for dynamic files. If left as spaces this will be set to the FDES-FILE-NAME.

FDES-FILE-NAME file name for dynamic files. The catalogued logical file name of data management files. If left as spaces, this is set to the logical file name.

FDES-PASSWORD this field is no longer used and may be left as spaces.

FDES-FCS-CLASS class of file to be opened. If left as a space TIPFCS will open the first file that it can find in the catalog with the supplied name.

'E' is an Editor dynamic file.

'P' is a permanent dynamic file.

'S' is a data management file.

'T' is a temporary dynamic file.

FDES-FCS-TYPE designates type of file (or element) wanted

'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 for updating (ie. SAM)

' ' - read/write

FDES-FCS-LOCK Is exclusive use of file wanted?

'Y' - obtain exclusive use of file. A LOCKED status is returned if any other process has the file in use.

-+*+-

6.8 'TIPFCS' FOR INDEXED FILES FCS: indexed

Indexed files include ISAM, single index MIRAM, and multi index MIRAM. IRAM files are handled as single index MIRAM.

MULTI-INDEXED FILES

MIRAM files may be accessed by user programs via any index. The philosophy taken is that the program may read a MIRAM file by any index but all updates should be done via index one (1); furthermore, the primary index (1) should not allow duplicate keys.

To process the file by a secondary index follow the KEY parameter on the TIPFCS call with a halfword parameter specifying the index number.

Example:

```
05  INDEX-NUM      PIC 9 SYNC COMP-4.
```

Summary of TIPFCS calls using the index number.

```
CALL 'TIPFCS' USING FCS-GET,  file-pkt, record, key, index-num.
CALL 'TIPFCS' USING FCS-SETL, file-pkt, key, index-num.
CALL 'TIPFCS' USING FCS-NEXT, file-pkt, record, key, index-num.
```

If the INDEX-NUM parameter is omitted the the primary index is assumed. Once the file is put in sequential mode all GET's until an ESETL would be via the INDEX-NUM supplied on the SETL.

MIRAM AND DUPLICATE KEYS

MIRAM allows a file to contain duplicate keys. This may present a problem when you want to display several records from a file via some secondary index which contains duplicates.

After the ESETL, the next SETL will restart at the beginning of the set of duplicates.

A recommended approach is to save the primary key of the last record processed (ie: displayed on the terminal). If continuation is requested, issue the SETL again and skip over records until you get to the one with the primary key that was saved and continue from that point.

When you design your MIRAM file always define the primary index as "no duplicates" (NDUP). This index may then reliably be used to uniquely identify records in the file.

6.8.1 INDEXED: ADD RECORD TO FILE FCS-ADD

This function code will add a new record to a file.

Syntax:

CALL 'TIPFCS' USING FCS-ADD, file-pkt, record [,key]

Where:

FCS-ADD	function code from the TC-FCS copy book.
file-pkt	logical file name packet.
record	record area containing new record data.
key	Key of the record. If omitted the key will be taken from the record area.

Example:

```
CALL 'TIPFCS' USING FCS-ADD, MST-FILE, MST-REC, MST-KEY.
```

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	some I/O error occurred on the disk.

-+*+-

6.8.2 INDEXED: ROLL BACK UPDATES FCS-BACK

For files generated as HOLD=TR, all updates since the last transaction end are rolled back. A transaction end is marked by an input message arrival, prior use of FCS-BACK, or use of FCS-TREN.

Syntax:

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

Where:

FCS-BACK function code from the TC-FCS copy book.

file-pkt Logical file name packet.

Example:

```
CALL 'TIPFCS' USING FCS-BACK, MSTR-FILE.
```

Error Conditions:

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

Additional Considerations:

Use of this function will force the file to random access mode.

-+*+-

6.8.3 INDEXED: CLOSE FILE FCS-CLOSE

Remove the file from use by programs at the associated terminal. The entry is removed from the Active File Table. If there are no other users of the file AND the file was generated with OPEN=NO, FCS will issue the CLOSE imperative macro to Data Management.

Syntax:

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

Where:

FCS-CLOSE function code from the TC-FCS copy book.

file-pkt Logical file name packet.

Example:

```
CALL 'TIPFCS' USING FCS-CLOSE, PAYFILE.
```

Error Conditions:

PIB-FUNCTION file is not assigned to the program.

Additional Considerations:

This function should only be issued for files which were opened by issuing and FCS-OPEN function code.

-+*+-

6.8.4 INDEXED: DELETE RECORD FCS-DELETE

Flag a record with the delete code as specified for the file when TIP/30 was generated. For MIRAM files with DELETE=RCB the record will be marked deleted using the MIRAM delete facility; for non-RCB delete schemes, the record will be appropriately flagged.

Syntax:

```
CALL 'TIPFCS' USING FCS-DELETE, file-pkt, record [,key].
```

Where:

FCS-DELETE function code from the TC-FCS copy book.

file-pkt Logical file name packet.

record record area.

key record key. If omitted the key will be taken from the record area.

Example:

```
CALL 'TIPFCS' USING FCS-DELETE, MST-FILE, MST-REC, MST-KEY.
```

Error Conditions:

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

PIB-IO-ERROR some I/O error occurred on the disk.

PIB-NOT-HELD an FCS-GETUP was not successfully done for this record or the record hold has been released.

Additional Considerations:

Note that records are not physically deleted but flagged with a known value. Record space is never re-used by data management. You should periodically re-organize files which have had a lot of delete/add activity.

Records that have been flagged as deleted (except MIRAM RCB deleted) are actually returned to the user record area on a GET function. The status is set to PIB-NOT-FOUND, but the record contents are available in the specified record area. A previously deleted record could be reincarnated by turning off the delete flag and issuing an FCS-ADD function.

-+*+-

6.8.5 INDEXED: END SEQUENTIAL PROCESSING FCS-ESETL

Set the file back to random processing mode. When an indexed file is in sequential mode, it is effectively locked. Other user programs must wait for the file to be released before they may access it.

It is important that the file be set back to random mode prior to soliciting any terminal input, since failure to do so locks the file while your program is awaiting terminal input.

Syntax:

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

Where:

FCS-ESETL function code from the TC-FCS copy book.

file-pkt Logical file name packet.

Example:

```
CALL 'TIPFCS' USING FCS-ESETL, MST-FILE.
```

Error Conditions:

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

-+*+-

6.8.6 INDEXED: FLUSH FILE FCS-FLUSH

Causes the file buffers to be written to disk and the file to be physically closed then immediately re-opened.

This function is used primarily to force the VTOC end-of-data pointers to be updated after records have been added to a file.

The FCS-FLUSH should be used with discretion since it is a relatively time consuming operation that makes the file inaccessible to everyone for a few seconds.

Syntax:

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

Where:

FCS-FLUSH function code from the TC-FCS copy book.

file-pkt Logical file name packet.

Example:

```
CALL 'TIPFCS' USING FCS-FLUSH, MASTR-FILE.
```

Error Conditions:

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

PIB-IO-ERROR some I/O error occurred on the disk.

-+*+-

6.8.7 INDEXED: READ RECORD FCS-GET

Read the record with the specified key from the file.

To get the next sequential record from an indexed file the file must have been previously set to sequential mode via an FCS-SETL call.

Using FCS-GET means that the record is not held for update.

Syntax:

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

Where:

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

file-pkt Logical file name packet.

record Record area

key record key. If omitted the key will be taken from the record area. If the file is in sequential mode then the key is not required.

index-num binary halfword holding the index number. If omitted then index 1 is assumed.

Example:

```
CALL 'TIPFCS' USING FCS-GET, MST-FILE, MST-REC, MST-KEY.
```

Error Conditions:

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

PIB-IO-ERROR some I/O error occurred on the disk.

PIB-EOF end of file was reached during sequential processing

PIB-NOT-FOUND the record does not exist or it has been flagged deleted.

Additional Considerations:

If the file was set in sequential mode on a secondary index the FCS-GET will process the file sequentially via that index.

-+*+-

6.8.8 INDEXED: READ RECORD AND LOCK FCS-GETUP

Read the record with the specified key with intent to update. The record is HELD for update.

Syntax:

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

Where:

FCS-GETUP function code from the TC-FCS copy book.

file-pkt Logical file name packet.

record Record area

key record key. If omitted the key will be taken from the record area.

index-num binary halfword holding the index number. If omitted then index 1 is assumed.

Example:

```
CALL 'TIPFCS' USING FCS-GETUP MST-FILE, MST-REC, MST-KEY.
```

Error Conditions:

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

PIB-IO-ERROR some I/O error occurred on the disk.

PIB-NOT-FOUND the record does not exist or it has been flagged deleted.

PIB-HELD the record is currently being updated by some other program. Try again later if you like.

PIB-WRONG you have issued an FCS-GETUP to a file which is not set for random processing.

Additional Considerations:

If the GETUP is done via a secondary index, the key of primary index is placed in the key holding table.

-+*+-

6.8.9 INDEXED: HOLD RESOURCE FCS-HOLD

Place a user defined value in the key holding table. Programming conventions may be adopted to use this function to lock all or some portion of a file instead of just one record.

Syntax:

```
CALL 'TIPFCS' USING FCS-HOLD, file-pkt, value.
```

Where:

FCS-HOLD function code from the TC-FCS copy book.

file-pkt Logical file name packet.

value this must be a four character field.

Example:

```
CALL 'TIPFCS' USING FCS-HOLD, MST-FILE, WE-AGREE.
```

Error Conditions:

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

PIB-HELD the key value is currently held by some other program. Try again later if you like.

-+*+-

6.8.10 INDEXED: GET NEXT RECORD FCS-NEXT

This function code does the equivalent of FCS-SETL, FCS-GET and FCS-ESETL. FCS-NEXT should be used when only one record is required at a time.

Syntax:

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

Where:

FCS-NEXT function code from the TC-FCS copy book.

file-pkt Logical file name packet.

record record area

key record key. If omitted the key will be taken from the record area.

index-num binary halfword holding the index number. If omitted then index 1 is assumed.

Example:

```
CALL 'TIPFCS' USING FCS-NEXT MST-FILE, MST-REC, MST-KEY.
```

Error Conditions:

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

PIB-IO-ERROR some I/O error occurred on the disk.

PIB-NOT-FOUND the next record does not exist.

PIB-WRONG you have issued an FCS-NEXT to a file which is not set for random processing.

Additional Considerations:

This function is time consuming. If the intent is to read several records before outputing a display to the terminal then it would be better to issue SETL, GET, GET, ..., ESETL.

-+*+-

6.8.11 INDEXED: CANCEL UPDATE FCS-NOUP

Release the record previously held for update. The program should avoid locking a record until it is certain the record will be updated. This limits the time the record is unavailable to other users.

A record that has been held for update via an FCS-GETUP call is automatically released when the record is updated (FCS-PUT).

If the held record will not be updated it should be released by issuing an FCS-NOUP.

Syntax:

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

Where:

FCS-NOUP function code from the TC-FCS copy book.

file-pkt Logical file name packet.

key Record key. If omitted all records held for this file are released.

Example:

```
CALL 'TIPFCS' USING FCS-NOUP, MST-FILE.
```

Error Conditions:

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

PIB-NOT-HELD the record was not held.

PIB-WRONG you have issued an FCS-NOUP to a file which is not set for random processing.

-+*+-

6.8.12 INDEXED: OPEN FILE FCS-OPEN

Make the specified file available for processing by programs at the calling terminal. An entry is made in the Active File Table for the file. If there are no other users of the file and the file was generated with OPEN=NO, TIPFCS will issue the Data Management OPEN. Set FDES-FILENAME of FILE-DESCRIPTOR to the catalogued logical file name. Set FILE-NAME of file-pkt to the logical file name to be used.

The optional fourth parameter is used to change the logical file name (as known to the operating system) before opening the file.

Syntax:

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

Where:

FCS-~~GETUP~~ (OPEN) function code from the TC-FCS copy book.

file-pkt Logical file name packet.

file-desc file descriptor packet. See the TC-FDES copy book. If omitted the name in the file-pkt will be used to build a file descriptor.

alt-lfd Optional eight character field used to permanently change the LFD name associated with this file.

Example:

```
CALL 'TIPFCS' USING FCS-OPEN, MST-FILE.
```

Error Conditions:

PIB-IO-ERROR some I/O error occurred while opening the file.

PIB-DUP-AFT-NAME a file with the same name given in file-pkt is already assigned to the terminal.

-+*+-

6.8.13 INDEXED: UPDATE RECORD FCS-PUT

Rewrite a record that was obtained by a previous FCS-GETUP.

Syntax:

```
CALL 'TIPFCS' USING FCS-PUT, file-pkt, record  [,key].
```

Where:

FCS-PUT	function code from the TC-FCS copy book.
file-pkt	Logical file name packet.
record	record area
key	Record key. If omitted the key will be taken from the record area.

Example:

```
CALL 'TIPFCS' USING FCS-PUT MST-FILE, MST-REC, MST-KEY.
```

Error Conditions:

PIB-FUNCTION	the file is not assigned to the program.
PIB-IO-ERROR	some I/O error occurred on the disk.
PIB-NOT-HELD	the record was not held and therefore can not be updated.

-+*+-

6.8.14 INDEXED: RELEASE RESOURCE FCS-RELEASE

Release an entry previously held by FCS-HOLD.

Syntax:

```
CALL 'TIPFCS' USING FCS-RELEASE, file-pkt, key.
```

Where:

FCS-RELEASE function code from the TC-FCS copy book.

file-pkt Logical file name packet.

key this can only be a four character field.

Example:

```
CALL 'TIPFCS' USING FCS-RELEASE, MST-FILE, WE-AGREE.
```

Error Conditions:

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

PIB-NOT-HELD the key value was not held.

-+*+-

6.8.15 INDEXED: SET SEQUENTIAL MODE FCS-SETL

This will set the file for sequential processing beginning with the next record with a key greater than or equal to that given.

If the file is currently in sequential mode for another user TIPFCS will queue the request until the file is set back to random mode before issuing the SETL.

Care should be taken to insure that an ESETL is issued to the file prior to requesting any input from the terminal otherwise programs may be locked in the process.

Syntax:

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

Where:

FCS-SETL function code from the TC-FCS copy book.

file-pkt Logical file name packet.

key Record key. If omitted then processing begins with the first record in the file.

index-num binary halfword holding the index number. If omitted then index 1 is assumed.

Example:

```
CALL 'TIPFCS' USING FCS-SETL, MST-FILE, MST-ACCT.
```

Error Conditions:

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

PIB-IO-ERROR some I/O error occurred on the disk.

PIB-NOT-FOUND the record does not exist or it has been flagged deleted.

PIB-EOF end of file reached.

-+*+-

6.8.16 INDEXED: SET SEQUENTIAL MODE FCS-SETL-EQ

This will set the file for sequential processing beginning with the record with a key equal to that given.

This function code will be treated as FCS-SETL for ISAM files.

If the file is currently in sequential mode for another user TIPFCS will queue the request until the file is set back to random mode before issuing the SETL.

Care should be taken to ensure that an ESETL is issued to the file prior to requesting any input from the terminal otherwise programs may be locked in the process.

Syntax:

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

Where:

FCS-SETL-EQ function code from the TC-FCS copy book.

file-pkt Logical file name packet.

key Record key. If omitted then processing begins with the first record in the file.

index-num binary halfword holding the index number. If omitted then index 1 is assumed.

Example:

```
CALL 'TIPFCS' USING FCS-SETL-EQ, MST-FILE, MST-ACCT.
```

Error Conditions:

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

PIB-IO-ERROR some I/O error occurred on the disk.

PIB-NOT-FOUND the record does not exist or it has been flagged deleted.

PIB-EOF end of file reached.

-+*+-

6.8.17 INDEXED: SET SEQUENTIAL MODE FCS-SETL-GT

This will set the file for sequential processing beginning with the next record with a key greater than that given. If the file is currently in sequential mode for another user TIPFCS will queue the request until the file is set back to random mode before issuing the SETL. Care should be taken to insure that an ESETL is issued to the file prior to requesting any input from the terminal otherwise programs may be locked in the process.

Syntax:

```
CALL 'TIPFCS' USING FCS-SETL-GT, file-pkt, [,key[,index-num]].
```

Where:

FCS-SETL-GT function code from the TC-FCS copy book.

file-pkt Logical file name packet.

key Record key. If omitted then processing begins with the first record in the file.

index-num binary halfword holding the index number. If omitted then index 1 is assumed.

Example:

```
CALL 'TIPFCS' USING FCS-SETL-GT, MST-FILE, MST-ACCT.
```

Error Conditions:

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

PIB-IO-ERROR some I/O error occurred on the disk.

PIB-NOT-FOUND the record does not exist or it has been flagged deleted.

PIB-EOF end of file reached.

-+*+-

6.8.18 INDEXED: MARK TRANSACTION END FCS-TREN

If a file was generated with 'hold for transaction' (HOLD=TR), then TIP/30 will automatically roll back any updates if the program aborts. If a program has updated record(s) and wants to signal transaction end, thus preventing rollback if a subsequent abort occurs, the program would use the FCS-TREN function.

Syntax:

```
CALL 'TIPFCS' USING FCS-TREN, file-pkt.
```

Where:

FCS-TREN function code from the TC-FCS copy book.

file-pkt Logical file name packet.

This may be any file currently used by the program.

Example:

```
CALL 'TIPFCS' USING FCS-TREN, DUMMY.
```

Error Conditions:

None.

Additional Considerations:

If the program has updated records an automatic FCS-TREN is issued when the program calls TIPMSGI or TIPTERM with a get function.

-+*+-

6.9 'TIPFCS' FOR DIRECT ACCESS FILES FCS: direct

Direct access files include DAM, and direct MIRAM. Direct IRAM is handled by direct MIRAM.

All records are processed by the relative record number. Record number 1 is the first record in the file and so on.

In the case of DAM, the relative record number is also the block number.

Direct MIRAM does support blocked files. The reading of a relative record may result in an entire block of records being read into memory by data management.

In all cases the key passed to 'TIPFCS' is a binary fullword holding the relative record number of the record to be processed.

Example:

```
05  REC-NUM          PICTURE 9(6) COMP-4 SYNC.
      .....
MOVE 14 TO REC-NUM.
CALL 'TIPFCS' USING FCS-GET, DETL-FILE, DETL-REC, REC-NUM.
IF NOT PIB-GOOD
    PERFORM CHECK-ERROR.
```

6.9.1 DIRECT: ADD RECORD FCS-ADD

This could be used to add new records to the end of a direct access file or to add a new record to the file in the place of one which had been logically flagged deleted.

Syntax:

```
CALL 'TIPFCS' USING FCS-ADD, file-pkt, record, rec-num.
```

Where:

FCS-ADD	function code from the TC-FCS copy book.
file-pkt	Logical file name packet.
record	record area
rec-num	binary fullword holding the relative record number.

Example:

```
CALL 'TIPFCS' USING FCS-ADD, DTL-FILE, DTL-REC, DTL-NUM.
```

Error Conditions:

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

Additional Considerations:

If the record number given is greater than the number of records in the file, the record will be added at end of file and its relative record number will be passed back in 'rec-num'.

-+*+-

6.9.2 DIRECT: ROLL BACK UPDATES FCS-BACK

For files generated as HOLD=TR, all updates since the last transaction end are rolled back. A transaction end is marked by an input message arrival, prior use of FCS-BACK, or use of FCS-TREN.

Syntax:

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

Where:

FCS-BACK function code from the TC-FCS copy book.

file-pkt Logical file name packet.

Example:

```
CALL 'TIPFCS' USING FCS-BACK, MSTR-FILE.
```

Error Conditions:

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

Additional Considerations:

Use of this function will force the file to random access mode.

-+*+-

6.9.3 DIRECT: CLOSE FILE FCS-CLOSE

Remove the file from use by programs at the associated terminal. The entry is removed from the Active File Table. If there are no other users of the file and the file was generated with OPEN=NO, FCS will issue the CLOSE imperative macro to Data Management.

Syntax:

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

Where:

FCS-CLOSE function code from the TC-FCS copy book.

file-pkt Logical file name packet.

Example:

```
CALL 'TIPFCS' USING FCS-CLOSE, PAYFILE.
```

Error Conditions:

PIB-FUNCTION file is not assigned to the program.

Additional Considerations:

An FCS-CLOSE should only be issued to files which the program did an FCS-OPEN.

-+*+-

6.9.4 DIRECT: DELETE RECORD FCS-DELETE

Flag a record with the delete code as specified for the file when TIP/30 was generated.

Syntax:

```
CALL 'TIPFCS' USING FCS-DELETE, file-pkt, record, rec-num.
```

Where:

FCS-DELETE function code from the TC-FCS copy book.

file-pkt Logical file name packet.

record record area

rec-num binary fullword holding the relative record number.

Example:

```
CALL 'TIPFCS' USING FCS-DELETE, DTL-FILE, DTL-REC, DTL-NUM.
```

Error Conditions:

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

PIB-IO-ERROR some I/O error occurred on the disk.

PIB-NOT-HELD an FCS-GETUP was not successfully done for this record or the record hold has been released.

Additional Considerations:

Note that the record is not physically deleted but flagged with a known value. The record space is never re-used by data management. You should periodically re-organize files which have had a lot of delete/add activity.

Logically deleted records are still returned to user programs (when read) with a status of PIB-NOT-FOUND.

-+*+-

6.9.5 DIRECT: FLUSH FILE FCS-FLUSH

Causes the file buffers to be written to disk and the file to be physically closed then immediately re-opened. This is used primarily to update the VTOC end-of-data pointers. The FCS-FLUSH should be used with discretion since it is quite a time consuming operation making the file inaccessible to everyone for a few seconds.

Syntax:

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

Where:

FCS-FLUSH function code from the TC-FCS copy book.

file-pkt Logical file name packet.

Example:

```
CALL 'TIPFCS' USING FCS-FLUSH, MASTR-FILE.
```

Error Conditions:

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

PIB-IO-ERROR some I/O error occurred on the disk.

-+*+-

6.9.6 DIRECT: READ RECORD FCS-GET

Read the specified record from the file. The relative record number of the desired record must be supplied.

The record is not held for update.

Syntax:

```
CALL 'TIPFCS' USING FCS-GET, file-pkt, record, rec-num.
```

Where:

FCS-GET	function code from the TC-FCS copy book.
file-pkt	Logical file name packet.
record	record area
rec-num	binary fullword holding the relative record number.

Example:

```
CALL 'TIPFCS' USING FCS-GET, MST-FILE, MST-REC, MST-KEY.
```

Error Conditions:

PIB-FUNCTION	the file is not assigned to the program.
PIB-IO-ERROR	some I/O error occurred on the disk.
PIB-EOF	end of file was reached during sequential processing
PIB-NOT-FOUND	the record does not exist or it has been flagged deleted.

-+*+-

6.9.7 DIRECT: READ RECORD AND LOCK FCS-GETUP

Read the relative record with intent to update. The record is HELD for update.

Syntax:

```
CALL 'TIPFCS' USING FCS-GETUP, file-pkt, record, rec-num.
```

Where:

FCS-GETUP	function code from the TC-FCS copy book.
file-pkt	Logical file name packet.
record	record area
rec-num	binary fullword holding the relative record number.

Example:

```
CALL 'TIPFCS' USING FCS-GETUP, DTL-FILE, DTL-REC, DTL-NUM.
```

Error Conditions:

PIB-FUNCTION	the file is not assigned to the program.
PIB-IO-ERROR	some I/O error occurred on the disk.
PIB-NOT-FOUND	the record does not exist or it has been flagged deleted.
PIB-HELD	the record is currently being updated by some other program. Try again later if you like.
PIB-WRONG	you have issued an FCS-GETUP to a file which is not set for random processing.

-+*+-

6.9.8 DIRECT: HOLD RESOURCE FCS-HOLD

Place a user defined value in the key holding table. Programming conventions may be adopted to use this function to secure all or some portion of a file instead of just one record.

Syntax:

```
CALL 'TIPFCS' USING FCS-HOLD, file-pkt, key.
```

Where:

FCS-HOLD function code from the TC-FCS copy book.

file-pkt Logical file name packet.

key this can only be a four character field.

Example:

```
CALL 'TIPFCS' USING FCS-HOLD, MST-FILE, WE-AGREE.
```

Error Conditions:

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

PIB-HELD the key value is currently held by some other program. Try again later if you like.

-+*+-

6.9.9 DIRECT: CANCEL UPDATE FCS-NOUP

Release the record held for update. The program should avoid locking a record until it is certain the record will be updated. This limits the time the record is unavailable to other users. A record that has been held for update via an FCS-GETUP call is automatically released when the record is updated (FCS-PUT). If the held record will not be updated it should be released by issuing an FCS-NOUP call.

Syntax:

```
CALL 'TIPFCS' USING FCS-NOUP, file-pkt, [,rec-num].
```

Where:

FCS-NOUP function code from the TC-FCS copy book.

file-pkt Logical file name packet.

rec-num binary fullword holding the relative record number

Example:

```
CALL 'TIPFCS' USING FCS-NOUP, DTT-FILE.
```

Error Conditions:

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

PIB-NOT-HELD the record was not held.

PIB-WRONG you have issued an FCS-NOUP to a file which is not set for random processing.

-+*+-

6.9.10 DIRECT: OPEN FILE FCS-OPEN

Make the specified file available for processing by programs at the calling terminal. An entry is made in the Active File Table for the file. If there are no other users of the file and the file was generated with OPEN=NO, TIPFCS will issue the Data Management OPEN. Set FDES-FILENAME of FILE-DESCRIPTOR to cataloged logical file name. Set FILE-NAME of file-pkt to the logical file name to be used.

The optional fourth parameter is used to change the logical file name (as known to the operating system) before opening the file.

Syntax:

```
CALL 'TIPFCS' USING FCS-OPEN, file-pkt [,file-desc
                          [,alt-lfd]].
```

Where:

FCS-OPEN function code from the TC-FCS copy book.

file-pkt Logical file name packet.

file-desc file descriptor packet. See the TC-FDES copy book. If omitted the name in the file-pkt will be used to build a file description.

alt-lfd Optional eight character field used to permanently change the LFD name used by this file.

Example:

```
CALL 'TIPFCS' USING FCS-OPEN, MST-FILE.
```

Error Conditions:

PIB-IO-ERROR some I/O error occurred while opening the file.

PIB-DUP-AFT-NAME a file of with the same name given in file-pkt is already assigned to the terminal.

-+*+-

6.9.11 DIRECT: UPDATE RECORD FCS-PUT

The program must have previously issued an FCS-GETUP to lock the record to be written.

Syntax:

CALL 'TIPFCS' USING FCS-PUT, file-pkt, record, rec-num.

Where:

FCS-PUT	function code from the TC-FCS copy book.
file-pkt	Logical file name packet.
record	record area
rec-num	binary fullword holding the relative record number.

Example:

CALL 'TIPFCS' USING FCS-PUT, DTL-FILE, DTL-REC, DTL-NUM.

Error Conditions:

PIB-FUNCTION	the file is not assigned to the program.
PIB-IO-ERROR	some I/O error occurred on the disk.
PIB-NOT-HELD	the record was not held and therefore can not be updated.

-+*+-

6.9.12 DIRECT: RELEASE RESOURCE FCS-RELEASE

Release an entry previously held by FCS-HOLD.

Syntax:

```
CALL 'TIPFCS' USING FCS-RELEASE, file-pkt, key.
```

Where:

FCS-RELEASE function code from the TC-FCS copy book.

file-pkt Logical file name packet.

key this can only be a four character field.

Example:

```
CALL 'TIPFCS' USING FCS-RELEASE, MST-FILE, WE-AGREE.
```

Error Conditions:

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

PIB-NOT-HELD the key value was not held.

-+*+-

6.9.13 DIRECT: MARK TRANSACTION END FCS-TREN

If a file was generated with 'hold for transaction' (HOLD=TR), then TIP/30 will automatically rollback any updates when a user program has aborted. If a program has updated record(s) and wants to signal transaction end, thus preventing rollback if a subsequent abort should occur, then the program would issue the FCS-TREN function code to TIPFCS.

Syntax:

```
CALL 'TIPFCS' USING FCS-TREN, file-pkt.
```

Where:

FCS-TREN function code from the TC-FCS copy book.

file-pkt Logical file name packet. This may be any file used by the program.

Example:

```
CALL 'TIPFCS' USING FCS-TREN, DUMMY.
```

Error Conditions:
None.

Additional Considerations:

If the program has updated records an automatic FCS-TREN is issued when the program calls TIPMSGI or TIPTERM USING GET.

-+*+-

6.10 'TIPFCS' FOR SEQUENTIAL FILES FCS: sequential

Sequential files include SAM, sequential MIRAM, TAPE, PUNCH (cards), and printers. Sequential IRAM files are handled as sequential MIRAM.

Printer files all follow the standard variable length record format with device independent carriage control as documented in the Univac data management guide. You may wish to refer to the section on TIPPRINT for more information regarding printer files.

6.10.1 SEQ: CLOSE FILE FCS-CLOSE

Remove the file from use by programs at the associated terminal. The entry is removed from the Active File Table. If there are no other users of the file and the file was generated with OPEN=NO, FCS will issue the CLOSE imperative macro to Data Management.

Syntax:

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

Where:

FCS-CLOSE function code from the TC-FCS copy book.

file-pkt Logical file name packet.

Example:

```
CALL 'TIPFCS' USING FCS-CLOSE, PAYFILE.
```

Error Conditions:

PIB-FUNCTION file is not assigned to the program.

Additional Considerations:

An FCS-CLOSE should only be issued for files which were opened via an FCS-OPEN.

In the case of printer and punch file a breakpoint will be issued at CLOSE time.

-+*+-

6.10.2 SEQ: READ RECORD FCS-GET

Read the next record from the file.

Syntax:

CALL 'TIPFCS' USING FCS-GET, file-pkt, record.

Where:

FCS-GET	function code from the TC-FCS copy book.
file-pkt	Logical file name packet.
record	record area

Example:

CALL 'TIPFCS' USING FCS-GET, MST-FILE, MST-REC.

Error Conditions:

PIB-FUNCTION	the file is not assigned to the program.
PIB-IO-ERROR	some I/O error occurred on the disk.
PIB-EOF	end of file was reached during sequential processing
PIB-WRONG	the file was not defined for input processing.

-+*+-

6.10.3 SEQ: OPEN FILE — FCS-OPEN

Make the specified file available for processing by programs at the calling terminal. An entry is made in the Active File Table for the file. If there are no other users of the file and the file was generated with OPEN=NO, TIPFCS will issue the Data Management OPEN. Set FDES-FILENAME of FILE-DESCRIPTOR to cataloged logical file name. Set FILE-NAME of file-pkt to the logical file name to be used.

The optional fourth parameter is used to change the logical file name (as known to the operating system) before opening the file.

Syntax:

```
CALL 'TIPFCS' USING FCS-OPEN, file-pkt [,file-desc
                                [,alt-lfd]].
```

Where:

FCS-OPEN function code from the TC-FCS copy book.

file-pkt Logical file name packet.

file-desc file descriptor packet. See the TC-FDES copy book. If omitted the name in the file-pkt will be used to build a file description.

alt-lfd an eight character field used to permanently change the LFD name used by this file.

Example:

```
CALL 'TIPFCS' USING FCS-OPEN, MST-FILE.
```

Error Conditions:

PIB-IO-ERROR some I/O error occurred while opening the file.

PIB-DUP-AFT-NAME a file of with the same name given in file-pkt is already assigned to the terminal.

-+*+-

6.10.4 SEQ: OUTPUT RECORD FCS-PUT

The record is appended to the sequential output file.

Syntax:

```
CALL 'TIPFCS' USING FCS-PUT, file-pkt, record.
```

Where:

FCS-PUT	function code from the TC-FCS copy book.
file-pkt	Logical file name packet.
record	record area

Example:

```
CALL 'TIPFCS' USING FCS-PUT, MST-FILE, MST-REC.
```

Error Conditions:

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

-+*+-

6.11 DYNAMIC FCS FILES FCS: dynamic

Each Dynamic FCS file may be processed by several users at the same time. Within the application program, FCS files have a complete description as mapped out by a FILE-DESCRIPTOR packet.

Associated with each active file is a Logical File Name which is defined when the file is accessed or created. The following are the functions that may be used to process FCS Dynamic files:

FCS-ACCESS	open an existing file
FCS-ASSIGN	open the file, if not there create it
FCS-CLOSE	close a file
FCS-CREATE	create a new file
FCS-GET	read a record(s) from the file
FCS-PUT	write a record(s) to the file
FCS-SCRATCH	scratch a file

NOTE:

- The size of the I/O work area must always be a multiple of 512 bytes (the physical block size of dynamic files) and must always be fullword aligned.

6.11.1 DYN: ACCESS FILE FCS-ACCESS

Before an application program can perform I/O to an existing file, the file must be allocated to the program. FCS-ACCESS is the function to use when opening a Dynamic file. Set USER-ID, CATALOG and file-pkt of FILE-DESCRIPTOR to the appropriate values. Also, any security data that was specified when the file was created may be required.

Syntax:

```
CALL 'TIPFCS' USING FCS-ACCESS, file-pkt, file-desc.
```

Where:

FCS-ACCESS function code from the TC-FCS copy book.

file-pkt File descriptor packet.

file-desc File descriptor packet. See the TC-FDES copy book.

Example:

```
CALL 'TIPFCS' USING FCS-ACCESS, BATCH-FILE, BATCH-FDES.
```

Error Conditions:

PIB-DUP-AFT a file of the name given in the file-pkt is already assigned to the terminal.

PIB-NOT-FOUND the requested file does not exist.

-+*+-

6.11.2 DYN: ASSIGN FILE FCS-ASSIGN

This FCS call will open an existing Dynamic file for use by the calling program. If the file does not exist FCS 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 book.
file-pkt	File descriptor packet.
file-desc	File descriptor packet. See the TC-FDES copy book.

Example:

```
CALL 'TIPFCS' USING FCS-ASSIGN, BATCH-FILE, BATCH-FDES.
```

Error Conditions:

PIB-DUP-AFT	a file of the name given in the file-pkt is already assigned to the terminal.
PIB-NOT-FOUND	the requested file does not exist.

-+*+-

6.11.3 DYN: CLOSE FILE FCS-CLOSE

When an application program is finished with a file it should remove the file from the Active File Table by issuing an FCS-CLOSE.

Syntax:

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

Where:

FCS-CLOSE function code from the TC-FCS copy book.

file-pkt Logical file name packet.

Example:

```
CALL 'TIPFCS' USING FCS-CLOSE, BATCH-FILE.
```

Error Conditions:

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

-+*+-

6.11.4 DYN: CREATE FILE FCS-CREATE

This function is used to create new files, either temporary or permanent. The application program must set up FILE-DESCRIPTOR and file-pkt with the correct values. If FDES-USER-ID is SPACES, then FCS will use the USER-ID that was given when the user logged on to TIP/30. If FDES-CATALOG is SPACES, FCS will build a unique name from the terminal-id and program stack level.

Set FDES-FCS-TYPE to 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 if you wish to have exclusive use of the file. If FILE-LOCK is equal to 'Y' then no other application will be allowed to access the file.

FCS will create a new file as specified if no errors were detected. The user program should check the status field of the file-pkt packet after every TIPFCS call.

Syntax:

```
CALL 'TIPFCS' USING FCS-CREATE, file-pkt, file-desc.
```

Where:

FCS-CREATE function code from the TC-FCS copy book.

file-pkt File descriptor packet.

file-desc File descriptor packet. See the TC-FDES copy book.

Example:

```
CALL 'TIPFCS' USING FCS-CREATE, BATCH-FILE, BATCH-FDES.
```

Error Conditions:

PIB-DUP-AFT a file of the name given in the file-pkt is already assigned to the terminal.

PIB-NOT-FOUND the requested file does not exist.

-+*+-

6.11.5 DYN: READ RECORD(S) FCS-GET

FCS Dynamic files are direct access type files. The user program must place the relative record number in RECORD-NUMBER. If the optional parameter RECORD-COUNT is not specified, FCS will read one 512 byte record into WORK.

Both RECORD-NUMBER and RECORD-COUNT must be fullword aligned and contain fullword binary values. [ie: 9(6) COMP-4, SYNC.]

Syntax:

```
CALL 'TIPFCS' USING FCS-GET, file-pkt, record, rec-num,
                                [,rec-count]
```

Where:

FCS-GET	function code from the TC-FCS copy book.
file-pkt	logical file name packet.
record	(512 x rec-count) byte record area. (Fullword aligned).
rec-num	Block number to be read. (Fullword aligned).
rec-count	Optional parameter which specifies how many blocks are to be read. Default is one.

Example:

```
CALL 'TIPFCS' USING FCS-GET, BATCH-FLE, REC-WRK, REC-NUM.
```

-+*+-

6.11.6 DYN: OPEN FILE FCS-OPEN

This function may be used to open any file. The file descriptor and any existing catalogue record are used to determine what type of file is to be opened. This function may create new files - either temporary or permanent. The application program must set up FILE-DESCRIPTOR and file-pkt with the correct values. If FDES-USER-ID is SPACES, then FCS will use the USER-ID that was given when the user logged on to TIP/30. If FDES-CATALOG is SPACES, FCS will build a unique name from the terminal-id and program stack level.

To open an existing file set FDES-FCS-TYPE to FCS-TYPE-OLD. To create a new file set FDES-FCS-TYPE to FCS-TYPE-NEW. If the FDES-FCS-TYPE is left as a space and the file exists then it will be accessed. If the file did not exist then it would be created. 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. Set FDES-FILE-CLASS to FCS-CLASS-PERM or FCS-CLASS-TEMP for permanent or temporary file. Set FDES-FCS-LOCK to FCS-LOCK-YES if you wish to have exclusive use of the file. If FILE-LOCK is equal to 'Y' then no other application will be allowed to access the file.

Syntax:

```
CALL 'TIPFCS' USING FCS-OPEN, file-pkt, file-desc.
```

Where:

FCS-OPEN	function code from the TC-FCS copy book.
file-pkt	Logical file name packet.
file-desc	file description packet. See the TC-FDES copy book.

Example:

```
CALL 'TIPFCS' USING FCS-OPEN, BATCH-FILE, BATCH-FDES.
```

Error Conditions:

PIB-DUP-AFT	a file of the name given in the file-pkt is already assigned to the terminal.
PIB-NOT-FOUND	the requested file does not exist.

-+*+-

6.11.7 DYN: WRITE RECORD(S) FCS-PUT

Dynamic file records are a fixed size of 512 bytes. If a RECORD-NUMBER is given (in an FCS-PUT) which is beyond the current end limit of the file, FCS will expand the file to accept that record.

An exception occurs if the new RECORD-NUMBER is beyond the maximum file size.

Expansion may occur until the dynamic file upper boundary size is reached; approximately 48 increments of 40 blocks, each 512 bytes.

Syntax:

```
CALL 'TIPFCS' USING FCS-PUT, file-pkt, record, rec-num
                                [,rec-count]
```

Where:

FCS-PUT function code from the TC-FCS copy book.

file-pkt File name descriptor.

record 512 character record area. (Fullword alligned)

rec-num is the block number to be read.

rec-count is an optional parameter which specifies how many blocks are to be read. (default: one block).

Example:

```
CALL 'TIPFCS' USING FCS-PUT, BATCH-FLE, REC-WRK, REC-NUM.
```

-+*+-

6.11.8 DYN: SCRATCH FILE FCS-SCRATCH

This deletes a Dynamic file from the FCS system. Temporary or Permanent Dynamic files may be scratched.

A file must be assigned to scratch it. Temporary Dynamic files are automatically scratched if TIP/30 terminates abnormally.

Syntax:

```
CALL 'TIPFCS' USING FCS-SCRATCH, file-pkt.
```

Where:

FCS-SCRATCH function code from the TC-FCS copy book.

file-pkt Logical file name packet.

Example:

```
CALL 'TIPFCS' USING FCS-SCRATCH, BATCH-FILE.
```

Error Conditions:

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

-+*+-

6.12 OUTPUT TO PRINT A FILE TIPPRINT

TIPPRINT is a re-entrant subroutine that can be called by a user program to perform printing functions. The print output can be directed to either a central site printer (via the OS/3 spooling system), or a communications printer.

The user interface with TIPPRINT is similar to that used in the TIPFCS calls (ie function-code, file-name, record). With TIPPRINT, however, the fourth parameter is always the name of a user supplied buffer which is used by TIPPRINT as a work area. The file-name (2nd parameter) is used to inform TIPPRINT of the destination of the print file (either batch or communications).

TIPPRINT uses variable length records, which contain a DI (device independent carriage control) code (see OS/3 Basic Data Management publication regarding 'DTFPR CTLCHR=DI').

If the user intends to write to the main-site printer then simply call TIPFCS with a file-name of 'PRNTR', since this is a standard file generated into every TIP system. Using TIPPRINT will cause a 3500 byte module (TI$PRINT) to be included in your program.

Syntax:

```
CALL 'TIPPRINT' USING FCS-OPEN,  FILE-PKT, INFO, BUFFER.
CALL 'TIPPRINT' USING FCS-PUT,   FILE-PKT, LINE, BUFFER.
CALL 'TIPPRINT' USING FCS-FLUSH, FILE-PKT, LINE, BUFFER.
CALL 'TIPPRINT' USING FCS-CLOSE, FILE-PKT, LINE, BUFFER.
```

Where:

Param 1 function code from the TC-FCS copy book.

FCS-OPEN open the file

FCS-PUT write a record to the file

FCS-FLUSH force the buffer to be written

FCS-CLOSE close the file

file-pkt file name packet:

```
05  PRINT-FILE-NAME.
  10  FILE-NAME          PIC X(8).
  10  FILE-STATUS        PIC X.
```

FILE-NAME is the name of the print file that is to be used by TIPPRINT.

If the first three characters of FILE-NAME are 'COP' or 'AUX' then TIPPRINT will route the output data directly to the auxiliary printer. In this case, the fourth character of the file name is used to specify the auxiliary device number (1-9), with the default being device number 1.

The remaining four characters of the file name may be used to specify a destination terminal name. The terminal name specified identifies the terminal that is to be used for the data transfer. The default destination is the terminal being used by the program calling TIPPRINT.

The terminal name may also be '*BYP' to direct the output to the bypass terminal of the associated terminal cluster. (See TIP SYSTEM GENERATION).

FILE-STATUS this will contain an 'O' to indicate a page overflow condition has occurred.

This may contain a 'B', to indicate that the print output has been interrupted. This could occur if the output message could not be delivered to the printer (ie: printer error) or if the terminal operator pressed the 'Message Waiting' key and replied 'NO' to the BREAK prompt.

The BREAK prompt is displayed as follows:

```
Break - Continue? >YES  >NO
```

If any I/O error occurs on the auxiliary device, a message is sent to the error reporting terminal (as specified in the 'open information packet'). This message will identify the error and the name of the terminal associated with the error. The message that is sent to the error reporting

terminal is as follows:

PRINT ERROR AT ____, ERROR = '_____'

If this condition occurs, a status code of "B" is returned to the calling routine to indicate that the printed output has been broken.

Param 3 information packet or record area

If the function code of FCS-OPEN is used, then parameter 3 is used to supply some general information to TIPPRINT.

The format of the information packet is defined by the supplied copy book TC-PRINT:

```
       05 TIPPRINT-INFORMATION-PKT   COPY  TC-PRINT  OF  TIP.
000001*
000002**  COPY ELEMENT FOR TIPPRINT INFORMATION PACKET
000003*
000004          10  PRINT-BUF-LEN              PICTURE 9(4) COMP-4.
000005          10  PRINT-PAG-LEN              PICTURE 9(4) COMP-4.
000006          10  PRINT-ERR-TERM             PICTURE X(4).
000007          10  PRINT-TOP-OF-FORM          PICTURE X.
000008          10  PRINT-LINE-FEED            PICTURE X.
000009          10  PRINT-NOW-PRINTING         PICTURE X.
000010          10  PRINT-UPPER-CASE           PICTURE X.
000011          10  PRINT-RESERVED             PICTURE X(4).
```

Where:

PI-BUFF-LEN is used to specify the length of the buffer that the user program is supplying. The minimum length is 512 bytes.

PI-PAGE-LEN is used to specify the page length. Whenever a number of lines equal to this value has been output, TIPPRINT will return an 'O' (overflow) status to the calling program.

PI-ERROR-TERM is used to specify the name of a terminal that is to receive the error message if an error condition occurs. The value specified may be the name of a valid communications terminal in the network, or the value '*CON' to indicate the central site operator's console, or a value of '*RET' to indicate that no error message is to be sent. In the case of '*RET', TIPPRINT will simply return to the calling program with a status of 'B' and the specific ICAM error code will be located in the third byte of the buffer area.

PI-TOP-OF-FORM is used to select an optional skip to top of form before the first page is output. If the skip is required, this location should be set to 'Y', otherwise a value of 'N' is used. The default value is 'N'.

PI-LINE-FEED is used to select an optional line feed at the end of every communications output message. This is required by some types of communications printers as they do not perform a line feed sequence automatically with every message. If this option is required, this location should be set to 'Y', otherwise a value of 'N' is used. The default value is 'N'.

PI-NOW-PRINTING is used to suppress the 'Now Printing' message which is displayed when the output is being routed to a batch print file. To select this option, this location should be set to 'N'. The default value is 'Y'.

PI-UPPER-CASE is used to select upper case only output. The default is 'Y' for batch print files (ie: PRNTR) and 'N' for communications print files (ie: AUX1).

line print line. If the function code of FCS-PUT is used, then paramter 3 is used to specify the record area (print line). The record used is of variable length and carriage control is performed by the use of a DI character (byte 5 of the record). The format of a record used with TIPPRINT is as follows:

```
05  PRINT-LINE.
    10  LI-LENGTH                    PICTURE 9(4) COMP-4.
    10  FILLER                       PICTURE XX.
    10  LI-DI-CONTROL                PICTURE X.
    10  LI-DATA                      PICTURE X(??).
```

Where:

PRINT-LINE is a variable length record containing length field, a DI code (for carriage control), and the data to be printed.

LI-LENGTH is used to specify the length of the print line. Note that the length specified includes the length of the the record header (5 bytes).

LI-DI-CODE These codes are described in the OS/3 data management manual in the section describing the printer I/O structures.

Standard FORTRAN spacing codes may be used if the output is being sent to an auxiliary device. It is best to use the DI codes as documented in the Data Management User Guide.

```
' '  -  single space
'0'  -  double space
'-'  -  triple space
'1'  -  skip to the top of a new page
```

TIPPRINT recognizes a special DI-code "V" (ignored if the output is a batch printer). This code instructs TIPPRINT to output the contents of the associated print line with NO modifcation or translation.

This allows the user to send arbitrary codes to an auxiliary device.

LI-DATA This is the data to be printed.

Param 4 work area buffer. This parameter is used to identify a buffer that is used by TIPPRINT. This buffer must be at least 512 bytes in size and should be full-word aligned (SYNC). The only information stored in this buffer that is of any interest to the calling program is the ICAM delivery status, which is located in the third byte of the buffer. The format of the buffer would be as follows:

```
05  TIPPRINT-BUFFER.
    10  BU-LEN                     PICTURE 9(4) COMP-4.
    10  BU-ICAM-STATUS              PICTURE X.
    10  FILLER                     PICTURE X(????).
```

Additional Considerations:

All auxiliary device messages are sent to the MEDIUM terminal queue.

ICAM should be generated with FEATURES=(OPCOM,OUTDELV).

There is a COBOL copy book supplied which contains some commonly used carriage control codes. This copy book should be copied into the WORKING-STORAGE SECTION. The format follows:

```
        01  PRINTER-CODES.   COPY TC-DI OF TIP.
000001*
000002* DEFINE CODE TO BE USED FOR PRINTER CARRIAGE CONTROL
000003*
000004      05  TC-DI-1                          PICTURE 9(4)
000005          COMPUTATIONAL-4 VALUE 9985.
000006      05  TC-FILLER1 REDEFINES TC-DI-1.
000007          10  TC-DI-HOME                   PICTURE X.
000008          10  TC-DI-PRINT-SPACE1           PICTURE X.
000009
000010      05  TC-DI-2                          PICTURE 9(4)
000011          COMPUTATIONAL-4 VALUE 515.
000012      05  TC-FILLER2 REDEFINES TC-DI-2.
000013          10  TC-DI-PRINT-SPACE2           PICTURE X.
000014          10  TC-DI-PRINT-SPACE3           PICTURE X.
000015
000016      05  TC-DI-3                          PICTURE 9(4)
000017          COMPUTATIONAL-4 VALUE 1029.
000018      05  TC-FILLER3 REDEFINES TC-DI-3.
```

```
000019          10  TC-DI-PRINT-SPACE4          PICTURE X.
000020          10  TC-DI-PRINT-SPACE5          PICTURE X.
000021
000022      05  TC-DI-4                         PICTURE 9(4)
000023          COMPUTATIONAL-4 VALUE 1543.
000024      05  TC-FILLER4 REDEFINES TC-DI-4.
000025          10  TC-DI-PRINT-SPACE6          PICTURE X.
000026          10  TC-DI-PRINT-SPACE7          PICTURE X.
000027
000028      05  TC-DI-5                         PICTURE 9(4)
000029          COMPUTATIONAL-4 VALUE 2057.
000030      05  TC-FILLER5 REDEFINES TC-DI-5.
000031          10  TC-DI-PRINT-SPACE8          PICTURE X.
000032          10  TC-DI-PRINT-SPACE9          PICTURE X.
000033
000034      05  TC-DI-6                         PICTURE 9(4)
000035          COMPUTATIONAL-4 VALUE 2576.
000036      05  TC-FILLER6 REDEFINES TC-DI-6.
000037          10  TC-DI-PRINT-SPACE10         PICTURE X.
000038          10  TC-DI-PRINT-NO-SPACE        PICTURE X.
```

6.13 FCS COBOL COPY ELEMENT TC-FCS

This copy book must always be placed in the WORKING-STORAGE SECTION.

```
       01  FCS-CODES. COPY TC-FCS OF TIP.
000001*  TC$FCS COPY ELEMENT FOR TIP/30 FILE CONTROL INTERFACE
000002*
000003*****************************************************************
000004*  THE FOLLOWING 05-LEVEL DATA ITEMS ARE USED AS                *
000005*            FCS FUNCTION CODES                                 *
000006*****************************************************************
000007      05  FCS-ACCESS      VALUE 'A'         PICTURE X.
000008      05  FCS-ADD         VALUE '9'         PICTURE X.
000009      05  FCS-ASSIGN      VALUE '>'         PICTURE X.
000010      05  FCS-BACK        VALUE 'B'         PICTURE X.
000011      05  FCS-CLOSE       VALUE 'D'         PICTURE X.
000012      05  FCS-CREATE      VALUE 'N'         PICTURE X.
000013      05  FCS-DELETE      VALUE '<'         PICTURE X.
000014      05  FCS-ESETL       VALUE '6'         PICTURE X.
000015      05  FCS-FLUSH       VALUE 'F'         PICTURE X.
000016      05  FCS-GET         VALUE 'G'         PICTURE X.
000017      05  FCS-GETUP       VALUE '0'         PICTURE X.
000018      05  FCS-HOLD        VALUE 'H'         PICTURE X.
000019      05  FCS-JOURNAL     VALUE 'T'         PICTURE X.
000020      05  FCS-NEXT        VALUE 'X'         PICTURE X.
000021      05  FCS-NOUP        VALUE '2'         PICTURE X.
000022      05  FCS-OPEN        VALUE 'O'         PICTURE X.
000023      05  FCS-PUT         VALUE 'P'         PICTURE X.
000024      05  FCS-RELEASE     VALUE 'R'         PICTURE X.
000025      05  FCS-SCRATCH     VALUE 'Q'         PICTURE X.
000026      05  FCS-SETL        VALUE '5'         PICTURE X.
000027      05  FCS-SETL-EQ     VALUE 'E'         PICTURE X.
000028      05  FCS-SETL-GT     VALUE 'Z'         PICTURE X.
000029      05  FCS-TREN        VALUE '*'         PICTURE X.
000030*
000031*****************************************************************
000032*  THE FOLLOWING 05-LEVEL DATA ITEMS ARE USED AS                *
000033*            FCS DYNAMIC FILE, CLASSES                          *
000034*****************************************************************
000035      05  FCS-CLASS-PERM VALUE 'P'         PICTURE X.
000036      05  FCS-CLASS-TEMP VALUE 'T'         PICTURE X.
000037      05  FCS-CLASS-QED  VALUE 'E'         PICTURE X.
000038*
000039*****************************************************************
000040*  THE FOLLOWING 05-LEVEL DATA ITEMS ARE USED AS                *
000041*            FCS DYNAMIC FILE, TYPES                            *
000042*****************************************************************
000043      05  FCS-TYPE-NEW    VALUE 'C'         PICTURE X.
```

FCS COBOL COPY ELEMENT

```
000044     05  FCS-TYPE-OLD   VALUE 'E'        PICTURE X.
000045*
000046*****************************************************************
000047*  THE FOLLOWING 05-LEVEL DATA ITEMS ARE USED AS                *
000048*            FCS FILE PERMISSIONS                               *
000049*****************************************************************
000050     05  FCS-PERM-READONLY  VALUE 'R'    PICTURE X.
000051     05  FCS-PERM-WRITEONLY VALUE 'W'    PICTURE X.
000052     05  FCS-PERM-UPDATE    VALUE 'U'    PICTURE X.
000053*
000054*****************************************************************
000055*  THE FOLLOWING 05-LEVEL DATA ITEMS ARE USED AS                *
000056*            FCS LOCK OPTIONS                                   *
000057*****************************************************************
000058     05  FCS-LOCK-YES   VALUE 'Y'        PICTURE X.
000059     05  FCS-LOCK-NO    VALUE 'N'        PICTURE X.
```

6.14 COMMON TIPFCS FUNCTIONS AND STATUS CODES

FUNCTION	POSSIBLE STATUS CODE	EXPLANATION
FCS-OPEN	PIB-DUP-AFT	dup in AFT
	PIB-FUNCTION	no DVC VOL LFD DM error on OPEN undefined file AFT exhausted
FCS-CLOSE	PIB-FUNCTION	undefined file
FCS-DELETE	PIB-NOT-HELD PIB-NOT-FOUND PIB-WRONG-MODE	delete request ignored
FCS-NEXT	PIB-WRONG-MODE PIB-EOF	not ISAM file or wrong mode
FCS-GET	PIB-WRONG-MODE	not in sequential mode or no DVC VOL LFD
	PIB-EOF	key out of range
FCS-GETUP	PIB-RECORD	another user has record held or table full. Try again.
FCS-ADD	PIB-DUP-KEY	duplicate key exists
FCS-HOLD	PIB-HELD	another user has key held or table is full. Try again.
FCS-RELEASE	PIB-NOT-HELD	FCS-HOLD or FCS-GETUP not issued.
FCS-NOUP	PIB-NOT-HELD	FCS-HOLD or FCS-GETUP not issued.
FCS-PUT	PIB-WRONG-MODE PIB-NOT-HELD	file not DAM, ISAM wrong mode FCS-GETUP not issued
FCS-SETL	PIB-EOF	key out of range

6.15 ASSEMBLER FCS FUNCTIONS AND STATUS CODES

The macro TP$BEGIN will generate 1 byte function codes as constants (F@xxx) and status code equates (F#xxx).

COBOL	ASSEMBLER FUNCTION
FCS-OPEN	F@OPEN
FCS-CLOSE	F@CLOSE
FCS-DELETE	F@DELETE
FCS-NEXT	F@GETNX
FCS-GET	F@GET
FCS-GETUP	F@GETUP
FCS-ADD	F@ADD
FCS-HOLD	F@HOLD
FCS-RELEASE	F@RLSE
FCS-NOUP	F@NOUP
FCS-PUT	F@PUT
FCS-FLUSH	F@FLUSH
FCS-SETL	F@SETL
FCS-ESETL	F@ESETL
FCS-BACK	F@BACK

COBOL	ASSEMBLER STATUS
STS-GOOD	F#OK
STS-DUP-AFT-NAME	F#DUPF
STS-DUP-KEY	F#DUPK
STS-EOF	F#EOF
STS-ACTIVE	F#ACT
STS-IO-ERROR	F#FATAL
STS-FUNCTION	F#BAD
STS-LOCKED	F#LCK
STS-NOT-FOUND	F#NOF
STS-SECURITY	F#PAS
STS-HELD	F#HELD
STS-NOT-HELD	F#NHLD
STS-WRONG-MODE	F#TYP

6.16 'FCS' FOR LIBRARY FILES FCS: libraries

Operating system (source) libraries may be accessed by user programs.

Opening a library file is similar to opening a data file. The FILE-DESCRIPTOR packet for library files has several additional fields.

The PIB status will be set to PIB-DUP-KEY on an FCS-OPEN with FDES-FCS-PERM set to 'W', if a module of the same name already exists in the library.

The following function codes are used to process library files:

```
FCS-OPEN  -  open file/element
FCS-GET   -  get next input record
FCS-PUT   -  put next output record
FCS-CLOSE -  close file,
             if reading then de-access file
             if writing then the old module is flagged
                        as deleted
FCS-NOUP  -  close file,
             if reading then de-access file
             if writing then the old module is not
                        flagged deleted.
```

LIBRARY FILE DESCRIPTOR

6.16.1 LIBRARY FILE DESCRIPTOR — FCS: libraries

The layout of the extended FILE-DESCRIPTOR packet for library files is as follows:

```
02  LIB-FDES.
    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.
    05  FDES-ELEMENT                    PICTURE X(8).
    05  FDES-COMMENTS                   PICTURE X(30).
    05  FDES-DATE                       PICTURE X(8).
    05  FDES-TIME                       PICTURE X(5).

02  LIB-RECORD                      PICTURE X(128).
```

Where:

FDES-FILE-NAME library file name, as specified in TIP catalogue.

FDES-FCS-TYPE the following values may be used for FDES-FCS-TYPE.

'S' - source module

'M' - macro or proc

'I' - read internal symbol dictionary for load module

'D' - read full directory of file

'F' - read directory of file (without comments)

FDES-FCS-PERM Specified when the element is opened. Set to 'R' if reading; 'W' if writing.

FDES-ELEMENT element (module) name within library.

FDES-COMMENTS	comments from header record
FDES-DATE	date module was created, YY/MM/DD
FDES-TIME	time module was created, HH:MM
LIB-RECORD	element record I/O area. The record area is a fixed length of 128 bytes which should be cleared to spaces.

-+*+-

6.16.2 LIB: CLOSE LIBRARY FCS-CLOSE

Remove the file from use by programs at the associated terminal. The entry is removed from the Active File Table. Library files are always physically closed by TIP/30. They are not allowed to remain open.

Syntax:

CALL 'TIPFCS' USING FCS-CLOSE, file-pkt.

Where:

FCS-CLOSE function code from the TC-FCS copy book.

file-pkt Logical file name packet.

Example:

CALL 'TIPFCS' USING FCS-CLOSE, PAYFILE.

Error Conditions:

PIB-FUNCTION file is not assigned to the program.

-+*+-

6.16.3 LIB: READ RECORD FCS-GET

Read the next record from the opened element.

Syntax:

CALL 'TIPFCS' USING FCS-GET, file-pkt, record.

Where:

FCS-GET	function code from the TC-FCS copy book.
file-pkt	Logical file name packet.
record	128 character record area.

Example:

```
CALL 'TIPFCS' USING FCS-GET, SRC-FILE, SRC-REC.
```

Error Conditions:

PIB-FUNCTION	the file is not assigned to the program.
PIB-IO-ERROR	some I/O error occurred on the disk.
PIB-EOF	end of file was reached during sequential processing
PIB-WRONG	the file was not defined for input processing.

-+*+-

6.16.4 LIB: CLOSE LIBRARY; ABORT OUTPUT FCS-NOUP

Remove the file from use by programs at the associated terminal. The entry is removed from the Active File Table. Library files are always physically closed by TIP/30.

This call will result in the current output element not being activated in the library directory.

Syntax:

```
CALL 'TIPFCS' USING FCS-NOUP, file-pkt.
```

Where:

FCS-NOUP function code from the TC-FCS copy book.

file-pkt Logical file name packet.

Example:

```
CALL 'TIPFCS' USING FCS-NOUP, SRCFILE.
```

Error Conditions:

PIB-FUNCTION file is not assigned to the program.

-+*+-

6.16.5 LIB: OPEN LIBRARY FCS-OPEN

Make the specified file available for processing by programs at the calling terminal. An entry is made in the Active File Table for the file.

TIPFCS will issue a Data Management OPEN for the library and . make the specified element available.

Syntax:

CALL 'TIPFCS' USING FCS-OPEN, file-pkt ,file-desc.

Where:

FCS-GETUP	function code from the TC-FCS copy book.
file-pkt	Logical file name packet.
file-desc	file descriptor packet (extended).

Example:

```
CALL 'TIPFCS' USING FCS-OPEN, MST-FILE.
```

Error Conditions:

PIB-IO-ERROR	some I/O error occurred while opening the file.
PIB-DUP-AFT-NAME	a file of with the same name given in file-pkt is already assigned to the terminal.
PIB-DUP-KEY	an element of that name already exists in the library. The program may wish to ignore this error.
PIB-LOCKED	the file is currently locked either by some other TIP/30 application program or by some batch job.

-+*+-

6.16.6 LIB: WRITE RECORD FCS-PUT

The record is appended to the output element.

Syntax:

CALL 'TIPFCS' USING FCS-PUT, file-pkt, record.

Where:

FCS-PUT	function code from the TC-FCS copy book.
file-pkt	Logical file name packet.
record	128 character record area.

Example:

CALL 'TIPFCS' USING FCS-PUT, SRC-FILE, SRC-REC.

Error Conditions:

PIB-FUNCTION	the file is not assigned to the program.
PIB-IO-ERROR	some I/O error occurred on the disk.
PIB-EOF	the file is full.
PIB-WRONG	the file was not opened for output processing.

-+*+-

6.17 'TIPFCS' FOR EDIT BUFFERS FCS: edit

The text editor supplied with TIP/30 does all of the actual editing in an FCS dynamic work file. When all updates are finished the user may request that the module be written to a library element.

The file structure used by the editor is maintained by FCS. An edit file consists of a control block (block 1), several index blocks, and data blocks.

Each data block holds up to six records. A record is 85 characters. The first 80 characters are the data from the library. The 81st character is a version number. Characters 82 to 85 are unused.

Records in the file may be accessed by a relative line number. If a record is deleted all following records move up. If a record is added all following records move down.

An edit file may be created by setting FDES-FCS-CLASS to FCS-CLASS-QED and FDES-FCS-TYPE to FCS-TYPE-NEW.

6.17.1 EDIT: ADD FCS-ADD

This function is used to add/insert a new record to an edit buffer.

Syntax:

```
 CALL 'TIPFCS' USING FCS-ADD, file-pkt, RECORD, LINE-NUM.
02  RECORD          PIC X(85).
02  LINE-NUM        PIC 9(7) COMP-4 SYNC.
```

Where:

FCS-ADD function code from the TC-FCS copy book.

file-pkt Logical file name packet.

record record area.

line-num binary fullword holding the relative record number.

Additional Considerations:

The record is written to the file at the specified position. Any records at that position or higher are shifted to the next higher position. The records are not actually moved, but the index is changed to reflect their new logical position in the file.

-+*+-

6.17.2 EDIT: CLOSE FCS-CLOSE

Syntax:

CALL 'TIPFCS' USING FCS-CLOSE, file-pkt.

Where:

FCS-CLOSE function code from the TC-FCS copy book.

file-pkt Logical file name packet.

Additional Considerations:

The file is closed. You should use FCS-FLUSH to flush out any updated blocks from the I/O buffers beforehand.

-+*+-

6.17.3 EDIT: DELETE **FCS-DELETE**

Syntax:

```
CALL 'TIPFCS' USING FCS-DELETE, file-pkt, RECORD, LINE-NUM.

02  RECORD          PIC X(85).
02  LINE-NUM        PIC 9(7) COMP-4 SYNC.
```

Where:

FCS-DELETE function code from the TC-FCS copy book.

file-pkt Logical file name packet.

record record area. Not actually used but must be provided as a place holder.

line-num binary fullword holding the relative record number which is to be deleted.

Additional Considerations:

The record at the specified position is deleted from the file. Any records at that position or higher are shifted to the next lower position. The records are not actually moved, but the index is changed to reflect their new logical position in the file.

-+*+-

6.17.4 EDIT: FLUSH FCS-FLUSH

Syntax:

CALL 'TIPFCS' USING FCS-FLUSH, file-pkt.

Where:

FCS-FLUSH function code from the TC-FCS copy book.

file-pkt Logical file name packet.

Additional Considerations:

Updated blocks are not written to disk unless FCS determines that they need to be written to make space in the buffer.

To force out all updated blocks use this function code. This should be done before closing the file.

-+*+-

6.17.5 EDIT: GET **FCS-GET**

Syntax:

```
CALL 'TIPFCS' USING FCS-GET, file-pkt, record, line-num.

02  RECORD          PIC X(85).
02  LINE-NUM        PIC 9(7) COMP-4 SYNC.
```

Where:

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

file-pkt Logical file name packet.

record record area

line-num binary fullword holding the relative record number.

Error Conditions:

PIB-EOF the record number is out of bounds.

-+*+-

6.17.6 EDIT: OPEN FCS-OPEN

Syntax:

```
CALL 'TIPFCS' USING FCS-OPEN, file-pkt, file-desc,
                                    IO-BUFFER, [NUM-BUFS].
```

Where:

FCS-OPEN function code from the TC-FCS copy book.

file-pkt Logical file name packet.

file-desc file description packet.

IO-BUFFER An I/O buffer to be used with the file. This buffer must be at least 1536 bytes.

This buffer must be fullword aligned.

If the file is being processed randomly more buffers will improve throughput. For sequential access 1536 is good enough. If this parameter is omitted FCS will allocate 1536 bytes from free memory.

NUM-BUFS is a halfword (PIC 9 COMP-4 SYNC) which indicates the number of I/O buffers in IO-BUFFER. The first 512 bytes of IO-BUFFER holds the control block. NUM-BUFS is the number of I/O buffers following the control block. Minimum is 2. Maximum is 12.

-+*+-

6.17.7 EDIT: PUT FCS-PUT

This function is used to replace a record in the edit buffer.

Syntax:

```
CALL 'TIPFCS' USING FCS-PUT, file-pkt, RECORD, LINE-NUM.

02  RECORD          PIC X(85).
02  LINE-NUM        PIC 9(7) COMP-4 SYNC.
```

Where:

FCS-PUT	function code from the TC-FCS copy book.
file-pkt	Logical file name packet.
record	record area.
line-num	binary fullword holding the relative record number

Error Conditions:

PIB-EOF	the line number is out of bounds.

-+*+-

6.17.8 EDIT: SCRATCH FCS-SCRATCH

Syntax:

CALL 'TIPFCS' USING FCS-SCRATCH, file-pkt.

Where:

FCS-SCRATCH function code from the TC-FCS copy book.

file-pkt Logical file name packet.

Additional Considerations:

The file will be scratched. All edit files are created as permanent and must be scratched to get rid of them.

-+*+-

6.18 TOTAL DATA BASE FCS: total

TIP/30 provides an interface to TOTAL, a Data Base developed and marketed by CINCOM Systems. The user should refer to documentation supplied by CINCOM for the programming conventions required to use TOTAL.

To use TOTAL online with TIP/30 the user must specify the DBMS=(TTL,n) parameter in the TIP/30 System Generation, where n is the amount of memory required to load TOTAL/7 and your DBMOD.

The calling sequence to use TOTAL from a batch program is:

```
CALL DATBAS USING  param-1, param-2, etc.
```

The calling sequence to use TOTAL online with TIP/30 is:

```
CALL TOTAL USING param-1, param-2, etc.
```

The same parameters as provided in a batch program are used in the online CALL.

The online programs have two additional function codes to TOTAL which are handled by the Allinson-Ross interface module.

```
CALL TOTAL USING FREEF,status,file,end
```

All records held by this user for the specified file will be released.

```
CALL TOTAL USING FREEX,status,file,end
```

All records held for all files by the user will be released.

If an online program must add a variable record to a master record when there are none in the chain, then it must use the ADDVC function. ADDVA and ADDVB both require that the program have previously read some other record in the chain in order to hold it.

```
CALL TOTAL USING ADDVC,status,file,.etc..
```

6.19 DATA BASE MANAGEMENT INTERFACE FCS: dbms

This is the interface between the online application program and the data base system generated into TIP/30 The user has the choice of TOTAL/7, DMS/90, DBS/90, or his own home grown.

But only one may be chosen.

CALL	DATA BASE
TOTAL	TOTAL/7
XR7DMS	DMS/90
IXF???	DBS/90
TIPDBMS	Home grown

6.19.1 DMS/90 - XR7DMS **FCS: dms/90**

Online programs must be processed by the DMS/90 DML pre-processor just as batch programs are. Online programs follow the sequence of IMPART, BIND, I/O, UNBIND, ... BIND, I/O , UNBIND, DEPART.

An online program must always UNBIND before transfering control to any other program or ending. (ie. UNBIND before TIPRTN, TIPXCTL, TIPSUB, etc..).

-+*+-

6.19.2 DBS/90 - IXF??? FCS: dbs/90

Most users access DBS/90 through the UNIS/90 I/O interface, although the direct interface is available.

IMS/90 programs may be catalogued as re-usable but not re-entrant, as the standard IXOIO module is not re-entrant. Also access to DBS/90 will not work properly if the programs are re-useable and keep the data base open across internal succession. This is a problem because the activation record may move and DBS/90 will not allow it to move] IMS/90 programs accessing DBS/90 must either be cataloged as non re-entrant and non-reusable or they must not keep the data base open during internal succession (ie. only do delayed internal]).

Native TIP/30 COBOL programs accessing DBS/90 may be catalogued as either re-entrant or not re-entrant but not as re-useable.

For native TIP/30 programs the user must link-edit with the modules TI$IXOIO and TI$DBSIF. TI$IXOIO contains entry points for IXFOPN and IXOIO. TI$DBSIF contains entry points for OPEN, CLOSE, RDKEY, etc...

-+*+-

6.20 JOURNAL FILE PROCESSING FCS: journal

FCS creates journal file records for each update of a user online file as selected in the TIP/30 generation. User programs may also write records to the journal file. In this way user statistics may be collected for billing, to audit performance and throughput etc.

Syntax:

```
CALL 'TIPFCS' USING FCS-JOURNAL, file-pkt, JRN-RECORD.
```

Where:

file-pkt Logical file name packet. The name used here is ignored.

JRN-RECORD journal record, halfword aligned in the following format.

```
05  JRN-RECORD.
      10  RECORD-LENGTH          PIC 9(4) COMP-4 VALUE ?.
      10  FILLER                 PIC X(2).
      10  TYPE                   PIC X(4).
      10  USER-ID                PIC X(8).
      10  PROG-ID                PIC X(8).
      10  FILE-PKT               PIC X(8).
      10  DATE                   PIC 9(6) COMP-3.
      10  TIME                   PIC 9(6) COMP-3.
      10  TERM-ID                PIC X(4).
      10  KEY                    PIC X(4).
      10  FILLER                 PIC X(8).
      10  USER-DATA.
```

The journal file holds variable length records. The first two bytes of the record (ie. RECORD-LENGTH) contains the length of the journal record to be written.

Where:

TYPE	BEFR - BEFORE IMAGE OF RECORD
	AFTR - AFTER IMAGE OF RECORD
	NEW - NEW RECORD ADDED TO FILE
	CKPT - FILE WAS CLOSED AT THIS TIME
	USER - USER SUPPLIED JOURNAL RECORD
	LGON - LOGON RECORD
	LGOF - LOGOFF RECORD
	TREN - TRANSACTION END (CHECKPOINT)
	PRST - PROGRAM START
	PREN - PROGRAM END
	STAT - STATUS
USER-ID	user who created the journal record
PROG-ID	name of program in use by USER-ID.
file-pkt	file pkt for the file.
DATE	date created.
TIME	time created.
TERM-ID	terminal name used by USER-ID.
KEY	block number for a DAM file
FILLER	not used

6.20.1 'LGOF' JOURNAL RECORD FORMAT — FCS: journal

```
02  USER-DATA.
    05  FILLER              X(2).
    05  TIME-LGON-HR        9(2) COMP-3.
    05  TIME-LGON-MM        9(2) COMP-3.
    05  TIME-LGON-SS        9(2) COMP-3.
    05  TIME-WALL-MSEC      9(7) COMP-4.
    05  TIME-CPU-MSEC       9(7) COMP-4.
    05  TIME-MSGIN          9(4) COMP-4.
    05  TIME-MSGOUT         9(4) COMP-4.
    05  TIME-DATE-ON        9(6) COMP-3.
    05  TIME-TIME-ON        9(6) COMP-3.
    05  TIME-DATE-OFF       9(6) COMP-3.
    05  TIME-TIME-OFF       9(6) COMP-3.
    05  TIME-RESPONSE       9(7) COMP-4.
```

-+*+-

6.20.2 BATCH JOURNAL FILE READ FCS: journal

There is an object module called TI$JRN which may be used in a batch program to read the TIP$JRN file. It will fetch module TB$JDK and/or TB$JMT which are also in the TIP/30 release library. The programs must be in the same library you execute your batch program from.

Three entry points are available:

CALL TI$JRNOP

- will open the file TIP$JRN.

CALL TI$JRNCL

- will close the file TIP$JRN.

CALL TI$JRNGT USING RECORD.

- will return a variable length journal file record in the format described above. End of file is signaled by returning a zero length record.

-+*+-

7. CHAPTER VII - MESSAGE CONTROL SYSTEM MCS

This chapter of the TIP/30 reference manual documents the TIP/30 facilities that are available to handle input and output from terminals.

There are three levels of interface provided:

- Message Control System (MCS)
- Line-oriented I/O
- Direct Communications I/O (DCIO)

The MCS interface is a high level interface; that is, application programmers may develop screen formats (templates) and use them in the on-line program. By using this interface, the programmer achieves a high degree of hardware independence.

The Line-oriented I/O interface consists of a number of supplied subroutines that facilitate the interactive use of the terminal on a line by line basis. The user can issue prompts and retrieve replies in a simple fashion.

The DCIO interface is provided as a means to achieve more direct control over the activity of the terminal. It is a low level interface that requires the application programmer to supply the control codes to be sent to the terminal.

This interface is intended primarily for assembly language applications and should only be used when the facilities of the high level interface (MCS) cannot achieve the desired results.

7.1 MESSAGE CONTROL SYSTEM MCS

The Message Control System provides the user with the capability of creating and testing displays to be used in online systems. The displays are not defined in the programs which use them. The user program sends and receives only relevant data to and from the CRT. The Message Control System handles all Communications Codes (DICE) and display heading information.

There are four major sections to MCS. Three are utility programs as follows:

MSGDEF - for MCS display definition
MSGTST - for MCS display testing
MSGAR - the MCS Librarian

The fourth is MSGFMT, the Message Formatter, which is an internal part of TIP/30 and interfaces the displays defined by MSGDEF and the user data to be displayed. MSGFMT is the TIP/30 display handler. It merges user data given in the MCS interface packet with a display and sends it to a terminal. On input, MSGFMT removes the user data from a display and stores it in the MCS interface packet. The user data area of the MCS interface packet has a layout similar to a fixed-length data record. There are no fields for tab stops or cursor co-ordinates. These are defined in the message format by MSGDEF and handled completely by MSGFMT at user program execution time.

The Message Formatter optimizes all output messages. For example, when it is more efficient, a series of blanks is replaced by a cursor positioning sequence. The user should therefore select the 'erase before display' option when defining displays using MSGDEF.

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

Once a display is defined it may be called by any user program which supplies its eight-character name. Furthermore, the user may change heading information in display formats without changing the programs which use them. User programs need only process the data since all communications control characters and heading information is handled by the Message Formatter in TIP.

These features greatly reduce programming effort and development time required to get online programs up and running.

7.2 MCS SPECIAL TERMINAL NAMES *MST/*BYP

Normally messages are sent back to the terminal which originated the transaction. When TIP/30 is generated, the user may define terminal clusters. This capability is quite useful since most newer terminal systems are designed as clusters. If the program sends a message to the terminal *MST, TIP/30 will direct it to the master terminal of the cluster to which the originating terminal belongs.

Similarly *BYP directs the message to the terminal defined as BYPASS, If there was no cluster definition or no bypass/master terminal given, the message is sent to the originating terminal.

The *BYP name may also be used by TIPFORK to start up a program on the 'bypass' terminal. This technique is used to queue print requests.

7.3 DOWN LINE LOADED DISPLAY MANAGEMENT MCS: dll

Another advantage of the clustering concept is that the controlling or master terminal is programmable. The user may use this feature to improve terminal I/O throughput. DLL is a program used to down line load display formats into terminal clusters.

When a set of displays are down line loaded MCS keeps track of their names. There is a table of display names maintained for each defined cluster. Whenever a user program requests an MCS display to be sent to a terminal the display table is checked to determine if the display has been down line loaded. If so then TIP will automatically use it.

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

Currently only UTS-400's which have some user programmable memory may use this facility.

7.4 MCS INTERFACE PACKET TC-MCS

The COBOL copy element TC-MCS defines the MCS interface packet (see example following). If several different displays are used in one program the user may re-define the MCS-DATA part of the packet to provide for the different screen layouts.

MCS-NAME, MCS-COUNT, MCS-FUNCTION etc. may be changed during program execution. Thus it is not normally necessary to have more than one MCS Interface Packet in each program.

```
       01  MCS-AREA.  COPY  TC-MCS OF TIP.
000001*
000002*         TIP/30  -  MESSAGE  CONTROL  SYSTEM  PACKET
000003*
000004      02  MCS-NAME                          PICTURE X(8).
000005      02  MCS-TERM                          PICTURE X(4).
000006      02  MCS-FUNCTION                      PICTURE X.
000007*
000008*        'A' - READ FULL SCREEN ON TIPMSGI
000009*        'D' - SEND DATA ONLY (NO HEADINGS DATA)
000010*            - LOW VALUE FIELDS ARE NOT SENT
000011*        'M' - SEND MESSAGE AS UNSOLICTED
000012*        'P' - CAUSE TERMINAL TO PRINT NON-TRANSPARENT
000013*        'R' - CAUSE TIPMSGE TO REFRESH ALL FCC'S
000014*        'S' - STOP SENDING HEADING DATA
000015*              WHEN MCS-COUNT REACHES ZERO
000016*
000017      02  MCS-HOLD                          PICTURE X.
000018*
000019*         'H' - TIPMSGI NOT TO RELEASE RECORD LOCK(S)
000020*
000021      02  MCS-SIZE                          PICTURE S9(4) COMP-4 SYNC.
000022      02  MCS-STATUS                        PICTURE X.
000023          88  MCS-GOOD                        VALUE ' '.
000024          88  MCS-XMIT                        VALUE ' '.
000025          88  MCS-MSG-WAIT                    VALUE '0'.
000026          88  MCS-FKEY1                       VALUE '1'.
000027          88  MCS-FKEY2                       VALUE '2'.
000028          88  MCS-FKEY3                       VALUE '3'.
000029          88  MCS-FKEY4                       VALUE '4'.
000030          88  MCS-FKEY5                       VALUE '5'.
000031          88  MCS-FKEY6                       VALUE '6'.
000032          88  MCS-FKEY7                       VALUE '7'.
000033          88  MCS-FKEY8                       VALUE '8'.
000034          88  MCS-FKEY9                       VALUE '9'.
000035          88  MCS-FKEY10                      VALUE 'A'.
000036          88  MCS-FKEY11                      VALUE 'B'.
000037          88  MCS-FKEY12                      VALUE 'C'.
```

```
000038          88  MCS-FKEY13                  VALUE 'D'.
000039          88  MCS-FKEY14                  VALUE 'E'.
000040          88  MCS-FKEY15                  VALUE 'F'.
000041          88  MCS-FKEY16                  VALUE 'G'.
000042          88  MCS-FKEY17                  VALUE 'H'.
000043          88  MCS-FKEY18                  VALUE 'I'.
000044          88  MCS-FKEY19                  VALUE 'J'.
000045          88  MCS-FKEY20                  VALUE 'K'.
000046          88  MCS-FKEY21                  VALUE 'L'.
000047          88  MCS-FKEY22                  VALUE 'M'.
000048          88  MCS-F-REBUILD               VALUE '1' '5'.
000049          88  MCS-F-NEXT                  VALUE '2' '6'.
000050          88  MCS-F-UPDATE                VALUE '4' '8'.
000051      02  MCS-FILLER                     PICTURE X.
000052*
000053*         VALID FILLER VALUES ARE ' ', '_', OR '*'
000054*
000055      02  MCS-COUNT                      PICTURE S9(4) COMP-4 SYNC.
000056/
000057      02  MCS-DATA.
000058*
000059*     USER SUPPLIED RECORD LAYOUT FOR MCS SCREEN FOLLOWS HERE
000060*
```

Where:

MCS-NAME is the name of an MCS display created using the message definition program (MSGDEF).

MCS-TERM alternate destination terminal. If blank the message goes to the current terminal.

MCS-FUNCTION normally this field is set to blank but the following codes cause special functions to be performed.

'P' on TIPMSGO will cause the cursor to move to the bottom right hand corner of the screen and print the screen non-transparent to auxiliary device 1 of the terminal.

'R' on TIPMSGE will cause all FCC's to be sent to the terminal. This should reset any left from a previous call to TIPMSGE.

'D' on TIPMSGO will result in only the data being

sent to terminal. It is assumed that the heading information is already displayed on the terminal from some previous operation. Data fields which have been set to LOW-VALUES will not be transmitted.

A data only transmission will also clear the error fields to spaces. When the MCS-COUNT of data is processed the remaining data fields will be set to the filler character.

'M' on TIPMSGO will send the message as unsolicited. The sona alert will sound and the message displayed when the terminal operator presses MSG-WAIT key.

'A' on TIPMSGI will guarantee that the entire display is received. If the count of the input received is less than the maximum size of the format then a small message is sent to pull the entire display in and re-format the input message.

'A' NOTE: This doubles the traffic on the line and should be used with discretion.

'S' forces a short transmission. When all data has been processed (MCS-COUNT), no more heading information will be sent to the terminal.

MCS-HOLD setting this field to an 'H' will prevent records held by TIPFCS GETUP from being released during TIPMSGI processing.

MCS-SIZE maximum number of user data characters in the packet. (IE. the size of the MCS-DATA area.) MSGDEF displays this value as the last step of the definition. Also MSGAR will give this value when the displays are printed. This value is not used during output and is set by MCS on input. By comparing this field with the MCS-COUNT field, the user can determine if a complete input was received.

MCS-STATUS This field will be set after an input message. The contents indicate what type of input was received. Observe the supplied 88-level items in the illustrated copy book.

After output this field will be set to 'M' if an unsolicited input message is waiting. The user

program must read this message via TIPMSGI or a Terminal Management Subroutine: PARAM, BREAK, ROLL, etc.

If printing had been requested with TIPMSGO this field will contain the delivery status. See the section on 'Auxiliary device I/O' for a list of the possible status codes.

MCS-FILLER this field may be set to a blank, an underscore ('_') or an asterisk ('*').

Leading spaces in numeric fields will be set to the filler character during TIPMSGO. Trailing spaces in alpha-numeric fields will be set to the filler character during TIPMSGO.

No filler characters will be used in protected data fields.

All filler characters received from the terminal during TIPMSGI will be removed.

MCS-COUNT the number of data characters to be output. If less than the display total the Message Formatter will blank fill (or MCS-FILLER) the remainder of the user fields on the display. On input, the number of data characters received is stored in MCS-COUNT. The input count will always be less than or equal to the value stored in the MCS-SIZE field and will always include the size of the last field entered (ie. if only the first character of a ten character field was entered as the last field of the display, then the count would indicate that the entire field was entered). MCS-COUNT is set to zero after a CALL TIPMSGE.

MCS-DATA Holds the users data for/from the message. The order in which the data fields appear in memory MUST correspond to the order in which they appear in the display. The fields in this area must consist of display type fields (ie: COMP specification not allowed).

Note that terminals transmit unprotected data from the CRT only when the transmit key is pressed. The user is advised to establish that MCS-XMIT was received in the MCS-STATUS field before examining any data fields. An excellent rule of thumb is to check first for MSG-WAIT or other function keys

(even if the program is designed to ignore them).

7.5 READ A MESSAGE FROM A TERMINAL TIPMSGI

Receive and process an MCS display from a terminal. This call is normally used at points in the user program where input is expected from the terminal (ie. after a CALL TIPMSGO or TIPMSGE).

The program should normally specify the same display name in MCS-NAME for both output and input functions.

IF the input message was caused by the user pressing the transmit key at the terminal (ie: NOT as a result of MSG-WAIT or some other function key), data fields in the input message will be extracted from the input message by TIP/30 and placed in the appropriate fields in MCS-DATA area.

Syntax:

```
CALL 'TIPMSGI' USING mcs-pkt.
```

Where:

mcs-pkt message control system packet.

Error Conditions:
None.

7.6 SEND AN ERROR MESSAGE TIPMSGE

After a TIPMSGI, the user program normally validates the data received. To highlight fields in error, move HIGH-VALUES to them and CALL TIPMSGE using the same MCS packet which was used for the TIPMSGI.

Ensure that the field MCS-COUNT contains the input data count from the CALL TIPMSGI, since TIPMSGE uses that value to determine how far to scan through MCS-DATA for the HIGH-VALUE fields.

On UTS-400 style terminals the fields in error will blink. On other terminals a start-of-blink character is sent to the position immediately in front of the data field.

An optional error message may be sent to the fields that were defined as error fields (using the "E" field descriptor code). Note that MCS-COUNT will be set to zero after a CALL TIPMSGE.

Syntax:

```
CALL 'TIPMSGE' USING mcs-pkt [,error-msg]
```

Where:

mcs-pkt message control system packet.

error-msg a field holding the text to be sent to the fields defined as error field(s) on the display. If not specified, the error fields are set to spaces.

The length of this field should be equal to the sum of the sizes of all error fields in the display.

Example:

```
05 ERROR-DATA                    PIC X(30).
     .
     .
     .
MOVE 'INVALID ACCNT NUMB'   TO ERROR-DATA.
CALL 'TIPMSGE' USING MCS-AREA, ERROR-DATA.
```

Error Conditions:
None.

7.7 OUTPUT A MESSAGE TO A TERMINAL TIPMSGO

Send an MCS display to a terminal. The number of data characters specified by MCS-COUNT is taken from MCS-DATA, merged with the display named in MCS-NAME and sent to MCS-TERM.

If MCS-TERM is SPACES, the message is delivered to the originating terminal.

Syntax:

```
CALL 'TIPMSGO' USING mcs-pkt [,fcc-mods]
```

Where:

mcs-pkt message control system packet.

fcc-mods Optional table of two byte entries for modifying the FCC (field control character) characteristics of each data field.

Each entry consists of two characters which will be used as the 'M' and 'N' characters to use in the construction of the FCC sequence [as described in UTS-400 Programmer Reference (UP-8359): FCC Sequence From Host Processor].

This facility is applicable to FCC style terminals such as UTS-400, UTS-20, UTS-40, and plug-compatibile UTS-400 terminals (ie: Q310).

If either character is an asterisk ('*') then the cursor will rest over the corresponding data field when the message is sent to the terminal.

A copy book named TC-FCC is supplied:

```
000001*
000002* TIP/30  -  FCC MODIFICATION EQUATES
000003*            (FOR USE ON OPTIONAL PARAMETER TWO, OF
000004*             'TIPMSGO' CALL.)
000005
000006***** FOLLOWING VALUES ARE USED FOR THE FCC 'M' FIELD
000007      05  FCC-M-TAB-NRM-CHG                PICTURE X VALUE '0'.
000008      05  FCC-M-TAB-OFF-CHG                PICTURE X VALUE '1'.
000009      05  FCC-M-TAB-LOW-CHG                PICTURE X VALUE '2'.
000010      05  FCC-M-TAB-BLK-CHG                PICTURE X VALUE '3'.
000011      05  FCC-M-TAB-NRM                    PICTURE X VALUE '4'.
000012      05  FCC-M-TAB-OFF                    PICTURE X VALUE '5'.
000013      05  FCC-M-TAB-LOW                    PICTURE X VALUE '6'.
000014      05  FCC-M-TAB-BLK                    PICTURE X VALUE '7'.
000015      05  FCC-M-NRM-CHG                    PICTURE X VALUE '8'.
000016      05  FCC-M-OFF-CHG                    PICTURE X VALUE '9'.
000017      05  FCC-M-LOW-CHG                    PICTURE X VALUE ':'.
000018      05  FCC-M-BLK-CHG                    PICTURE X VALUE ';'.
000019      05  FCC-M-NRM                        PICTURE X VALUE '<'.
000020      05  FCC-M-OFF                        PICTURE X VALUE '='.
000021      05  FCC-M-LOW                        PICTURE X VALUE '>'.
000022      05  FCC-M-BLK                        PICTURE X VALUE '?'.
000023*
000024
000025***** FOLLOWING VALUES ARE USED FOR THE FCC 'N' FIELD
000026      05  FCC-N-ANY                        PICTURE X VALUE '0'.
000027      05  FCC-N-ALPHA                      PICTURE X VALUE '1'.
000028      05  FCC-N-NUMERIC                    PICTURE X VALUE '2'.
000029      05  FCC-N-PROTECT                    PICTURE X VALUE '3'.
000030      05  FCC-N-ANY-RIGHT                  PICTURE X VALUE '4'.
000031      05  FCC-N-ALPHA-RIGHT                PICTURE X VALUE '5'.
000032      05  FCC-N-NUMERIC-RIGHT              PICTURE X VALUE '6'.
```

Error Conditions:

None.

7.8 CURSOR TO LAST POSITION & TRANSMIT TIPMSGRV

On most CRT type terminals, the data between HOME or the last SOE and the CURSOR is transmitted to the host when the TRANSMIT or ENTER key is pressed.

The operator may (by mistake) send only a partial screen instead of the whole screen, meaning that some data may be lost. The application program can ensure that the entire screen is sent by using the TIPMSGRV function.

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 total of the sizes of all data fields in the screen format). If MCS-COUNT is less than MCS-SIZE, then the operator did not have the cursor past the last data field when XMIT was pressed.

The program can choose to ignore this operator error by calling TIPMSGRV. The TIPMSGRV subroutine will position the cursor at the bottom right corner of the CRT and cause an auto-transmit to occur. After the call to TIPMSGRV all data fields from the screen will be in the data area of the MCS packet.

Syntax:

```
CALL 'TIPMSGRV' USING mcs-pkt.
```

Where:

mcs-pkt message control system packet.

Error Conditions:

None.

Additional Considerations:

Use of this function will double the traffic on a communications line and should be used with discretion. It may be preferable to send the terminal operator an error message (eg: Cursor Incorrectly Placed!) and instruct the operator to reposition the cursor and re-transmit.

7.9 LINE - ORIENTED TERMINAL I/O LINE I/O

The following subroutines provide terminal I/O handling capabilities that may be used to interact with the user on a line by line basis.

The examples are shown using COBOL-74 syntax.

A native mode program may use these subroutines to facilitate direct control of terminal input and output in situations that require low volume interaction with the user.

7.9.1 CHECK FOR OPERATOR BREAK BREAK

Subroutine BREAK will check for unsolicited input from the terminal. If there has been input, BREAK will prompt the user for a reply. The reply will be parameterized into PARAM-AREA. If no input is available, control returns to the user program.

BREAK is a convenient way to enable the terminal operator to control pauses in continuous displays.

Syntax:

```
CALL 'BREAK' USING PARAM-AREA.
```

Where:

PARAM-AREA Area (64 bytes) to receive the reply to the continuation query if there was an unsolicited interrupt.

-+*+-

7.9.2 PARAMETERIZE AN INPUT MESSAGE PARAM

Input to this subroutine is either from a terminal or a data area. PARAM breaks the input into as many as eight fields (each 8 bytes) using the characters "," "/" and space(s) as field delimiters. If the optional second parameter is given, the input is from the named data area; otherwise, input will be solicited from the terminal.

Alphanumeric sub-fields are left-justified and space filled, while numeric sub-fields are right-justified and zero filled. The MSG-WAIT key is received as the character string F#00. Function keys one through 22 are received as the character string "F#01" through "F#22" in the first 4 bytes of PARAM-AREA.

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 parameterization routine. If supplied, defines a 64 byte field to be parameterized. If omitted, 64 characters of input will be 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).
```

-+*+-

7.9.3 PROMPT THE USER FOR A REPLY PROMPT

PROMPT will display the program name, stack level and a SOE character on the bottom line of the terminal and then solicit input. The terminal operator response will be parameterized into PARAM-AREA.

Syntax:

```
CALL 'PROMPT' USING PARAM-AREA.
```

Where:

PARAM-AREA The 64 byte area where the parameterized terminal input will be placed.

-+*+-

7.9.4 PROMPT THE USER FOR TEXT PROMPTX

PROMPTX will display the program name, stack level, and a SOE character on the last line of the terminal. The first 64 bytes of the next input message will be stored in TEXT-AREA without parameterization.

Syntax:

```
CALL 'PROMPTX' USING TEXT-AREA.
```

Where:

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

-+*+-

7.9.5 PROMPT THE USER FOR TEXT PROMPTX8

PROMPTX8 will display the program name, stack level, and a SOE character on the bottom line of the terminal. The first 80 bytes of the next input message will be stored (without parameterization) in TEXT-AREA.

Syntax:

```
CALL 'PROMPTX8' USING TEXT-AREA.
```

Where:

TEXT-AREA The 80 byte area where the input data will be placed.

-+*+-

7.9.6 SEND ONE LINE AND ROLL SCREEN ROLL

ROLL will scroll the screen up one line and send one 80 byte line from TEXT-AREA to the bottom line of the terminal. If the optional second parameter is specified, ROLL will automatically call the subroutine "BREAK" (see description earlier) after each line is sent.

Syntax:

```
CALL 'ROLL' USING LINE [,PARAM-AREA].
```

Where:

LINE An 80 byte text area to be rolled on the terminal.

PARAM-AREA Optional field used to return result from call to "BREAK" subroutine.

Example:

```
05 LINE                                PIC X(80)
   VALUE "Please enter option:".
```

-+*+-

7.9.7 SET TERMINAL ROLL POINT ROLLPT

The subroutines ROLL, PROMPT, PROMPTX and BREAK all scroll the terminal display from bottom to top. (ie. the top lines will roll off the screen as new lines appear at the bottom.)

The default is to scroll the entire display. To retain a portion of the display on the screen, call this routine to define the the roll point.

Syntax:

```
CALL 'ROLLPT' USING ROLL-POINT.
```

Where:

ROLL-POINT The new roll point for the terminal. This field must be a binary halfword representing the number of lines to "freeze" at the top of the CRT.

Example:

```
77 ROLL-POINT PIC 9(3) COMP-4 SYNC VALUE 4.
```

Using a value of 4 (as in the example above) will cause the top four lines of the display to remain on the screen while the lower lines are rolled as necessary.

-+*+-

7.9.8 GET ONE LINE FROM TERMINAL TEXT

The TEXT subroutine will retreive an input message of up to 64 characters without parameterization.

Syntax:

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

Where:

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

Example:

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

-+*+-

7.9.9 GET ONE LINE FROM TERMINAL TEXT8

TEXT8 is similar to PARAM, except that a single 80 character field is retrieved and no parameterization is performed.

Syntax:

```
CALL 'TEXT8' USING TEXT-AREA.
```

Where:

TEXT-AREA The area where the 80 bytes of terminal input is to be placed (without parameterization).

Example:

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

-+*+-

7.9.10 ATTACH AN ALTERNATE TERMINAL TIPATTCH

When a program is loaded, TIP assumes that the terminal that requested the program will be the source of input for the program and the recipient of any messages sent from the program (ie. the original or primary terminal).

The TIPATTCH subroutine allows the program to select another terminal as its primary input device. The selected terminal must be in an idle state (no user logged on) before it can be selected as an alternate terminal. The attach function does not immediately use the alternate terminal (see TIPUALT), but reserves it for later use by the calling program.

Upon completion of this call the user should check the contents of the parameter field. If the call was rejected, this field will contain the value "NTRM" which indicates that the terminal name is not valid, or that the terminal is currently not idle.

Syntax:

```
CALL 'TIPATTCH' USING TERM, WORKAREA.
```

Where:

TERM The four character terminal name of the desired alternate terminal.

WORKAREA A four fullword workarea available to the TIPATTCH subroutine. This field must be fullword aligned.

Example:

```
77  TERM          PIC X(4)    VALUE 'TRM7'.
05  WORKAREA      PIC 9(6)    COMP-4 SYNC.
  10  FILLER      PIC X(12).
```

-+*+-

7.9.11 SEND PRINT CODE TO AUX PRINTER TIPCOP

To simplify the handling of an auxiliary printer the user may call TIPCOP to send the PRINT command to the 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. Default values are the last row and the calling terminal.

Syntax:

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

Where:

ROW The line number of the last line to be printed on the auxiliary printer.

TERM-NAME Optional name of the destination terminal. Default is the terminal that ran the program.

Example:

```
05  ROW        PIC 9(4) COMP-4  SYNC  VALUE 16.
05  TERM-NAME  PIC X(4)                VALUE 'T309'.
```

-+*+-

7.9.12 SET UTS-400 CONTROL PAGE TIPCPAGE

The user program may set the Transmit Field of a UTS-400 Control Page 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 data

"VAR " - transmit variable (data only)

"CHAN" - transmit only changed data

Example:

```
05 CPAGE-OPTION     PIC X(4)   VALUE "VAR".
```

-+*+-

7.9.13 DETACH ALTERNATE TERMINAL TIPDETCH

This function allows the user program to detach a terminal that was previously attached via TIPATTCH. This will free the detached terminal from the program allowing it to be used as a normal terminal. The detach function will reset the programs original state if a TIPUORG call had not been done.

Syntax:

```
CALL 'TIPDETCH' USING WORKAREA.
```

Where:

WORKAREA A four fullword workarea for use by the TIPDETCH subroutine. This field must be fullword aligned.

-+*+-

7.9.14 SCAN PARAMETERS FROM STRING TIPSCAN

This subroutine is very useful for parsing (scanning) fields from a string of characters. It utilizes delimiters which are specified by the calling routine. The following example would place the first field (found in ISTRING) which is 8 bytes or less long and ending with any of the specified delimiters (",/ ") into OSTRING-TXT.

IPTR will be set to the zero relative offset into ISTRING of the next character after the delimiter.

Repeated calls to TIPSCAN will scan out all such fields. The user must check the value of IPTR to determine when the end of ISTRING is reached.

Syntax:

```
CALL 'TIPSCAN' USING ISTRING, IPTR, OSTRING, DELIMS.
```

Example:

```
01  ISTRING                    PIC X(??).

01  IPTR                       PIC 9(3) VALUE 0 COMP-4.

01  OSTRING.
    02  OSTRING-LL             PIC 9(3) VALUE 8 COMP-4.
    02  OSTRING-TXT            PIC X(8).

01  DELIMS.
    02  DEL-LL                 PIC 9(4) VALUE 3 COMP-4.
    02  FILLER                 PIC X(3) VALUE ',/ '.
```

-+*+-

7.9.15 USE ALTERNATE TERMINAL TIPUALT

If an alternate terminal has not been reserved by a prior call to the TIPATTCH subroutine this call will be ignored. After calling TIPUALT all further input requested by the program will be solicited from the alternate terminal.

Similarly, all output messages will be sent to the alternate terminal unless specifically directed to another terminal.

Syntax:

```
CALL 'TIPUALT' USING WORKAREA.
```

Where:

WORKAREA A four fullword workarea for use by the TIPUALT subroutine. This field must be fullword aligned.

-+*+-

7.9.16 USE ORIGINAL TERMINAL TIPUORG

This function allows the program to switch back to the original terminal. After calling TIPUORG all subsequent terminal input will be solicited from the original terminal. Similarly, all output messages will be sent to the original terminal unless specifically directed to another terminal.

Syntax:

```
CALL 'TIPUORG' USING WORKAREA.
```

Where:

WORKAREA A four fullword workarea for the TIPUORG subroutine; This field must be fullword aligned.

-+*+-

7.10 DIRECT COMMUNICATIONS I/O Direct I/O

TIP/30 provides facilities that a user program may use to directly interface with the communications sub-system. This Direct Communication I/O interface is at a primitive level - that is, the user assumes the responsibility for generating the proper control information for the devices being manipulated.

With Direct Communications I/O, the user program interfaces with ICAM (the operating system communications control code) via calls to a TIP subroutine named "TIPTERM".

The user program is responsible for issuing messages that conform to ICAM specifications and which will produce the desired effect at the destined terminal. It is also the responsibility of the user program to decode all input messages and filtering user data from imbedded terminal-dependent control codes.

Direct Communications I/O is normally used in BAL programs. A complete set of macros for display programming is provided for the BAL programmer. It should be noted that all the message processing features of the TIP Message Control System (MCS) are available to BAL programs.

COBOL applications should take advantage of the extensive display handling capabilities of the Message Control System (MCS).

The documentation in this section of the manual requires a pre-requisite knowledge of the facilities of OS/3 ICAM and a familiarity with assembly-language programming. The examples are given in standard assembly language format.

7.10.1 INPUT AND OUTPUT MESSAGE FORMAT DCIO: prefix

All input and output messages must be preceded by a fixed format message prefix. Two BAL macros ("TIPIMP" and "TIPOMP") are provided to generate these prefixes.

Each macro generates a standard message prefix as documented in the Communications Manuals (ICAM) supplied by the manufacturer with one exception. An extra word is added at the beginning of each prefix and is used only by TIP, the remainder of the prefix is in fact the normal ICAM message prefix.

The label field is used to name the message and is referenced in the CALL TIPTERM instruction. It may be 1 to 7 characters in length.

The first parameter is used to reference the end of the message; this is required to generate the correct length value within the prefix. The length field (halfword) within the prefix may be referenced in a user program by affixing the suffix "L" to the label.

The output message prefix has an optional keyword parameter that is used to indicate that the message is to be printed on an auxiliary device when it is sent to a terminal.

The keyword parameter is coded as "PRINT=YES". The codes generated in the output prefix are for regular print (non-transparent) on auxiliary device one (AUX1).

Example:

```
*
* * *     INPUT MESSAGE PREFIX     * * *
*
INMSG     TIPIMP    INMGE                .
IMA       DS        CL100                . INPUT MESSAGE AREA
INMGE     EQU       *                    .
*
* * *     OUTPUT MESSAGE PREFIX    * * *
*
OUTMSG    TIPOMP    OUTMGE,PRINT=YES     .
OMA       DC        C'This is my output message'
OUTMGE    EQU       *
```

-+*+-

7.10.2 AUXILIARY DEVICE I/O DELIVERY STATUS DCIO: status

For all output messages sent to auxiliary devices, the sending user program is not re-scheduled until the message has been delivered.

For Direct I/O (CALL TIPTERM), the delivery status is returned in the first byte of the TIPOMP packet.

All messages sent to auxiliary devices are sent on the LOW priority communications queue, while messages sent to the main terminal are sent on the MEDIUM priority queue. This allows you to send an informational message to the terminal operator if the auxiliary device encounters an error. The delivery status codes are tabled below:

STATUS CODE	DESCRIPTION
BLANK	Good delivery
B	Line Down
C	Terminal Down
D	Invalid Destination
E	No available buffers
F	Disk I/O error
G	Invalid output message length
0	Auxiliary device down
1	Read/Write inoperative
2	Printer out of paper
3	End of cassette
4	No response

In all cases (other than "good delivery") the message involved with the error is deleted and the application program must take some recovery action.

-+*+-

7.10.3 GENERATE CARRIAGE RETURN DCIO: carret

This macro generates the control codes to "leave" a specified number of lines at this point in the output message.

Syntax:

CARRET (row,col) [, SOE]

Where:

(row,col) Optional number of rows to leave and number of columns of spaces to generate at the beginning of the next row. If omitted, no blank lines or spaces will be generated.

SOE Optionally generate an SOE character at the begining of the next line.

Example:

CARRET (2,0),SOE - leave two blank lines and put an SOE at the start of next line

-+*+-

7.10.4 CURSOR POSITIONING — DCIO: cursor

This macro will generate the control codes needed to position the CRT cursor.

Syntax:

CURSOR location [,SOE]

Where:

location The desired cursor location. The character string "HOME" is recognized as meaning the top left corner of the CRT. This parameter may also be coded as: "(row,col)" where row and col specify the desired row and column location (in decimal).

SOE Optional parameter indicating by its inclusion that an SOE (start of entry) character is to be generated at the specified location. Coded as "SOE".

Example:

```
CURSOR  HOME      : move cursor to home position
CURSOR  (3,54)    : move cursor to row 3 column 54
CURSOR  HOME,SOE  : move cursor to home location and generate
                    an SOE character.
```

-+*+-

7.10.5 DELETE FUNCTION DCIO: delete

This macro generates the control codes to delete a line. It may also be used to generate a delete character in the line or in the display with wraparound. (ie: UTS400 key DELETE IN DISP or DELETE IN LINE).

Syntax:

DELETE type [,(row,col)]

Where:

type The type of delete desired:

"LINE" - delete an entire line in the display. Note that all lines below the deleted line will be shifted up.

"INLINE" - delete a character in the line and shift all characters following in the line right one position.

"INDISP" - delete a character in the display and shift all characters following in the display left one position.

(row,col) Optional row and column where the cursor is to be placed before performing the delete function. If omitted the cursor will remain wherever it was.

Example:

```
DELETE  LINE,(5,6)    - go to row 5 col 6 and issue a
                        delete line function. Note the
                        cursor would remain at (5,6)
                        and lines 6 to end would be rolled
                        up one line.
```

-+*+-

7.10.6 ERASE FUNCTION — DCIO: erase

This macro will generate the control codes to erase a line, field or the entire display.

Syntax:

ERASE type [,(row,col),SOE]

Where:

type The type of erase to be performed:

"DISPLAY" - erase the screen from the row,col specified in parameter two to the end of the screen. If parameter two was not coded, assume entire screen. This command erases unprotected areas of the screen.

"PROTECTED" - the same as DISPLAY (described above) with the exception that both protected and unprotected portions of the screen will be erased.

"FIELD" - erase unprotected field at the location given by parameter two. If parameter two is omitted, the current field will be erased.

"LINE" - erase to end of line at the location given by parameter two. If parameter two is omitted, the current line is erased.

(row,col) Optional row and column location to place the cursor before the specified erase operation takes place.

SOE Optionally place an SOE character at the location given by the (row,col) parameter before performing the erase function.

Example:

ERASE DISPLAY,(4,1),SOE - place SOE at (4,1) and erase (unprotected) to end of screen.

-+*+-

7.10.7 GENERATE FIELD CONTROL CHARACTERS DCIO: fcc

This macro is used to generate Field Control Characters for use with the UTS-400 style terminal (including UTS20, UTS40, Q310, DELTA 2400 etc).

Syntax:

```
FCC   (row,col),[ Normal ] , [ Skip    ] , [ Unprotected ] , [ Right ]
                [ Low    ]   [ Tab     ]   [ Protected   ]
                [ Blink  ]   [ Alpha   ]
                [ Off    ]   [ Numeric ]
```

Where:

(row,col) Required parameter indicating the row and column of the FCC.

Example:

```
FCC  (1,1),B,NUM,UNP   - set up a blinking, numeric,
                         unprotected field at row 1, col 1.
```

Additional Considerations:

Positional parameters 2 through 5 are as described in the UTS400 manual. If any or all of those parameters are omitted the defaults used are: NORMAL,SKIP,UNPROTECTED.

Only the first character of positional parameters 2 through 5 is checked by the macro.

-+*+-

7.10.8 INSERT FUNCTION — DCIO: insert

This macro will generate the control codes to perform the INSERT function. It may be used to insert a line in the display (thus pushing down all lines below) or to insert a character in the line or in the display (ie: UTS400 key INSERT IN LINE or INSERT IN DISP).

Syntax:

```
INSERT  type  [,(row,col)]
```

Where:

type The type of insert function to perform:

"LINE" - insert a line in the display.

"INLINE" - insert a space in the line. All characters in the line and to the right of the space inserted will be shifted to the right with no wraparound.

"INDISP" - insert a space in the display. All characters in the display after the inserted space will be shifted to the right with wraparound.

(row,col) Optional row and column to position the cursor before the insert function is performed.

Example:

```
INSERT  INLINE,(9,12)   - go to row 9, col 12 and insert a
                          space inline.
```

-+*+-

7.10.9 ROLL THE SCREEN — DCIO: roll

This macro generates the control codes to ROLL the screen up or down a number of lines from a given co-ordinate location.

To roll the screen up, a suitable number of "delete line" commands is generated; to roll down, "insert line" commands are generated.

Syntax:

ROLL direction [,lines, (row,col) ,SOE]

Where:

direction The string "UP" or "DOWN" indicating the direction to roll the display.

lines Optional number of lines to roll. Default is 1.

(row,col) Optional location of the roll to occur. Default is the home position (1,1).

SOE The string "SOE" indicating that an SOE character is to be placed in the begining of the next line.

Example:

```
ROLL  UP,5          - roll display up 5 lines
```

-+*+-

7.10.10 SCAN FUNCTION **DCIO: scan**

This macro will generate the control codes to move the cursor a specified number of positions in any direction (up, down, left, right).

Syntax:

SCAN direction [,n]

Where:

direction The direction to move the cursor:

"UP" - move the cursor up from its current location;

"DOWN" - move the cursor down from its current location;

"RIGHT" - move the cursor to the right of its current location;

"LEFT" - move the cursor to the left of its current location;

Example:

SCAN LEFT,40 - move the cursor to the left 40 positions.

Additional Considerations:

If the cursor crosses a screen boundary (ie: top, bottom, leftside, rightside) while 'scanning', it will behave exactly as the arrow keys on the keyboard. For example, if you scan "UP" and hit the top of the screen, the cursor goes to the bottom and continues moving upward.

-+*+-

7.10.11 TAB FUNCTIONS DCIO: tab

This macro will generate control codes to set tab stops on the screen or position the cursor at the next tab stop on the screen.

Syntax:

TAB type [,(row,col)]

Where:

type A choice of "SET" or "MOVE" indicating whether a tab is to be set at the location given by parameter two or whether to move to the next tab position.

(row,col) Optional specification of a row and column to move the cursor before the tab operation is performed.

Example:

```
TAB  SET,(24,1)      - set a tab at row 24, column 1.
```

Additional Considerations:

If a move operation is specified when there are no tab stops on the screen, the cursor will go to the home position.

-+*+-

7.10.12 TRANSMIT FUNCTION **DCIO: xmit**

This macro generates the control codes to simulate pressing the transmit key (auto-transmit).

Syntax:

TRANSMIT [PROTECTED]

Where:

PROTECTED Optionally indicates that only protected data is to be transmitted. If omitted, all data will be transmitted.

Example:

TRANSMIT PROTECTED - will generate an auto-transmit of protected data.

DCIO: yes/no"

-+*+-

7.10.13 YES/NO FUNCTION

This macro will generate the control codes for a common terminal prompt; namely, a YES or NO query. The prompt may be issued with "YES" as the default reply or "NO" as the default reply.

Syntax:

```
      YESNO
  or  NOYES
```

Where:
No parameters required.

Example:

YESNO

Will generate the following sequence:

>YES._ >NO.

NOYES

Will generate the following sequence:

>NO._ >YES.

Where the ">" represents an SOE character, the "_" shows the cursor resting location and the "." is a TAB STOP.

Additional Considerations:

This usually would be the last part of the text of an output message.

-+*+-

7.10.14 TIPTERM FUNCTIONS DCIO: tipterm

User programs may request direct terminal I/O services by calling the supplied subroutine "TIPTERM" with parameters indicating the desired function and (optionally for some functions) the appropriate input or output message area.

The input and output message areas are defined using the TIPOMP and TIPIMP macros described elsewhere.

Following is a summary of the function codes available (they represent equated symbols generated by the TP$BEGIN macro):

T@CNTRL	CONTROL TERMINAL INPUT
T@FREE	ALLOW FREE TERMINAL INPUT
T@GET	GET A MESSAGE FROM A TERMINAL
T@PHONE	CHANGE DIAL-UP LINE TEL NUMBER
T@PUT	PUT A MESSAGE TO A TERMINAL
T@TEST	TEST FOR UNSOLICITED TERMINAL INPUT
T@UN	SEND AN UNSOLICITED TERMINAL MESSAGE

-+*+-

7.10.15 CONTROL TERMINAL INPUT — TIPTERM: cntrl

This function allows the program to control input from a terminal. The normal mode of operation of a terminal is "controlled". This means that every input message to the system must be answered by at least one output. This call is usually given at a point in the program after a T@FREE has been used.

Syntax:

```
CALL   TIPTERM,(T@CNTRL)
```

Where:

No other parameters required.

-+*+-

DISCONNECT DIAL-UP LINE

7.10.16 DISCONNECT DIAL-UP LINE — TIPTERM: disc

This function allows the program to disconnect a dial-up line connection.

Syntax:

```
        CALL   TIPTERM,(T@DISC,DISMSG)
*
DISMSG  TIPOMP  DISE              .
        DC      CL4'????'         . ICAM LINE NAME HERE
DISE    EQU     *                 .
```

-+*+-

7.10.17 ALLOW FREE TERMINAL INPUT — TIPTERM: free

This function allows the program to retrieve several input messages from a terminal without having to reply with an output message after each input.

This function is the logical opposite of the controlled mode (see TIPTERM: cntrl). It is called once to set the mode of operation of the terminal. It is used for file transfers from Bi-sync devices (eg.IBM 2780).

All batch mode terminals such as the IBM 3741 and 2780 are automatically set to free input at TIP startup.

Syntax:

```
CALL   TIPTERM,(T@FREE)
```

Where:

No other parameters required.

-+*+-

7.10.18 GET AN INPUT MESSAGE **TIPTERM: get**

This function is used to get input from the terminal. It is normally issued after a T@PUT. Although the input message may not be available immediately, (ie. the operator may be entering data in a display), TIP knows that a request for input has been made and when it is available from the associated terminal TIP will move the message to the input message area of the specified TIPIMP and re-activate the program at the instruction after the call to TIPTERM.

Syntax:

```
        CALL   TIPTERM,(T@GET,INMSG)

INMSG   TIPIMP INMSGE          .
INMSGTXT DC    CL80' '         . This application will accept up to
INMSGE  EQU    *               . 80 input characters
```

Where:
No other parameters are required.

Additional Considerations:

After the above call is performed, and the response is received, the program may determine the status of the input message by checking the first byte of the Input Message Prefix (TIPIMP).

STATUS	DESCRIPTION
T$ERR	Truncated input. Input Message Area is too small.
T$OK	Input message received ok.
T$FUNC	Function key received

-+*+-

7.10.19 CHANGE DIAL-UP LINE TELEPHONE NUMBER TIPTERM: phone

This function allows the user to change the telephone number of a dial-up line.

Syntax:

```
         CALL   TIPTERM,(T@PHONE,PHNMSG) .
*
PHNMSG   TIPOMP PHNE                .
         DC     CL4'line name'      .
         DC     C'phone number'     .
PHNE     EQU    *
```

-+*+-

7.10.20 OUTPUT A MESSAGE **TIPTERM: put**

This function is used when the user program wants to output a message to a terminal. The message should contain terminal control codes to format the display at the destination terminal. The message must be set up in the output message area of a TIPOMP.

Syntax:

```
CALL   TIPTERM,(T@PUT,OUTMSG)
```

Where:
No other parameters required.

Example:

The following output message might be used (as illustrated above) to home the cursor and clear the screen.

```
*
OUTMSG   TIPOMP OUTE            .
         CURSOR HOME            . generate cursor to home
         ERASE  PROTECTED       . and erase display
OUTE     EQU    *               .
```

-+*+-

7.10.21 TEST FOR INPUT **TIPTERM: test**

This function allows the user to determine if any unsolicited input has occurred. This is frequently used as a 'break' function in programs which generate continuous output. For example, a program to display an item master file may generate many lines of output (rolling the screen). By testing for input after every 'n' lines of output the program could determine if input had been generated (ie. operator pressed Message Waiting) and send a message to the operator to ask if continuation is desired.

Syntax:

```
CALL   TIPTERM,(T@TEST,INMSG)
```

Additional Considerations:

After the above call is completed, the program must check the first byte of the input message prefix (TIPIMP) to determine if a message has been received.

STATUS	DESCRIPTION
T$ERR	No input available.
T$OK	Input message received.
T$FUNC	Function key received

-+*+-

7.10.22 SEND AN UNSOLICITED MESSAGE — TIPTERM: un

This call 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 will send the message on the low priority message queue. In addition, TIP will send a sona-alert message on the high priority message queue (causing the Message-Waiting alarm to sound at the terminal).

The message will not be displayed until the terminal operator presses Message Wait. The terminal operator should position the cursor to an unused line on the CRT prior to pressing Message Wait.

Syntax:

```
CALL   TIPTERM,(T@UN,OUTMSG)
```

-+*+-

8. CHAPTER VIII - SYSTEM MAINTENANCE TIPGEN

This chapter of the TIP/30 reference manual contains information that is needed by system programming personnel for the maintenance and operation of the TIP/30 system.

To make effective use of the information in this chapter, the reader should be familiar with the operating system (OS/3) and the communications interface (ICAM).

Familiarity with assembly language programming would be an asset.

8.1 TIP/30 LIBRARY FILE REQUIREMENTS

There are two OS/3 libraries that are required by TIP/30. The first library (// LFD TIP) is used to hold the TIP/30 release system. This file is also used in the generation of TIP/30, as the PROC library during assemblies, as the ALIB library during a link-edit, and as a copy library for COBOL 74 compiles.

The user must NOT put any of his own modules in this library. This will simplify loading a new release of TIP/30.

The second file (// LFD TIPLOD) must contain the TIP/30 load modules, TIP support programs and TIP utilities as well as all user-written transaction programs. When the user load modules are copied into this library, they may be written in block load format (LIBS BLK Command). This procedure improves library searching and reduces initial program load times to a minimum when running TIP/30.

Note: All Y system files are automatically assigned to TIP/30. (Example: YJCS, YLOD etc).

EXECUTION TIME WORK FILES

8.2 EXECUTION TIME WORK FILES workfiles

TIP requires three additional disc files during execution. The first is the TIP Catalogue File (SAT) and is used to hold MCS Display formats, USER-IDs, logical program names and FCS Dynamic file header records. This file should be about 8 to 12 cylinders.

TIP uses several SAT (System Access Technique) files. These files are NOT library files. They cannot be copied or listed with LIBS. Attempting to use any function of LIBS with these files could destroy the files! The program "DMPRST" may be used to move these files.

The second file is the TIP Random File (SAT) and is used for the allocation of Dynamic FCS files, and edit buffers. Each FCS random file begins at 40 blocks of 512 bytes each and grows in units of this size as required. The size of this file varies according to the requirements of dynamic files and edit buffers. A typical starting value is about 10 to 16 cylinders.

The third required file is the TIP Swapping File (SAT) and is used for high speed physical I/O swapping storage. It must be large enough to hold a copy of each user program for each terminal in the system, ie. ((terminals * largest prgm * 2) / cyl capacity) + 4. Part of the Swap File is used by TIP/30 as the storage for the TIP Transients. Try about 16 cylinders to start. If a mix of sectored and non-sectored discs are available, this file is best placed on the non-sectored units.

EXECUTION TIME WORK FILES

SUMMARY OF FILES REQUIRED BY TIP/30

FILE	REQUIRED LFD-NAME	TYPE	COMMENTS
TIP CATALOGUE	TIP$CAT	SAT	- Catalogue entries for user-ids, programs and files. - MCS screen formats.
RANDOM FILE	TIP$RNDM	SAT	contains FCS dynamic files and Edit Buffers
SWAPPING FILE	TIP$SWAP	SAT	Swapping storage for Programs, Work Areas and File buffers.

8.3 TIP/30 SYSTEM GENERATION TIPGEN

To generate a TIP/30 system the user must prepare an input stream similar in concept to the OS/3 Supervisor and ICAM generation streams.

This input stream, containing generation control statements, parameters, keywords and options is read and interpreted by a batch program called TB$GEN.

TB$GEN creates a JCL element containing the correct ASSEMBLER statements to generate a TIP/30 system. The responsibility of conforming to Assembler coding conventions is assumed by TB$GEN.

The user prepares free format statements defining the TIP/30 requirements of the site.

There are three basic types of generation control statements:

TIPGEN	defines the TIP/30 Control Area
FILE	defines all on-line files
CLUSTER	defines on-line terminals

These generation control statements have required parameters and optional keywords which specify variable information. The following sections will describe the generation control statements and the parameter and keyword requirements.

Many of the keywords described have short forms. The keywords are illustrated as a mix of upper and lower case letters.

The short form of a keyword is the sequence of upper case letters. The lower case letters may be omitted.

To avoid confusion, it is recommended that the user specify keywords even if the default value is satisfactory; new releases of TIP/30 may result in a change in the default value for a keyword and thus cause some grief for unwary system programmers.

8.3.1 TIPGEN DEFINITION TIPGEN

The TIPGEN control statement begins the definition of the characteristics of the TIP/30 Monitor for your site.

The information and tables constructed by this part of the generation are known as the TIP/30 Control Area (TCA). The TIP/30 monitor is released as one load module (TB$TIP).

When TB$TIP (or the TCA name) is executed it loads in the TCA whose name is given in the job control data set and modifies internal tables with information supplied in the TCA.

The TCA name must be at least 3 characters long, and the first 3 characters must be unique if several TCA's are being generated.

TIPGEN DEFINITION

Syntax:

		Description	Default
TIPGEN	tcaname	. required positional parameter	
	AFT=	. Number of active files/terminal	MAX
	BaCK=	. Number of background programs	2
	B4=	. Before image file YES or NO	NO
	CATPooL=	. Number of catalog buffers	6
	CDM=	. Consolidated Data Mngmnt	NO
	DBMS=	. Data Base to be used	NO
	DECIMAL=	. Define European decimal point	DECIMAL
	DM=	. shared data management used?	Rev 6: NOSHARE
		.	Rev 7: SHARE
	ESCape=	. System escape character	@
	FaSTLoaD=	. Fastload index size	(25,50)
	FCSxtent=	. FCS Dynamic file auto increment	40
	FiLeBufs=	. Number of file buffers	3
	FREEm=	. Additional free memory needed	2560
	GDA=	. Global data area size	0
	JOB=	. JOB name of next gen step	TJ$GEN
	JouRNaL=	. Disk journaling YES or NO	NO
	KeYTaBLe=	. Key holding table	(0,0)
	LIST=	. Generation list option	NO
	LOG=	. Tape logging YES or NO	NO
	LoGoN=	. logon is required	YES
	MaXPRoG=	. Maximum program size	30000
	MAXTiMe=	. Maximum time before calling TIP	30 sec.
	MSGPooL=	. Message pooling (number,size)	(3,500)
	NeTWoRK=	. ICAM network (CCA,TCIPWD)	(TIPC,TIPPWD)
	PRSTEN=	. Journal 'PRST' and 'PREN'	NO
	ReaDYmsg=	. send ready message at startup	NO
	shutDowN=	. Automatic shutdown program	''
	StartUP=	. Automatic startup program	''
	SITEid=	. Site identification	'TIP30'
	STatS=	. Statistics interval	(10,60)
	TCP=	. Command processor name	'TCP'
	TeRMS=	. Number of on-line terminals	<required>
	TeRMSiZe=	. Terminal screen size	(24,80)
	TeRMTYPe=	. Type of terminal	UNISCOPE
	TIMeoff=	. Time to auto logoff (minutes)	5 min.
	TIMeouT=	. External succession timeout	540 min.
	TIPFILES=	. JPROC to be called by job	TIPFILES
	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

Where:

tcaname is the name of the TIP/30 Control Area.

This name is used as the name of the TIP/30 Generation and must be specified as the first parameter of the TIPGEN control statement.

The TCANAME is given in the TIP/30 Job Control to identify the TCA to TB$TIP. This may also be directly executed (// EXEC tcaname).

AFT=n is the average number of active files expected per terminal. This parameter is used to pre-allocate memory for storage of 'Active File Tables'. If omitted TIP will insure that there is enough room for every terminal to have every file open at the same time.

BACK=n is the maximum number of background programs which may be running at any one time. Default=2.

B4=YES defines a separate 'before image file' to be used by TIP/30 for automatic rollback of data files. This parameter can be used with disk journaling (JRNL=YES) or with tape logging (LOG=YES). Whenever records of files defined as HOLD=TR are read for update a copy of the record before update is stored in the TIP$B4 file until the transaction successfully terminates. Default=NO.

CATPOOL=n is the number of catalogue buffers. Increasing this value may improve performance. Each buffer is 256 bytes. Default=6.

CDM= Whether or not Consolidated Data Management is to be used by TIP/30.

Default is NO - implies use of Basic Data Management (DTF's).

DBMS=name,size 'name' is the name of the Data Base Management interface module to be used. The object module must be in the TIP library and be called TS$'name'. With entry points of TS$'name' for I/O, TS$'name'OP for OPEN routine, and TS$'name'CL for CLOSE routine. Supplied interface modules include TS$TTL, TS$DMS, TS$DBS.

'size' is the number of bytes to be reserved for the Data Base interface as a work area. For example TOTAL (TS$TTL) needs storage to hold the DBMOD and TOTAL/7.

DBMS=DMS interface to single thread DMS/90.

DBMS=DMT,n interface to multi thread DMS/90, where 'n' is the number of threads.

DBMS=DBS interface to single thread DBS/90.

DBMS=TTL,n interface to TOTAL/7, where 'n' is the amount of memory to be reserved for the DBMOD and all needed TOTAL modules. and TOTAL/7. If this memory is too small TIP/30 may abort.

DECIMAL=COMMA reverses the meaning of '.' and ',' in M.C.S. editing of numeric fields. ',' becomes the decimal point. Default=DECIMAL.

DM=SHARE causes the TCA to be generated to allow the use of shared data management. This may reduce memory requirements. If this option is used then the user must EXECute the TCA instead of 'TB$TIP'. This must be done to have the OS/3 supervisor setup the shared code linkages. Default is NOSHARE for OS/3 6.x. Default is SHARE for OS/3 7.x and up.

ESCAPE=X defines the system escape character which when encountered by TIP/30 as the first character of an input message will 'escape' the current program to the TIP/30 Command Processor and increment the program stack. Default=@.

FASTLOAD=(m,n) 'm' is the number of fastload index entries kept for memory relocated programs. 'n' is the number of fastload index entries for disc addresses of programs in the TIP$SWAP file. Default is (25,50).

This table is only used for non-re-entrant programs.

FCSXTENT=n is the size of one dynamic file extent. A dynamic file may have a maximum of 48 extents. If this value must be changed after TIP has been running, it will be necessary to insure that there are no dynamic files in existence. Default=40.

FILEBUFS=n is the number of buffers to be reserved for loading DTFs at execution time. Each buffer is individually sized to the largest DTF residing in that buffer. Some attempt should be made to group non-conflicting FILEBUFS (DTF's) by size. The absolute size includes the DTF, IOAREA, INDEX AREA and one work area the size of one record. Default=3.

FREEM=n 'n' is the amount of extra memory to be reserved for additional functions. For example an OS/3 spool work area used by SPL and BCP programs is 1200 bytes. TIP/30 will always reserve at least 3K bytes; anything specified here is in addition to the basic 3K. Default=2560.

The SPL program and ODD and QED make use of free memory.

GDA= The size in bytes of the optional Global Data Area. Default is zero.

JOB= is the element and job name used when the output of the generation program is written to YJCS. Default=TJ$GEN.

JOURNAL=YES updates are to be written to TIP$JRN. A recovery module will be generated for use by the file recovery program TB$RCV.

Default=NO.

KEYTABLE=(m,n) 'm' is the maximum number of records to be held for update at any one time.

'n' is the size in bytes of the largest key used. If nothing is specified the generation program will generate the table with 2 entries for each file, and will determine the largest key used automatically. Default (0,0).

LIST=YES the generated code for the TIP system generation is to be listed. Default=NO.

LOG=YES Indicates that file updates are to be written (logged) to the TIP/30 log tape (TIP$LOG). A recovery module will be generated for use by the file recovery program TB$RCV. Default=NO.

LOGON=NO specfies that all terminals do not require users to log on to TIP/30 before running transactions. Default=YES.

MAXPROG=n 'n' is the maximum size program in bytes, TIP/30 expects to load. TIP/30 will insure that a memory region of 'n' bytes is available. Default=30000.

MAXTIME=n the maximum wall time to be elapsed by a program without calling some TIP/30 function. TIP/30 will abort the program on the assumption that it is looping. Default=30.

MSGPOOL=(n,m) 'n' is the number of message file buffers to be allocated and 'm' is the size of the buffers. This parameter becomes important if you have a 'high transaction' volume and your application programs use several different messages in a cycle. Messages larger than 'm' are not pooled. Default=(3,500).

NETWORK=(cca,pwd) 'cca' is the ICAM CCA name. 'pwd' is the ICAM CCA password. Default=(TIPC,TIPPWD).

NETWORK may be changed by job control parameters.

PRSTEN=YES specfies that the statistcal journal record 'PRST' (program start) and 'PREN' (program end) are to be written. If these records are of no real interest to you then leave this parameter as 'NO' and reduce journal I/O's. Default=NO.

READYMSG=YES this will cause TIP/30 to send all terminals a short 'TIP/30 READY' message when the system starts up. Default=NO.

You may in addition (via job control) specify Banner1='string1' and Banner2='string2'.

SHUTDOWN=trid is the name of a program that is to be started as a background program automatically when TIP/30 goes to end of job via the EOJ command. Default is ''.

This program must be catalogued in the TIPY group.

SITEID=siteid is the name of the company or site where TIP/30 will be run. This appears in console messages from TB$TIP. It should be a single word of 12 characters or less in length or enclosed in quotes. Default=TIP30.

STARTUP=trid is the name of a program that is to be started as a background program automatically when TIP/30 starts up. Default is ''.

This program must be catalogued in the TIPY group.

STATS=(jrn,cnsl) 'jrn' specifies the time interval used to control the writing of journal 'STAT' records. 'cnsl' specifies the time interval used to control the display of the system statistics on the operator's console. Both values represent a time interval in minutes. Short form STS. Default is (10,60).

TCP= The name of a non-standard Tip Command Processor to be used.

This parameter should only be specified on specific instructions from Allinson-Ross personnel.

TERMS=n number of terminals in the system. 'n' must be greater than or equal to the number of TERM statements specified in the ICAM generation. Required keyword.

TERMSIZE=(m,n) 'm' is the number of rows on the screen and 'n' is the number of columns. Ie. the screen size of TERMTYPE. Default=(24,80).

TERMTYPE=type the most common type of on-line terminal in the network. Default=UNISCOPE; acceptable values are: UNISCOPE, UTS400, U100, U200, U400, UTS20 and UTS400F.

UTS400F is a UTS400 with the character protect feature installed.

TIMEOFF=n is the time in minutes which must elapse before TIP/30 will automatically log a user off the system. This timer is only in effect when the user is not in any application program. Default=5.

TIMEOUT=n is the maximum time which an IMS/90 program may stay in external succession. Default is 540 minutes (also the maximum allowed value).

TIPFILES=jproc is an alternate JPROC to be called by the output job. This is useful for bringing up a new releases of TIP/30 while leaving the currently running release in production. Default=TIPFILES.

WORK1=vol[,dvc] 'vol' is the disk volume which is to be used for the ASM WORK1 file. (This could be RES or RUN) 'dvc' is the device number to be used (omit if specifying RES or RUN).

WORK2=vol[,dvc] 'vol' is the disk volume which is to be used for the ASM WORK2 file. (This could be RES or RUN) 'dvc' is the device number to be used (omit if specifying RES or RUN).

XMIT=M Defines the default control page XMIT setting for UTS400 type terminals.

Default is "do not alter control page". If one of "VAR", "ALL", "CHAN" is specified, the terminals will have the transmit setting of the control page set automatically at logon time.

XMITALL= Specifies the number of a function key which will be interpreted by the TIP system as if it was a request to place the cursor in the bottom right corner of the CRT followed by a TRANSMIT ALL sequence.

EG: XMITALL=22 will cause function key 22 to behave in the manner described above.

XMITCHAN= Similar to <XMITALL>, except that only changed data will be transmitted.

XMITVAR= Similar to <XMITALL>, except that only data fields will be transmitted.

-+*+-

8.3.2 FILE DEFINITION FILE

To access files via the TIP/30 File Control System, Data Management control tables must be generated for each file the user wishes to access on-line.

The generation control statement FILE is used to specify file names and their characteristics.

Syntax:

```
FILE      lfd,filetype      . required positional
          ACCess=           . access                      EXCR
          AFTER=            . After images journaled      YES
          AUTOIO=           . CDM file, force I/O         NO
          BEFoRe=           . Before images journaled     NO
          BLKsiZe=          . block size (required)
          BUFfer=           . DTF buffer number
          CLOSE=            . inhibit file to be opend    NO
          DeLeTe=           . delete flag                 (X'FF',0)
          FILesiZe=         . max. number of records
          HoLD=             . key holding for this file   YES
          INDsiZe=          . index size                  256 .
          IO=               . type of file                INOUT
          JouRNaL=          . updates journalized         NO
          KeYLeN=           . key length
          KeYLoC=           . key location                0
          KEY1=             . MIRAM Key1
          KEY2=             . MIRAM Key2
          KEY3=             . MIRAM Key3
          KEY4=             . MIRAM Key4
          KEY5=             . MIRAM Key5
          OPEN=             . keep file open              YES
          OPTioNal=         . optional file               NO
          PCYLofl=          . percent cyl overflow
          RCB=              . MIRAM Rec. Cntrl. byte      NO
          ReCForM=          . record format               FIXBLK
          RECsiZe=          . record size (required)
          RESident=         . file is resident            NO
          USEFiLe=          . same as other FILE
          VSEC=             . variable sector size        256
```

lfd,filetype 'lfd' is the LFD name of the file as specified in the TIP/30 JCL.

'filetype' is the type of file. These are required parameters and must be specified as the first operands of the FILE gen control statement.

Valid file types are ISAM, DAM, SAM, TAPE, MIRAM, DMIRAM, SMIRAM, PRINT, LIB, PUNCH.

ACCESS=opt is the access option as described in the data management manual. Default is EXCR.

AFTER=NO Specifies that the image of records after update is not to be done. Default is YES.

AUTOIO=YES Specifies that (for Consolidated data management) I/O be automatically performed irrespective of the contents of a file buffer.

BEFORE=YES Specifies that the image of records before update is to be done. Default=NO.

BLKSIZE=n 'N' is the true block size required by the DTF.

BUFFER=n is the file buffer to which this file is to be assigned. This keyword gives the flexibility of assigning files to specific buffers. If the file must be swapped it will be swapped to and from buffer 'n'. Default is "assigned by FCS".

CLOSE=YES will mark the file as not available for on-line use. The file may be made available with main site console command of OPEN. See the section on operator communication. Default=NO.

DELETE=(byte,loc) 'byte' is the logical record delete flag and 'loc' is the zero-relative offset of the flag into the record. The value for 'byte' may be specified as X'__' or C'_'.

Default value for 'byte' is X'FF' (HIGH VALUES).

Default value for 'loc' is the first data position in the record that is NOT part of a key field.

DELETE=RCB is only valid for MIRAM files which have been initially loaded with the Record Control Byte option activated (// DD RCB=YES). When this keyword is specified, TIP/30 issues a MIRAM imperative macro which marks the record as deleted from the file.

FILESIZE=n is the maximum number of records in the file. This parameter is only meaningful for memory files.

HOLD=TR means that file updates will be held for the duration of the transaction. Transaction time is defined as the interval between terminal inputs to the program processing the file. Valid for indexed and direct files. If the transaction aborts before completing all updates are rolled back. To use this option, the TIPGEN parameter B4=YES must be specified.

HOLD=UP means that file records will be held after a GETUP until the update is made. There is no provision for on-line roll back. Multiple records may be held per file.

HOLD=YES means that file records will be held after a GETUP until the update is made or another GETUP is made to the same file. Only one record per file may be held at any one time.

INDSIZE=n is the INDEX AREA SIZE for this file. Default=256.

IO=type type is the I/O type of file. Ie. INPUT, OUTPUT or INOUT. Default=INOUT.

JOURNAL=YES Specifies that journaling is to be done for this file. Default=NO.

KEYLEN=n is the length of the key for the file.

This keyword should be omitted for indexed MIRAM.

KEYLOC=n is the zero relative location of the key in the record. Default=0.

This keyword should be omitted for indexed MIRAM.

KEY1= (size,loc,NDUP,CHG/NCHG)

defines MIRAM index 1

KEY2= (size,loc,DUP/NDUP,CHG/NCHG)

defines MIRAM index 2

KEY3= (size,loc,DUP/NDUP,CHG/NCHG)

defines MIRAM index 3

KEY4= (size,loc,DUP/NDUP,CHG/NCHG)

defines MIRAM index 4

KEY5= (size,loc,DUP/NDUP,CHG/NCHG)

defines MIRAM index 5

OPEN=NO marks file to be not kept OPEN when no program is using it. Unless you have a specific reason you should always specify OPEN=YES.

OPTIONAL=YES Specifies that this file is optional. It may not have been specified in the JCL. When input is requested, an end-of-file status will be returned. When output is attempted, it will be ignored and no error status will be presented. Default is NO.

PCYLOFL=n is the percentage of cylinder overflow for ISAM.

RCB=YES the file has been loaded with the RCB activated. TIP/30 will support it and keyword DELETE=RCB could be used.

RECFORM=n is the record format. Acceptable values are FIXBLK, VARBLK, FIXUNB, VARUNB. Default=FIXBLK.

RECSIZE=n is the size of records in the file.

RESIDENT=YES Makes the file's DTF,index,work...etc. resident. (remains in memory at all times). Default=NO.

USEFILE=file Specifies that this file definition is the same as a preceeding FILE definition defined by 'file'. All parameters must be specified and must be exactly the same as the alternately named FILE definition. This parameter is used to reduce TCA compilation time when many 'characteristicly identical' files are being defined.

VSEC=n For MIRAM files, defines the variable sector index buffer size (see OS/3 Data Mngt.). Default is 256.

-+*+-

8.3.3 CLUSTER DEFINITION CLUSTER

TERMTYPE and TERMSIZE of the TIPGEN control statement indicate the most common terminal in the network. The CLUSTER control statement allows the user to define logical clusters or sets of terminals of uncommon type or characteristic.

This is most useful if there are UTS-400 terminals, since it is important for TIP/30 to know the bypass terminal names associated with master and slave units.

Information generated in these CLUSTER tables is used by TIP/30 to modify internal tables; so that the messages produced for the different terminal types will be of the correct format and size.

All terminal names that are specified in CLUSTER statements (master, slaves) must reference TERM names as specified in the ICAM generation used by TIP/30.

NOTE: These cluster definitions represent logical groupings of terminals with similar characteristics. They have no particular physical significance.

Syntax:

```
CLUSTER   master      . required positional parameter
          BYpass=     . name of UTS400 cluster bypass terminal
          LoGoN=      . is logon required (YES or NO)
          ReaDYmsg=   . send ready message when TIP/30 starts
          SiZe=       . screen size of the terminals in cluster
          SLaVes=     . up to 8 slave terminal names
          TCP=        . alternative TCP program for cluster
          TYPe=       . cluster terminal type
          XMIT=       . default control page XMIT value
```

Where:

master is required and must be specified as the first (positional) parameter of the CLUSTER control statement. This defines the primary terminal of the group of terminals.

Programs running at terminals in this cluster can refer to this terminal by the reserved terminal name "*MST".

CLUSTER DEFINITION

BYPASS=name is the name of the bypass terminal of the cluster.

Programs running at terminals in this cluster can refer to this terminal by the reserved terminal name "*BYP".

LOGON=Y/N NO does not require the user of any terminal in the cluster to logon TIP/30.

YES will require the user of any terminal in the cluster to logon TIP/30.

READYMSG=NO Specifies that a ready message is not to be sent to terminals in this cluster.

SIZE=(m,n) 'm' is the number of rows on the screen and 'n' is the number of columns (ie. the screen size).

Default =(24,80).

SLAVES=(a,b,c...) a,b,c... are the names of up to 8 other terminals in the cluster.

These terminals have similar characteristics.

TCP=name The name of a non-standard TCP program for this cluster of terminals.

TYPE=type is the terminal type of all terminals in the cluster.

Choices: UNISCOPE, U100, U200, UTS400, U400, UTS400F, U400F, UTS10, UTS20, UTS-20, UTS40, UTS-40, TTY, Q310, HAZEL.

Default=UNISCOPE.

XMIT=value is the control page XMIT value to be sent to this cluster at logon time: VAR, ALL, CHAN.

Default is TIPGEN specification.

-+*+-

8.3.4 TIP/30 GENERATION KEYWORD SUMMARY

Some generation control statements have certain constraints and some keywords may only be specified for certain control statements.

The following tables provide a cross reference of gen control statements, limits and keywords.

CONTROL	PARM/KYWRD	OPT	REQ	SPECIFICATIONS
TIPGEN	TCANAME		X	
	B4	X		YES or NO
	DM	X		SHARE or NOSHARE
	FILEBUFS	X		1 to 25
	JOURNAL	X		YES or NO
	LOG	X		YES or NO
	MAXPROG	X		20000,160000
	LOGON	X		YES or NO
	TYPE	X		UNISCOPE U100 U200 U400 U400F UTS400 UTS400F TTY Q310 UTS20
CLUSTER	MASTER		X	Positional parameter
	BYPASS	X		
	LOGON	X		YES or NO
	SIZE	X		
	SLAVE	X		
	TYPE	X		(see TIPGEN:TYPE list)

TIP/30 GENERATION KEYWORD SUMMARY

```
FILE     LFD,TYPE                X
*
**       Where TYPE is ISAM   SAM     DAM
**                     MIRAM SMIRAM DMIRAM
**                     MEM   LIB    PRINT
**                     TAPE  PUNCH
**
**       The valid keywords for the above file types are:
**
```

TYPE	Keyword	Optional	Required	Possible values
ISAM	ACCESS	X		EXC EXCR SRDO SRD
	AFTER	X		YES NO
	BEFORE	X		YES NO
	BLKSIZE		X	
	BUFFER	X		1,25
	HOLD	X		YES UP TR
	JOURNAL	X		YES NO
	KEYLOC		X	
	KEYLEN		X	
	RECFORM	X		FIXBLK FIXUNB VARBLK VARUNB
	RECSIZE		X	
	RESIDENT	X		YES NO
	IO	X		INPUT OUTPUT INOUT
SAM	ACCESS	X		EXC EXCR SRDO SRD
	AFTER	X		YES NO
	BLKSIZE		X	
	BUFFER	X		1,25
	JOURNAL	X		YES NO
	RECFORM	X		FIXBLK FIXUNB VARBLK VARUNB
	RECSIZE		X	
	RESIDENT	X		YES NO
	IO	X		INPUT OUTPUT
DAM	ACCESS	X		EXC EXCR SRDO SRD
	AFTER	X		YES NO
	BEFORE	X		YES NO
	BLKSIZE		X	
	BUFFER	X		1,25
	FILESIZE	X		
	HOLD	X		YES UP TR
	JOURNAL	X		
	RESIDENT	X		
	IO	X		INPUT OUTPUT INOUT

MIRAM	ACCESS	X		EXC EXCR SRDO SRD
	AFTER	X		YES NO
	BEFORE	X		YES NO
	BLKSIZE		X	
	BUFFER	X		1,25
	HOLD	X		YES UP TR
	JOURNAL	X		YES NO
	KEYLOC		X	
	KEYLEN		X	
	RECFORM	X		FIXBLK FIXUNB VARBLK
	RECSIZE		X	
	RESIDENT	X		YES NO
	IO	X		INPUT OUTPUT INOUT
*	Direct access MIRAM			
DMIRAM	ACCESS	X		EXC EXCR SRDO SRD
	AFTER	X		YES NO
	BEFORE	X		YES NO
	BLKSIZE		X	
	BUFFER	X		1,25
	HOLD	X		YES UP TR
	JOURNAL	X		YES NO
	RECSIZE		X	
	RESIDENT	X		YES NO
	IO	X		INPUT OUTPUT INOUT
*	Sequential access MIRAM			
SMIRAM	ACCESS	X		EXC EXCR SRDO SRD
	AFTER	X		YES NO
	BEFORE	X		YES NO
	BLKSIZE		X	
	BUFFER	X		1,25
	JOURNAL	X		YES NO
	RECSIZE		X	
	RESIDENT	X		YES NO
	IO	X		INPUT OUTPUT
MEM	FILESIZE		X	
	HOLD	X		YES UP
	RECSIZE		X	
LIB	BUFFER	X		1,25
PRINT	BUFFER	X		1,25
PUNCH	BUFFER	X		1,25
*	Sequential access TAPE			
TAPE	AFTER	X		YES NO
	BEFORE	X		YES NO
	BLKSIZE		X	

BUFFER	X		1,25
RECSIZE		X	
RESIDENT	X		YES NO
IO	X		INPUT OUTPUT

-+*+-

8.3.5 TIP/30 GENERATION JCL EXAMPLE

TB$GEN will generate a job control stream called 'TJ$GEN'.

```
//        JOB       MY$GEN,,C000,D000
//        WORK1
//        TIPFILES
//        OPTION    JOBDUMP
//        EXEC      TB$GEN,TIP
/$
*
** TIP/30 GENERATION FOR ACE CARD COMPANY
*
TIPGEN    TIPACE  TERMS=12  TERMTYPE=U400  SITEID=ACE-TEST
                  MSGPOOL=(5,1024) JOURNAL=YES FILEBUFS=4
*
** DEFINE ONLINE LIBRARY FILES
*
FILE      ACESRC,LIB
FILE      TSTSRC,LIB
/
**   DEFINE INVENTORY MASTER FILE
*
FILE      INVMST,ISAM BLKSIZE=303 RECSIZE=296
                      KEYLEN=16 PCYLOFL=20 RESIDENT=YES
*
**   DEFINE PAYROLL MASTER AND INDEX FILES
*
FILE      PAYMST,MIRAM  BLKSIZE=1024 RECSIZE=300
                        KEY1=(8,2,NDUP,NCHG)
                        KEY2=(12,14,NDUP,CHG)
FILE      PAYIND,DMIRAM BLKSIZE=512 RECSIZE=128
*
**   DEFINE TERMINAL CLUSTERS
*
CLUSTER   PAY1        TYPE=UTS400,SIZE=(24,80)  SLAVE=(PAY2,PAY3)
                      SLAVE=(PAY4,PAY5)
/*
/&
```

-+*+-

8.3.6 TIP/30 GENERATION PARAMETER RUN TJ$PARAM

This supplied job stream will execute the TB$GEN program. Global parameters are supplied to enable the user to specify the library and element name of the generation parameter stream.

```
//        JOB       TJ$PARAM,,C000
//        GBL       RUN=AUTO,F=SYSGEN,TCA=BOOTCA
//        TIPFILES
//        USERFILE
//        EXEC      WRTBIG
//        PARAM     ' TIP/30 GEN'
//        PARAM     ' PARAMETERS'
//        PARAM     ' FOR &TCA'
//        PARAM     'DAT$'
//        PARAM     ' TIP/30 GEN'
//        PARAM     ' PARAMETERS'
//        PARAM     ' FOR &TCA'
//        PARAM     'DAT$'
//        WORK1
//        EXEC      TB$GEN,TIP
//        PARAM     &RUN.RUN
//        PARAM     IN=&F/&TCA
/&
//        FIN
```

-+*+-

8.3.7 PARAM OPTIONS FOR TJ$PARAM TJ$PARAM: options

The TIP/30 generation statements may be placed in an element in an OS/3 library. To direct the generation procedure to it, include the following in the JCL which runs the TB$GEN program:

```
// PARAM IN=file/elt
```

If the assembly and link edit is to be automatically scheduled (if no errors are detected), include the following:

```
// PARAM AUTORUN
```

If the assembly and link edit is not to be automatically scheduled include the following:

```
// PARAM NORUN
```

-+*+-

8.4 RUN TIME JOB CONTROL OPTIONS TIP: exec

There are several options the user may specify when TIP/30 is executed. These options are entered on run control statements which are free format (similar to the generation control statements).

The information specified on these control statements is used to modify internal tables in TIP/30 and override parameters which were defined by the TIP system generation.

This provides a certain degree of flexibility in that numerous parameters may be changed without having to do a TIP System Generation.

SAMPLE TIP/30 EXECUTION JOB STREAM (BOOTSTRAP SYSTEM)

```
//        JOB       TJ$TIP30,,20000,30000,10,SUP,P,TIP,2X2,BOTH,NOHDR
//        GBL       TCA=BOOTCA,NET=(TIPC,TIPPWD),ICAM=C1,RCV=NO
//        JNOTE     'TIP30 WILL LOAD ICAM &ICAM AUTOMATICALLY.'
//        CC        &ICAM
//        OPR       '*****************************************'
//        OPR       '*                                       *'
//        OPR       '*  T I P / 3 0   V E R S I O N   2 . 5  *'
//        OPR       '*                                       *'
//        OPR       '*****************************************'
//        OPR       'TI01  USING   TCA=&TCA'
//        OPR       'TI02  USING   NET=&NET'
//        OPR       'TI03  USING   ICAM=&ICAM'
//        TIPFILES
//        DATAFILE
//        ICAMFILE
// NOP    &$LBL     TIP$B4              'IF B4 FILE IS TO BE USED'
//        USERFILE
//        IF        ('&RCV' EQ 'NO')NORCV
//        WORK1
//        EXEC      TB$RCV,TIP,1
//        PARAM     &TCA
/$
          QUICK;
/*
//NORCV   &$LBL     TIP$CAT
// NOP    &$LBL     TIP$JRN             'IF JOURNAL=YES SPECIFIED'
//        &$LBL     TIP$RNDM
//        &$LBL     TIP$SWAP
//        &$LBL     TIP$TQL
//        OPTION    SUB,JOBDUMP,NSRCH
//        EXEC      &TCA,TIPLOD,1
/$
&TCA
NETWORK=&NET
IMS=YES
BANNER1='*  BOOT STRAP  *'
BANNER2='*    SYSTEM    *'
/*
/&
//        FIN
```

The following run-time control statements may be specified as parameter cards to TIP/30 (Upper case letters in keywords are required; lower case letters may be omitted):

Banner1=	. banner line 1, for ready message
Banner2=	. banner line 2, for ready message
CLoSe=	. files are not available until opened
ESCape=	. system escape character
IMS=	. 'Y' if using IMS/90 emulation
LNEREQ=	. list of lines to open
Network=	. ICAM CCA network name, and password
RESident=	. programs made resident at initializtion
RESMOD=	. specify resident TIP internal modules
shutDowN=	. background program for shutdown
SITE=	. site name
startUP=	. background program for startup
TeRMSize=	. terminal screen size rows,columns
TeRMType=	. general terminal type

Where:

Banner1='48 char' up to 48 character message which is displayed as part of the message sent to terminals when TIP begins execution.

Banner2='48 char' up to 48 character message which is displayed as part of the message sent to terminals when TIP begins execution.

CLoSe=(a,...,z) a,...,z are the LFD names of on-line files which are to be unavailable to the on-line network until an OPEN request is made for the file. Up to 10 files may be specified.

ESCape=c is the system escape character to be used.

Default is the commercial at sign (@).

IMS= "Y" if standard IMS/90 emulation is to be used.

May be specified as the name of a local IMS emulator.

LNEREQ=(line...names) up to 30 line names which are generated in ICAM. If this parameter is not specified, TIP/30 will instruct ICAM to open the entire network. However, if any line cannot be opened, ICAM will not open any lines.

By specifying the line names explicitly, TIP will instruct ICAM to open the named lines individually. If an error occurs on one line it will not affect the opening of others.

(This is quite useful for System/80's with local work stations.)

Network=(m,n) 'm' is the CCA name of the ICAM which TIP will use and 'n' is the CCA PASSWORD.

RESident=(a,b,c...) a,b,c are the names of transaction programs that are to be made permanently resident when TIP is initialized.

These programs will be loaded once from the TIPLOD library and may not be reloaded while TIP is executing.

A maximum of 90 programs may be made resident.

RESMOD=TB$SCT asks for a fast I/O routine to be loaded for use with the swap file (TIP$SWAP). This can only be used for sector style disks such as 8416, 8417, 8418, and 8419.

If this option is selected, the SWAP file (TIP$SWAP) must be one (1) contiguous extent with no secondary allocation.

For optimal performance, alternate track assignments within the swap file should be avoided.

The TB$SCT routine has been found to reduce the swap time of a 40K program from 125 ms to 70 ms (a 40% reduction).

shutDowN=trid The catalogued name of a background program which is automatically started before TIP goes to end of job.

startUP=trid	The catalogued name of a background program which is to be automatically started when TIP is initialized.
SITE=siteid	Site name (up to 12 characters. Enclosed in single quotes if imbedded blanks are present.
TeRMSize=(m,n)	'm' is the number of rows and 'n' is the number of columns of TERMTYPE. Eg: the screen size.
TeRMType=trmtype	trmtype is most common type of terminal in the on-line network. See CLUSTER statement in TIPGEN section for valid entries.

8.5 FILE RECOVERY TB$RCV

This program is capable of undoing certain updates made to files (roll backward) and re-doing updates (roll forward).

To roll a file forward it should be restored from the appropriate backup and then the TIP recovery may be run.

TB$RCV gets commands from control stream card images.

The commands are entered in columns 1 to 70 of the card.

All commands must begin with the word ROLL or QUICK and end with a semi-colon (";").

The first card in the control stream must be a // PARAM card identifying the TCA used when TIP was executed.

Syntax:

```
// PARAM tca
/$
ROLL func lfd  [FOR clause]  [FROM clause] [TO clause];
QUICK;
/*
```

Where:

tca The name of the TIP CONTROL AREA which was in use when TIP was running.

function The direction to roll. Choose one of "FORWARD" or "BACKWARD".

lfd The 'LFD' name of the file this command refers to, or '*ALL' if it refers to all files.

FOR clause the word 'FOR' is required if this clause is used.

USER=name, only process updates done by this user.

TRAN=prog, only process updates done by the identified TIP program.

TERM=term, only process updates done at the identified terminal.

FROM clause the word 'FROM' is required if this clause is used.

FROM YY/MM/DD HH:MN

only process updates done after this date and time.

TO clause the word 'TO' is required

TO YY/MM/DD HH:MN

only process updates done before this date and time.

As many commands as required may be entered. TB$RCV makes 2 passes on the data to roll a file backward.

A quick recovery will do a scan of the journal file (or before image file) and create recovery requests for all files which TIP had active and was logging at the time of a system crash or power failure.

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. A quick recovery could be performed every time TIP is executed. This is possible because no file updates will have been lost if TIP terminated normally and closed all files. Quick recovery should only be used after a crash where TIP could not terminate normally. If specified, this option must be stated first in the control stream, but it may be followed by other 'ROLL' requests.

The following is an example of a recovery runstream.

```
// JOB AR$RCV,,F000,1F000
// TIPFILES
// DVC 76 // VOL SMP333 // LBL ARC$ISAM1 // LFD   ISAM1,,ACCEPT
// DVC 76 // VOL SMP333 // LBL ARC$IRAM1 // LFD   IRAM1,,ACCEPT
// DVC 76 // VOL SMP333 // LBL TIP$JRN   // LFD TIP$JRN,,ACCEPT
// WORK1
// EXEC  TB$RCV,TIP
// PARAM TIPTST
/$
        ROLL      FORWARD ISAM1;
        ROLL BACKWARD IRAM1 FOR USER=RJNORMAN
                            FROM 79/08/23 18:00
                              TO 79/08/23 24:00;
/*
/&
```

If TIP/30 was creating a tape log then the recovery job control should assign the tape using an LFD name of TIP$LOG.

When doing a QUICK recovery from the before image file, the user may assign the before image file (in place of the journal file) with an LFD name of TIP$B4.

8.6 TIP/30 BATCH PROGRAMS TIP: batch jobs

This section of the manual documents the available batch job streams that are supplied to support the on-line TIP/30 system.

There are job streams illustrated to:

- list the TIP/30 catalogue
- list the TIP/30 journal file
- run the batch document generator
- compile and link a COBOL 74 TIP program

The illustrated job streams are representative of the JCL that is required. Minor modification may be necessary (and may have been made) to suit individual site requirements.

8.7 TIP FILE INITIALIZATION TB$INT

The user is supplied a batch program (Load Module: TB$INT) which is capable of initializing TIP/30 required workfiles and creating catalogue records for USER-IDs, FILES and PROGRAMS from free format style control cards.

There are four control statements used by TB$INT.

COPY	collect input from file/elt
USER	define a USER-ID
PROG	catalogue a program
FILE	catalogue a file

TB$INT must be used to format the work files used by TIP. The program will scan the Job Control Statements used at execution time and only format those files which are assigned.

If the file TIP$CAT (catalogue) is assigned for formatting, TB$INT will ask for confirmation to INIT the catalogue. Reply with one of the following:

YES

- continue file catalogue initialization.

NO

- do not INIT the catalogue, TB$INT will format any remaining files specified.

CAT

- INIT the catalogue part only (ie. leave all message formats in the message partition of TIP$CAT).

If the catalogue is to be initialized you will be asked for a password. Enter a password of "TIP/30" at this time.

The file names used must be from the following set:

TIP$RNDM Random file. Formatted as a SAT file with 512 byte block size.

TIP$SWAP 2 partition SAT file. Formatted for use as swapping storage.

TIP$JRN Journal file. Formatted as a SAT file with 512 byte block size.

The user may use the DATA utility to back up this file to another tape or disk, and then re-format it.

Each time TIP is run the file is extended The only way to INIT it is to re-format.

TIP$B4 Before image file. Formatted as a SAT file with 512 byte block size. There is no need to back up this file since it is re-used each time TIP starts up.

It must initialized by TB$INT before executing TIP.

TIP$CAT TIP/30 Catalogue.

2 partition SAT file, partition 1 holds the catalogue, partition 2 holds the message formats.

You will get a message asking if you really want to INIT the catalogue.

If so, reply "Y" and then "TIP/30" to the password request prompt.

OLDCAT An existing TIP catalogue file which is to be moved to TIP$CAT.

The catalogue file is a hashed file and access to the file slows as the file grows.

To re-organize the catalogue: allocate a larger disk file and use the TB$INT program to copy OLDCAT to TIP$CAT.

TIP$RNDM and TIP$JRN may be copied with DATA utility for backup.

TIP$CAT may be copied by TB$INT (with OLDCAT/TIP$CAT combination) to backup the catalogue.

TIP$SWAP contains no information of interest when TIP is down; there is no need to back it up.

Although TIP$CAT and TIP$SWAP are SAT files they are definitely NOT library files. Do NOT try to copy them or look at them using QED, TLIB, or LIBS.

Unless specifically noted herein, TIP files should only be moved about with the DMPRST utility (recommendation is to use the "FILE" mode too).

The following sections describe the catalogue control statements and valid operands, keywords and options.

8.7.1 COPY IN STATEMENTS TB$INT: copy

The COPY statement is used to read statements that are stored in a library element.

Syntax:

COPY FILE/ELT

Where:

file The library file name (LFD) containing the element.

elt The name of the element to copy at this point.

Additional Considerations:

A module called TIP/TC$CATLG is supplied. It contains statements to catalogue all supplied TIP utility programs and standard files.

-+*+-

8.7.2 USER, PROGRAM, FILE COMMANDS TB$INT: cat

The format of the catalogue commands for TB$INT (ie: USER, PROGRAM, and FILE) is identical to the corresponding commands used by online catalogue manager.

Refer to the documentation of the on-line Catalogue Manager Program (CAT) for the syntax and requirements of these commands.

-+*+-

8.7.3 CATALOGUE INITIALIZATION SAMPLE TB$INT: sample

```
// JOB ACEINT,,A000,C000
// DVC 20 // LFD PRNTR
// DVC 50 // VOL ACE000 // LBL $LOK30.TIP$LIB // LFD TIP
// DVC 50 // VOL ACE000 // LBL TIP$CATFILE    // LFD TIP$CAT
// DVC 50 // VOL ACE000 // LBL TIP$SWAP       // LFD TIP$SWAP
// DVC 50 // VOL ACE000 // LBL TIP$RNDM       // LFD TIP$RNDM
// OPTION JOBDUMP
// EXEC   TB$INT,TIP
/$
*
* USER-IDS FOLLOW
*
USER MASTER           TYPE=MASTER  PWD=ACE.
COPY TIP/TC$CATLG
USER MARYJANE         PWD=A TYPE=APPL GRP=COMP.
*
* STANDARD ONLINE PROGRAM NAMES
*
PROG DBGGD,TT$GD      DEBUG=IDA.
PROG ED,TT$QED        MCS=1200,WORK=4096,USAGE=RNT.
PROG QED,TT$QED       MCS=1200,WORK=6000,USAGE=RNT.
PROG TCP,TT$TCP       WORK=1536,CDA=64,USAGE=RNT.
PROG TCSED,TT$QED     MCS=1200,WORK=2048,USAGE=RNT,GRP=COMP.
*
* FILES USED BY SYSTEM
*
FILE TCSCNTRL,TCSCL   GRP=COMP
/*
/&
```

-+*+-

8.7.4 TIP FILE INITALIZATION JOBS TB$INT: jobs

The TIP/30 release library (TIP) contains a number of job streams that are used to initialize files during the installation of TIP.

These jobs are merely specific executions of the TB$INT program.

Following is a summary of the supplied jobs and their purpose.

TJ$B4 Allocate and format the (optional) TIP/30 Before Image file (TIP$B4).

Create JPROC "TIP$B4".

TJ$CAT Allocate and initialize the TIP/30 Catalogue File (TIP$CAT).

Create JPROC "TIP$CAT".

TJ$CATB Allocate and initialize the TIP/30 Catalogue Backup File (TIP$CATB).

Create JPROC "TIP$CATB".

TJ$COP Copy all the TIP load modules from the TIP release library (TIP) to the TIP load library (TIPLOD).

TJ$CRBAK Copy TIP$CAT to TIP$CATB, then dump TIP$CATB and TIP$RNDM to tape.

TJ$CRRST Restore TIP$CATB and TIP$RNDM from tape, then copy TIP$CATB to TIP$CAT.

TJ$HST Allocate and initialize the (optional) TIP/30 journal history file (TIP$HST).

Create JPROC "TIP$HST".

TJ$ICAM Allocate ICAM discfiles. (Eg: TCIDTF, DQUEUE1 etc).

Create JPROC "ICAMFILE".

TJ$JRN Allocate and initialize the (optional) TIP/30 journal file (TIP$JRN).

Create JPROC "TIP$JRN".

TIP FILE INITALIZATION JOBS

TJ$LOD Allocate the TIP/30 load library.

Create JPROC "TIPFILES".

TJ$RNDM Allocate and intialize the TIP/30 Random file (TIP$RNDM).

Create JPROC "TIP$RNDM"

TJ$SWAP Allocate and initialize the TIP/30 Swap file. (TIP$SWAP).

TJ$TQL Allocate the TIP Query Language dictionary

Create JPROC TIP$TQL

TJ$USER Create JPROC "USERFILE".

-+*+-

8.8 JOURNAL FILE COPY AND INITIALIZATION TB$JRN

Although TB$INT will initialize the TIP journal file, another batch program (TB$JRN) is provided for the specific purpose of journal file maintenance plus initialization.

This program will copy the execution journal file (TIP$JRN) to a journal history file (called TIP$HST) and upon completion of the copy, the execution journal file will be initialized.

It is recommended that the execution journal file be initialized before every execution of TIP.

The history file (TIP$HST) is extended by the TB$JRN program and may be backed up with DATA utilities.

If the history file is not allocated (ie: // LFD TIP$HST does not appear in the JCL) when TB$JRN is executed, the TB$JRN program will simply initialize the journal file.

The history journal file could be initialized by assigning it with an LFD of TIP$JRN and executing TB$JRN.

Since the history journal file and the execution journal file are the same format, the journal file list program (TB$LST) and the file recovery program (TB$RCV) can be executed from either file.

In the case of file recovery, a 'QUICK' recovery should NOT be done from the history file.

The following is a sample execution of the journal file copy program.

```
// JOB JRNINT,,6000
// TIPFILES
// DVC 50 // VOL ACE000 // LBL TIP$JRN      // LFD TIP$JRN
// DVC 51 // VOL ACE001 // LBL TIP$JRN$HISTORY // LFD TIP$HST
// OPTION JOBDUMP
// EXEC   TB$JRN,TIP
/&
```

8.9 COMPILE COBOL-68 TIP PROGRAM TJ$COB68

This supplied job stream compiles and links (into TIPLOD) a TIP/30 native mode program that is written in COBOL-68.

Global parameters are provided to allow the user to specify the source program library and element name.

```
//        JOB       TJ$COB68,,D000,10000
//        GBL       TF=TIPFILES,E,F=SYSGEN
//        IF        ('&E' NE '')EOK
//        JNOTE     'E PARAMETER MISSING, COMPILE IGNORED.'
//        JNOTE     'SPECIFY E=PROGRAM NAME'
//        OPTION    TEST
//        GO        END
//EOK     &TF
//        USERFILE
//        EXEC      WRTBIG
//        PARAM     'TIP COMPILE'
//        PARAM     'OF &E'
//        PARAM     'FROM &F'
//        PARAM     'DAT$'
//        PARAM     'TIP COMPILE'
//        PARAM     'OF &E'
//        PARAM     'FROM &F'
//        PARAM     'DAT$'
//        WORK1
//        WORK2
//        WORK3
//        EXEC      COBOL
//        PARAM     IN=&E/&F
//        PARAM     LIN=TIP
//        PARAM     OUT=(M)
//        PARAM     LST=(C,D,E,L,M,P,S,X)
//        SKIP      ERROR,11
//        SKIP      OK
//ERROR   OPR       'ERRORS COMPILING &E, LINK BYPASSED.'
//        SKIP      END
//OK      OPTION    SUB
//        WORK1
//        EXEC      LNKEDT
//        PARAM     ZRO,ALIB=TIP,OUT=$Y$RUN
//        PARAM     CMT='&E - ONLINE TIP PROGRAM'
/$
          LOADM     &E
/*
//        SKIP      ERROR,11
//        SKIP      OK
//ERROR   OPR       'ERRORS LINKING &E, MODULE NOT REPLACED.'
//        SKIP      END
```

```
//OK      OPTION    SUB
//        EXEC      LIBS
/$
        FIL       D1=$Y$RUN,D2=TIPLOD
        COP       D1,L,&E,D2
/*
//END    NOP
/&
//        FIN
```

8.10 COMPILE COBOL-74 TIP PROGRAM TJ$COB74

This supplied job stream compiles and links (into TIPLOD) a TIP/30 native mode program that is written in COBOL-74.

Global parameters are provided to allow the user to specify the source program library and element name.

```
//        JOB       TJ$COB74,,D000,10000
//        GBL       TF=TIPFILES,E,F=SYSGEN
//        IF        ('&E' NE '')EOK
//        JNOTE     'E PARAMETER MISSING, COMPILE IGNORED.'
//        JNOTE     'SPECIFY E=PROGRAM NAME'
//        OPTION    TEST
//        GO        END
//EOK     &TF
//        USERFILE
//        EXEC      WRTBIG
//        PARAM     'TIP COMPILE'
//        PARAM     'OF &E'
//        PARAM     'FROM &F'
//        PARAM     'DAT$'
//        PARAM     'TIP COMPILE'
//        PARAM     'OF &E'
//        PARAM     'FROM &F'
//        PARAM     'DAT$'
//        WORK1
//        WORK2
//        WORK3
//        EXEC      COBL74
//        PARAM     IN=&E/&F
//        PARAM     LIN=&F
//        PARAM     IMSCOD=YES,PROVER=YES
//        PARAM     MXREF=YES,LSTREF=NO
//        PARAM     AXREF=YES,AXNON=YES
//        SKIP      ERROR,11
//        SKIP      OK
//ERROR   OPR       'ERRORS COMPILING &E, LINK BYPASSED.'
//        SKIP      END
//OK      OPTION    SUB
//        WORK1
//        EXEC      LNKEDT
//        PARAM     ZRO,ALIB=TIP,OUT=$Y$RUN
//        PARAM     CMT='&E - ONLINE TIP PROGRAM'
/$
          LOADM     &E
          INCLUDE   &E,$Y$RUN
C@@MSI    EQU       1
/*
//        SKIP      ERROR,11
```

```
//        SKIP       OK
//ERROR   OPR        'ERRORS LINKING &E, MODULE NOT REPLACED.'
//        SKIP       END
//OK      OPTION     SUB
//        EXEC       LIBS
/$
          FIL        D1=$Y$RUN,D2=TIPLOD
          COP        D1,L,&E,D2
/*
//END     NOP
/&
//        FIN
```

8.11 THE BATCH DOCUMENT GENERATOR TJ$DOCS

Input to the Document Generator may be either in 80 column card-images or in source elements in a disc library file (in which case only cols 1 - 72 are used).

If the input is a source module, a // PARAM statement must contain the name of the source module and the LFD name of the file containing the input source module.

If the input is in a card deck, no PARAM statement is necessary. The usual start-of-data ("/$") and end-of-data ("/*") statements are required.

The minimum memory specification (JOB card parameter three) must be specified when executing the Document Generator. The minimum storage required for the Document Generator is X'11000' (68K).

```
//       JOB       TJ$DOCS,,11000
//       GBL       C=UPPER,F=SYSGEN,E=DOCINP
//       TIPFILES
//       USERFILE
//       WORK1
//       TEMP1
//       OPR       'GENERATION OF THE DOCUMENT &E FROM &F'
//       EXEC      TB$DOC,TIP
//       PARAM     CASE=&C
//       PARAM     IN=&F/&E
/&
//       FIN
```

8.11.1 TJ$DOCS PARAM CARD FORMAT TJ$DOCS: param

The param card formats are as follows:

// PARAM IN=file/element

- This parameter defines the source element that is to be used as input to the Document Generator. This element must be contained in a standard OS/3 library file.

// PARAM VER=xxx

- This parameter specifies the version (xxx) of the input element. This version number may be referenced by calling macro @33. Default is "2.5".

// PARAM REV=nn

- This parameter specifies the revision number of the input element. This version number may be referenced by calling macro @39. Default is zero or the version number of the element if the element has been edited by the TIP/30 text editor (QED).

// PARAM CASE=UPPER

- This parameter specifies that all alphabetics in the output document are to be forced to upper case regardless of the actual case of the input data.

// PARAM FLAG=c

- This defines the margin flagging character. Default is the vertical bar character ("|").

// PARAM TAPE=YES

- This will cause the program to create an output tape in addition to the normal printed output. TAPE LFD name must be "DOCTAPE". Tape will be variable length blocked 7680.

-+*+-

8.12 CATALOGUE FILE LISTING TJ$LC

This batch job will list the TIP/30 catalogue file. Parameters are available to also generate a cross reference listing of load module names, directory of MCS screen formats (which are stored in a separate partition of the catalogue file), and user passwords.

```
//        JOB       TJ$LC,,B000,D000
//        GBL       TF=TIPFILES,MCS=YES,XREF=YES,USER
//        &TF
//        EXEC      WRTBIG
//        PARAM     ' LISTING OF '
//        PARAM     '   TIP/30   '
//        PARAM     '  CATALOG   '
//        PARAM     'DAT$'
//        PARAM     ' LISTING OF '
//        PARAM     '   TIP/30   '
//        PARAM     '  CATALOG   '
//        PARAM     'DAT$'
//        &$LBL     TIP$CAT
//DM01    WORK1     BLK=1280
//DM02    WORK2     BLK=1280
//        OPTION    SCAN,SUB
//        EXEC      TB$LC,TIPLOD
/$
//        IF        ('&USER' EQ '')NOUSER
          USER=&USER
//NOUSER  NOP
          MCS=&MCS,XREF=&XREF..
/*
/&
//        FIN
```

8.12.1 CATALOGUE LIST PROGRAM PARAMETERS TJ$LC: params

The batch catalogue list program expects a card image input stream containing free form keyword parameters as described below. The keywords may appear in any order. The last keyword should be terminated with a period.

USER= Identifies a user-id and password of a valid user of the TIP/30 system. If the entire catalogue is to be processed, then the user-id specified must be of security level 1 (TECH). If a user-id with a security level numerically greater than 1 is given, then only those records to which the specified user-id has access are processed. This keyword must be given.

User passwords are displayed ONLY if a level 1 userid and password is given.

GRP=(g1,...,g8) Identifies a list of up to eight group names that are to be processed. These name may be prefix names (ie. *PAY = all groups starting with PAY) Default is all groups to which the given user has access, or, in the case of a level 1 user, all groups.

XREF=NO Indicates the suppression of cross reference listings. XREF=YES is the default.

MCS=NO Indicates that a directory of MCS screen formats is not desired. MCS=YES is the default.

BLK=YES Indicates that the physical block numbers for the catalogue records are to be printed. BLK=NO is the default.

. (period) Required delimiter after the last specified keyword.

If the keyword parameters are not specified in the JCL, then the program will prompt the user at the OS/3 operator console for keyword entries until the user terminates a keyword with a period (".").

-+*+-

8.13 LIST JOURNAL FILE TJ$LST

The TJ$LST job will read the TIP/30 journal file (tape or disk) and will sort the information and produce a summary report. Parameter cards are used to specify the type of summary desired.

```
//        JOB       TJ$LST,,D000,10000
//        GBL       TF=TIPFILES,LIST=ALL,F=TIP$JRN
//        &TF
//        &$LBL     &F
//        OPR       'LISTING TIP/30 JOURNAL FILE &F'
//DM01    WORK1
//DM02    WORK2
//        EXEC      TB$LST,TIP
//        PARAM     LIST=&LIST
/&
//        FIN
```

Syntax:

// PARAM option

Where:

"option"	may be chosen from the following:
LIST=SUMMARY	Produce summary level total information only.
LIST=ALL	list all journal records and summarize. The program will list the first 50 bytes of your data record in character format. If you wish to see more of the data record you must write your own version of the TB$LST program.
ACCT=name	name is a logon account to be selected
FILE=name	name is a file name to be selected
PROG=name	name is a transaction code to be selected
TERM=name	name is a terminal name to be selected
TYPE=name	name is a record type to select (Eg: LGOF)
USER=name	name is a USER-ID to be selected

Additional Considerations:

A maximum of 20 of each of ACCT, FILE, PROG, TERM, TYPE, and USER specifications may be present. Only one LIST option is meaningful.

Specification of LIST=SUMMARY causes the list program to ignore all other qualifiers (eg: USER= FILE= etc).

Refer to the section of the manual describing access to the TIP/30 Journal file (FCS) for the layout and content of journal file records.

8.14 OS/3 CONSOLE OPERATION opr commands

The operator should never change the system date or time while TIP/30 is running. Changing the date (or time) could result in critical journal information being written incorrectly. Furthermore, user programs may be dependent on the date and time for scheduling activity etc.

TIP/30 is critically dependent on OS/3 timer services. If the time of day is changed while TIP/30 is running TIP may go into a continuous wait state for several hours!!

DO NOT CHANGE THE TIME OF DAY WHILE TIP IS RUNNING !!

If TIP is (inadvertently) executed when the TIME or DATE is not correct, it is recommended that you shutdown TIP/30 gracefully as quickly as reasonable, correct the date and time and restart TIP/30.

The following commands may be presented to TIP/30 as unsolicited operator keyins. As unsolicited commands, the TIP/30 job slot number must precede the command (ie: 10 WHOSON).

APB ...text... Send the specified text to all users who are logged on.

APB/ALL ...text... send the message to all terminals in the network.

CLOSE lfd Flags the file called 'lfd' unavailable for use by TIP/30 user programs and physically closes it. 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 an OPEN is issued. Any program trying to access such a file is returned an error (LOCKED). Programs currently using the file are allowed to continue until they DE-ACCESS the file. When no 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).

CRASH Cancels TIP and gives a memory dump.

DATE Will display the current date and time in English.

Eg: WEDNESDAY AUGUST 18 1982

DIE/user [/term] causes the program running under USER on TERM to be cancelled with a memory dump. It may be necessary to press MSG-WAIT (or some such input) on the user's terminal to cause TIP to recognize this action.

DOWN line will set down the named line. If this is a work station then the terminal is available to interactive services again.

DOWN term will set down the line of the named terminal. If this is a work station then the terminal is available to interactive services again.

EOJ Sets a TIP/30 flag which prevents users from executing other programs once their current program terminates; then when all users have finished initiates an orderly and normal end of job.

FILES Produce an I/O summary report of active OS/3 files assigned to TIP/30.

FLAGS Display the current status of the 32 TIP flags.

LMOFF/term Turn off software line monitor.

To obtain the line monitor printout, first breakpoint the print queue for the active TIP30 job (BR ACT,PRT,JOB=TIP30) then start a burst mode output writer (PR BX,JOB=TIP30).

LMON/term Turn on the software line monitor. A display of all input and output messages (plus delivery notification from ICAM) for the specified terminal will be dumped to the site printer.

MSG/term ...text...	the message is sent to a specfic terminal.
MSG/user ...text...	the message is sent to a specific
OFF,Fl,...Fn	Cause the named flags to be placed in THE 'OFF' STATE. The values of Fl through Fn may be 0 to 31.
ON,Fl,...Fn	Cause the named flags to be placed in THE 'ON' STATE. The values of Fl through Fn may be 0 to 31.
OPEN lfd	Make the file lfd (as it was defined in the TIP/30 generation) available for use by TIP/30 programs. This is only to be done if it was previously CLOSED. Several lfd names may be specified (separated by commas). This command will cause TIP/30 to issue a real Data Management OPEN for the files.
QCLEAR term,q-id	forces ICAM to flush out it's queues after an output delivery notification loss to a terminal. Only one 'q-id' value may be given at a time and must be 'H' for high, 'M' for medium or 'L' for low.
STATS	Produce a statistical report of the TIP/30 system at the operator's console.
STOP	Forces TIP into an immediate, but normal, end of job.
TERMS	List all terminals in the ICAM network, their status (up/down) and USER-ID of current User.
UP line	will mark up the named line. If a 'ready messge' is to be sent then it will be sent to the first terminal on the line. (This is useful for System/80 workstations).
UP term	will mark up the line of the named terminal. If a 'ready messge' is to be sent then it will be sent to this terminal. (This is useful for System/80 workstations.)

WHOSON Gives a list of the USER-IDs and the associated terminal names.

8.15 CONSOLE MESSAGES messages

Following is a list of TIP/30 console messages that may occur and any corresponding actions to be taken.

In the examples of message text, underscores represent data in the message that will be supplied by TIP/30.

Note that messages emanating from TIP/30 are prefixed by the string "TI#nnn" where "nnn" represents the internal message number.

TI#01 TIP/30 INITIALIZATION ALLINSON-ROSS CORPORATION

- Informational: TIP/30 has been loaded and the initialization has begun.

TI#02 __/__/__ - __:__:__ TIP/30 READY FOR _____

- Informational: TIP/30 initialization completed at the date and time specified. The SITE-ID is also indicated.

TI#03 UNABLE TO LOAD "TB$JCS"

- Tables required for processing run control statements which are supplied as parameters in the TIP/30 JCL are missing.

TI#04 UNKNOWN OPERATOR REQUEST - CONSULT MANUAL

- The operator has entered an unknown command to TIP/30

TI#05 ERROR OCCURED ATTACHING "SCHEDULING"

- An error occurred when the main TIP/30 task tried to 'ATTACH' the subtask that performs all program scheduling. This error usually indicates that 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.

TI#06 ERROR OCCURED ATTACHING "COMMUNICATIONS"

- 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.

TI#07 ERROR OPENING ICAM (____/_____), CODE=_____

- An error occurred when TIP/30 tried to open the communications network (VIA ICAM MOPEN MACRO). This error usually indicates that either the CCA name or password are incorrect. This error may also occur if the disk queue files are not contiguously allocated, or if the ICAM network has more terminals defined than was generated into TIP/30 (see TERMS= parameter of TIPGEN macro). TIP/30 will terminate when this error occurs.

- The error codes are listed at the back of the OS/3 System Messages handbook (Table A-1, Category 'AA').

TI#08 UNABLE TO LOAD CONTROL MODULE _____

- TIP is unable to load the users TCA module which is produced by the TIPGEN procedure. TIP/30 will terminate.

TI#09 TIP/30 CATALOGUE IS NOT INITIALIZED

- An error occurred when TIP/30 tried read the root record of the catalogue. This error usually indicates that the user has not run the job TB$INT which processes parameters and creates Catalogue file entries for USER-IDs, online files and programs. TIP/30 will terminate when this error occurs.

TI#10 TIP/30 TERMINATED

- This message appears when TIP has encounters an unrecoverable error.

TI#11 _____ PAGES OF MEMORY FOR PROGRAMS

- Indicates the size of the user program region (working storage) of TIP/30. This size is specified as a number of 2K pages of memory, exclusive of resident programs. The size of this area determines how many user programs may reside in memory concurrently.

TI#12 INSUFFICIENT MEMORY TO EXECUTE TIP/30

- The user program region (working storage) of TIP/30 is too small. The minimum amount of storage required is that specified by the 'MAXPROG= ' keyword of the TIPGEN. 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.

TI#13 FATAL INITIALIZATION ERROR -- TIP/30 ABORTED

- A previous error has occurred and the execution of TIP/30 cannot continue.

TI#14 INVALID OPTION SELECTED AT _____

- An invalid option was selected in the run control statements provided in the TIP JCL.

TI#15 _____ IS NOW AVAILABLE FOR ONLINE USE

- Confirmation that a closed file has been successfully re-opened at the operator's request.

TI#16 _____ HAS _____ USERS. "CLOSE HELD PENDING"

- Warning that requested close of a file is being delayed by activity

TI#17 _____ IS CLOSED & NOT AVAILABLE FOR ONLINE USE

- Confirmation that online file has been closed for batch use.

TI#18 FILE _____ DOES NOT EXIST

- Warning that requested file was not found (possible spelling error)

TI#19 UNABLE TO LOAD PMDA : _____ _____

- There is insufficient free memory 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.

TI#20 SWAPPING STORAGE 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 = TIP$SWAP).

TI#21 UNABLE TO LOAD RESIDENT PROGRAM: _____

- The program named, which is specified on a RESIDENT job control statement, cannot be loaded.

TI#22 ____% OF CATALOGUE FILE USED

- Informational: Percentage of catalogue file (TIP$CAT) used.

TI#23 ____% OF MESSAGE FILE USED

- Informational: Percentage of Message file (TIP$CAT) used.

TI#24 _____ BLOCKS OF DYNAMIC FILE USED

- Informational: Number of blocks of the Dynamic file (TIP$RNDM) used.

TI#25 SWAPPING STORAGE EXHAUSTED

- The TIP Swap file (TIP$SWAP) has become full. The user must scratch and reallocate the file to increase its size. When this file is reallocated, it must be reformatted using the TB$INT program.

TI#26 SWAPPING STORAGE FILE - I/O ERROR - DM__

- An unrecoverable hardware I/O error occurred on the TIP$SWAP file. As a temporary solution, try moving the file to another location or disk drive. The Univac customer engineer should by made aware of any persistent problem such as this. TIP/30 will terminate when this error occurs.

TI#27 MEMORY MANAGEMENT ERROR - JOB CANCELLED

- An error as occurred which has corrupted TIP memory management. Possibly a rogue user program has destroyed part of the TIP region. TIP/30 will terminate when this error occurs.

TI#28 PROGRAM EXCEPTION; PSW= _____ __ __ _____

- An error 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 Allinson-Ross Corporation with as much supporting information as possible. TIP/30 will terminate when this error occurs.

TI#29 TIP/30 INTERNAL SOFTWARE FAILURE

- TIP has detected an unrecoverable error (internal tables have been modified in error). If the error cannot be traced to rogue user programs, then the memory dump produced by this condition should be forwarded to Allinson-Ross Corporation with as much supportive information as possible. TIP/30 will terminate when this error occurs.

TI#30 UNABLE TO ATTACH USER TASK

- TIP was unable to ATTACH a new User. This error usually results from an insufficient number on TCBs being specified on the JOB card.

TI#31 ICAM NOTE ON ____ - _____

- TIP 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.

TI#32 ICAM ERROR _____, FROM ____ = _____

- An ICAM error occurred when TIP issued a request to the specified terminal.

TI#33 TIP/30 SYSTEM UNSTABLE - ADVISE CANCEL

- TIP has determined that internal tables or code has been altered erroneously and suggests that TIP/30 should be cancelled.

```
TI#34 **** TIP/30 VERSION _____ - STATISTICS ****
TI#35 _____ AT __:__:__
TI#36 MSG IN= _____ TOTAL LEN= _____ AVG= _____
TI#37 MSG OUT= _____ TOTAL LEN= _____ AVG= _____
TI#38 NUMBER OF SWAPPING STORAGE FILE I/OS = _____
TI#39 _____ SLOW PROGRAM LOADS FOR _____ REQUESTS
TI#40 AVERAGE RESPONSE TIME IS _____ SECONDS
TI#41 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 in the TIP/30 GEN).

TI#42 _____ IS NOT LOGGED ON

- An invalid USER-ID has been specified as a parameter in an unsolicited command to TIP/30.

TI#43 ____ IS AN INVALID TERMINAL NAME

- an invalid TERM-ID has been specified an a parameter in an unsolicited command to TIP/30.

```
TI#44 USER-ID TERMINAL SEC PROGRAM LEVEL
TI#45 _____ _____ ___ _____ __
```

- These heading lines are displayed in response to the WHOSON console operator command.

FLAG - STATE FLAG - STATE FLAG - STATE FLAG - STATE
__ - ___ __ - ___ __ - ___ __ - ___

- These heading lines are displayed in response to the FLAGS console operator command.

TI#48 FILE #I/O'S OUTPUT #USERS OPEN
TI#49 _____ _____ _____ _____ _

- These heading lines are displayed in response to the FILES operator command. Indicated is the file name, the total number of I/O requests issued to this file; the number of I/O requests which were outputs; the number of current users of the file and whether the file is currently open.

TI#50 YOU NEED AT LEAST 4 TASK CONTROL BLOCKS
TI#51 NO MORE THAN 12 TASKS ARE REQUIRED

- These headings are displayed when TIP/30 determines that an insufficient number of task control blocks have been specified on the TIP/30 JOB card.

TI#52 NO MEMORY TO LOGON ____

- The memory manager in TIP/30 is unable to acquire enough free memory to logon the user at the named terminal.

TI#53 UNABLE TO LOAD _____ FOR ____

- TIP/30 was unable to load the specified program. The program does exist in the TIPLOD file but an I/O error occurred during the load. The program should be re-linked and placed back into the TIPLOD file. (The TIPLOD library may be compromised!)

TI#54 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.

TI#55 _____(____):_____

- This is a message to the operator from the user and terminal indicated.

TI#56 <<<KEY HOLDING TABLE IS FULL>>>

- This message is displayed whenever TIP/30 detects that the key holding table is FULL. Any requests to hold a record when the table is full, will receive a "record held" status. This should be brought to the attention of the systems programmer.

TI#57 * NOT USED ***

- This message is not used in this version of TIP/30.

TI#58 UNAUTHORIZED USER ATTEMPTED LOGON AT ____

- The operator is being informed that an unsuccessful attempt to logon has been detected at the indicated terminal.

TI#59 DLL: LOAD OF ____, SIZE = _____ BYTES

- The named UTS-400 terminal has been down line loaded with a module of the size indicated.

TI#60 DLL: STATUS FROM ____ = "_____"

- The status of the down line load to the named terminal is given.

TI#61 _____ DISK I/O'S FOR _____ MCS FORMAT REQUESTS

- Informational: The number of disk I/Os to the MCS Display library which were required for the number of request for MCS formats. This ratio may give the user some insight into whether MSGPOOLing could be adjusted to gain more advantage.

TI#62 _____ CATALOGUE FILE I/O'S

- Informational: The number of I/Os to the TIP/30 Catalogue File (TIP$CAT).

TI#63 _____ DYNAMIC FILE I/O'S

- Informational: The number of I/Os to the TIP/30 Random File (TIP$RNDM).

TI#64 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 TCB's is 4 and maximum is 12).

TI#65 TIP$RNDM ERROR DM__ FOR _____/_____/_____

- An I/O error has occurred on this file, check OS/3 System Message handbook for description of the error.

TI#66 _____(____):_____ ABENDED; PMDA SCHEDULED

- Reports the USER-ID(terminal):program name, which has aborted. The TIP/30 Post Mortem Dump Analysis program has been automatically loaded for the user.

TI#67 BCP: ____ LOGGED ON _____

- This message reports that a BCP user has logged on TIP/30 at the indicated terminal.

TI#68 BCP: ____ LOGGED OFF _____

- This message reports that a BCP user has logged off TIP/30 at the indicated terminal.

TI#69 BCP: ____ ERROR _ ON _____

- Informational: Console note of BCP status.

TI#70 BCP: ____ TRANSMITTING FILE _____

- This message advises the operator that the indicated BCP user is transmitting the specified file from the terminal to the host.

TI#71 BCP: ____ RECEIVING FILE _____

- This message advises the operator that the BCP user indicated is receiving the specified file from the host.

TI#72 BCP: ____ _____ RECORDS TRANSFERRED

- This message advises the operator of the count of records transferred via BCP.

TI#73 BCP: ____ BATCH JOB SCHEDULED _____

- The BCP user indicated has submitted a batch job for scheduling.

TI#74 BCP: ____ RESERVED

- This message is reserved for future use.

TI#75 BCP: ____ RESERVED

- This message is reserved for future use.

TI#76 BCP: ____ RESERVED

- This message is reserved for future use.

TI#77 BCP: ____ RESERVED

- This message is reserved for future use.

TI#78 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.

TI#79 _____ BLOCKS OF JOURNAL FILE USED

- Informational: The number of blocks of the journal file which have been used to date. This file should be initialized periodically with TB$INT.

TI#80 _____ FILE NOT ASSIGNED

- The file is not assigned in TIP job control for a TIP gen'd function. Execution will continue without that function.

TI#81 JOURNAL FILE ERROR DM__

- I/O error on journal file. Consult OS/3 error messages book for description of error codes of the class 'DMnn'.

TI#82 SIGNED OFF TOTAL/7 FOR _____, STATUS=____

- TIP has done an automatic SINOF to TOTAL/7.

TI#83 LIBRARY ERROR ON _____: "_____"

- TIP/30 has detected an error of the indicated type on a library.

TI#84 ABNORMAL TERMINATION; ERROR CODE:___

- TIP has been cancelled either internally or by the operator; a DUMP will be produced.

TI#85 LOST DELIVERY NOTICE FOR ____

- A message, destined for an auxiliary device of the named TERM-ID, has been reported lost by ICAM.

TI#86 _____(____)SYM:_____

- The USER-ID(TERM-ID) has scheduled an OS/3 symbiont via the SYM program. This fact is reported to the operator and will also be placed in the console log.

TI#87 _____(____): DATA BASE NOT ACCESSED

- IMS type access not allowed in DMCL.

TI#88 _____(____): DATA BASE TERMINATED

- DBMS has died and the user noted has been cancelled.

TI#89 _____(____): DATA BASE RNT LIMIT EXCEEDED

- This occurs when a user program's bind has more than 30 parameters and is catalogued as re-entrant.

TI#90 _____(____): DATA BASE IMPART UNSUCCESSFUL

- This occurs because DMS/90 is not loaded or active.

TI#91 _____(____): DATA BASE NOT BOUND

- A DML verb has been issued while imparted but not bound.

TI#92 _____(____): DATA BASE INVALID REQUEST

- The DML request is not defined under DMS/90.

TI#93 _____(____): DATA BASE BIND UNSUCCESSFUL

- DMS/90 has returned an error during the bind function.

TI#94 _____(____): DATA BASE DEPART REQUIRED

- This warning indicates that an impart for a new user has occurred while a previous user was still imparted.

TI#95 _____(____): DATA BASE

- This message is reserved for future use.

TI#96 _____(____): DATA BASE

- This message is reserved for future use.

TI#97 _____(____): DATA BASE

- This message is reserved for future use.

TI#98 TIP/30 HAS EXPIRED; CONTACT A.R.C.

- This message is reserved for future use.

TI#99 _____(____): _____ ROLLED BACK _____

- The USER-ID(TERM-ID) using program named has caused a record roll back on the file named.

TI#100 GEN PARAMETERS VALIDATED - START _____? (Y/N)

- The TIP/30 parameterization procedure has successfully completed and is asking whether the named generation job should be scheduled.

TI#101 TIP/30 FILE FORMATTER VERSION _____

- This is a heading note indicating that program TB$INT is beginning execution.

TI#102 _____ FORMATTED, BLOCKS = _____

- The named file has been formatted to the specified capacity.

TI#103 OLDCAT SUCCESSFULLY COPIED TO TIP$CAT

- A previous TIP/30 catalogue has been successfully copied to the current catalogue 'TIP$CAT'.

TI#104 INITIALIZE THE TIP/30 CATALOGUE? (Y/N/CAT)

- Program TB$INT is asking for confirmation to initialize the TIP catalogue (Do you really want to do this ???). Answering 'N' will prevent it; 'Y' will init it completely including the MCS screen partition; 'CAT' will

allow continuation but protects the screen partition.

TI#105 ENTER PASSWORD TO INITIALIZE CATALOGUE

- Program TB$INT is requesting a password authorization before initialization of the catalogue. Be absolutely sure you know what you are doing, or be prepared to update your resume!

TI#106 TIP$CAT WILL HOLD _____ CATALOGUE RECORDS

- The TIP catalogue file has been initialized and has a remaining CAT capacity in partition one as specified.

TI#107 TIP$CAT WILL HOLD _____ MESSAGE FORMATS

- The TIP catalogue file has been initialized and has a remaining MCS capacity in partition two as specified.

TI#108 INVALID USER-ID/PASSWORD - CATALOGUE NOT PROCESSED

- The operator has not specified the correct password for TIP$CAT initialization and the function will not to done.

TI#109 ENTER CATALOGUE LIST OPTIONS?

- The batch catalogue listing program has found that there is insufficient data in the job control and is prompting the user at the OS/3 console for display options. Refer to batch job documentation for replies. A response ending in a period (".") will discontinue prompting.

TI#110 I/O ERROR ON "WORK1", DM__

- The system scratch file has had an occurence of a Data Management error as specified. See the OS/3 System Messages handbook for the explanation.

TI#111 NO FILES ASSIGNED IN RECOVERY JCL

- The TIP Recovery module being executed has not been able to read any legitimate LFD's from its job control stream. Check the spelling of the LFD's against the DTF names in

the TIP generation.

TI#112 INSUFFICIENT MEMORY TO LOAD TCA

- The TCA specified in the TIP job stream will not fit in the memory space specified on the JOB card. More memory must be allocated for this TCA.

TI#113 RECOVERY MODULE (TCA) NOT FOUND

- The TCA specified does not exist (or TIP was not generated with journaling) or is incorrectly spelled. Check generation options and recovery job JCL to resolve problem.

TI#114 TIP RECOVERY IN PROGRESS VERSION _____

- Informational: The TIP Recovery module is being executed.

TI#115 SYNTAX ERROR; CORRECT AND TRY AGAIN

- The parameters specified are incorrect for this job. Look them up and fix them!

TI#116 MISSING SEMI-COLON; - CONTINUE? (Y/N)

- The recovery program has detected a syntax error in the control stream. It wants to know whether the user wishes to continue.

TI#117 _____ IS NOT DEFINED (NO DVC-LFD SEQUENCE)

- The JCL for the file named is missing or incorrectly spelled.

TI#118 TOO MANY ERRORS; RECOVERY TERMINATED

- The recovery program has detected too many syntax errors and is terminating automatically.

TI#119 _____ JOURNAL RECORDS READ

- Informational: Statement of the number of journal records processed in forward recovery.

TI#120 _____ JOURNAL RECORDS READ FOR BACKWARD RECOVERY

- Informational: Statement of the number of journal records processed in backward recovery.

TI#121 BEGINNING SCAN FOR QUICK RECOVERY

- Informational: Recovery program is running.

TI#122 TIP/30 FILE RECOVERY COMPLETED

- Informational: Normal job termination.

TI#123 _____ RCVD _____ ON _____ (__/__/__ __:__)

- Informational: Number of records recovered either forward or backward for the named file as of the indicated date and time.

9. CHAPTER IX - APPENDICES

9.1 DIRECTORY OF COBOL COPY BOOKS COPY BOOKS

This appendix lists the COBOL language copy books that are supplied with the TIP/30 system. These copy books (except where noted) are for use in the Data Division of a TIP/30 native mode program.

The copy books are contained in the TIP release library (normally catalogued with the logical file name "TIP").

Listings of the copy books are expanded elsewhere in this reference manual in the sections where their use is recommended.

To obtain a listing of a copy book, the user may run the following transaction:

```
>TLIB PRINT TIP/TC-?????
```

????? is the suffix of the desired copy book.

DIRECTORY OF COBOL COPY BOOKS

Copy Book element name and description.

==========

TC-BITS	Layout of workfields for "TIPBITS" and "TIPBYTES" subroutines.
TC-CDA	Layout of Command Line format of the Continuity Data Area.
TC-DI	Values of popular Device-independent codes (DI-codes) for UNIVAC printers.
TC-FCC	Values used for optional FCC modification when using the TIPMSGO subroutine.
TC-FCS	Values containing function codes used by TIPFCS subroutine.
TC-FDES	Layout of the FCS file descriptor packet.
TC-FLAG	Values that may be used as function codes for use with TIPFLAG subroutine.
TC-LFN	Layout of logical file name packet.
TC-LIBS	Layout of extended file descriptor packet for library files (including record area).
TC-MCS	Layout of MCS work area (parameter passed to TIP native mode programs).
TC-PIB	Layout of TIP/30 Process Information Block (parameter passed to TIP native mode programs).
TC-PRINT	Layout of information packet for TIPPRINT subroutine.
TC-STS	Values of status codes in 9th byte of logical file name packet (LFN).

9.2 Basic Compiler-Interpreter TIP/BASIC

This appendix describes the Allinson-Ross Corporation 'Basic' system (TIP/BASIC). It consists of four subsystems; the monitor, the compiler, the map printer, and the run-time interpreter.

The monitor processes input commands and is responsible for scheduling the compiler, the interpreter, the text editor, the librarian, and the catalogue manager.

The compiler reads an QED file containing the TIP/BASIC source program, performs syntax checks, and if it encounters no errors, produces a dynamic file which contains the the object module. If errors are encountered the compiler outputs a dynamic file which contains the error messages.

The map printer is called by the compiler to produce a diagnostic report, source code list, and object map.

The interpreter loads the object module from the dynamic file and begins executing it.

TIP/BASIC is intended to be used in the interactive mode with a crt or teletype terminal. It includes most features of the proposed ANSI standard Basic (1) as well as extensive string manipulation capabilities.

Section 2 of this manual describes the language elements. Section 3 provides operating instructions for TIP/BASIC. Section 4 describes internal structures of the compiler and interpreter as well a memory requirements. The appendices list compiler and run-time error messages.

9.2.1 DESCRIPTION OF THE TIP/BASIC LANGUAGE

Elements of TIP/BASIC are listed in alphabetical order in this section of the manual. The syntax of the element is shown, followed by a description and examples of its use. The intent is to provide a reference to the features of TIP/BASIC and not to teach the Basic language.

A program consists of one or more properly formed TIP/BASIC statements. An END statement, if present, terminates the program, and additional statements are ignored.

In this section the "Syntax" presents the general form of the element. Square brackets [] denote an optional feature while braces {} indicate that the enclosed section may be repeated zero or more times. Terms enclosed in <> are either non-terminal elements of the language, which are further defined in this section, or terminal symbols. All special characters and capitalized words are terminal symbols.

-+*+-

9.2.2 ABS Predefined Function ABS

Syntax:

ABS (<expression>)

Where:

The ABS function returns the absolute value of the <expression>. The argument must evaluate to a floating point number.

Example:

```
ABS(X)
ABS(X*Y-7**2)
```

-+*+-

9.2.3 ASC Predefined Function ASC

Syntax:

ASC (<expression>)

Where:

The ASC function returns the EBCDIC numeric value of the first character of the <expression>. The argument must evaluate to a string. If the length of the string is zero (null string) an error will occur.

Example:

```
ASC(A$)
ASC("X")
ASC(RIGHT$(A$,7))
```

Additional Considerations:

This function is provided for compatibility with other versions of Basic. It is identical to the EBC predefined function.

-+*+-

9.2.4 ATN Predefined Function ATN

Syntax:

ATN (<expression>)

Where:

The ATN function returns the arctangent of the <expression>. The argument must evaluate to a floating point number.

Example:

```
ATN(X)
ATN(SQR(SIN(X)))
```

Additional Considerations:

All other inverse trigonometric functions may be computed from the arctangent using simple indentities.

-+*+-

9.2.5 CALL Statment CALL

Syntax:

[<line number>] CALL <expression>

Where:

The CALL statement is used to call a TIP/BASIC program. When the called program terminates control is passed to the statement following the CALL statement.

Example:

```
10 CALL "FOOTBALL"
   CALL PROGRAM$
```

-+*+-

9.2.6 CBRT Predefined Function CBRT

Syntax:

CBRT (<expression>)

Where:

The CBRT function returns the cube root of the expression. The expression must evaluate to a floating point number.

Example:

```
CBRT(X)
CBRT( 27 )
```

-+*+-

9.2.7 CHAIN Statement CHAIN

Syntax:

[<line number>] CHAIN <expression>

Where:

CHAIN is a function which will transfer control to another program. The other program must have already been compiled.

Example:

```
10 CHAIN "CALC3"
   CHAIN "CALC" + STR$(NUMBER)
```

-+*+-

9.2.8 CHR$ Predefined Function CHR$

Syntax:

CHR$ (<expression>)

Where:

The CHR$ function returns a character string of length 1 consisting of the character whose EBCDIC equivalent is the <expression> converted to an integer modulo 256. The argument must evaluate to a floating point number.

Example:

```
CHR$(A)
CHR$(28)
CHR$((A+B/C)*SIN(X))
```

Additional Considerations:

CHR$ can be used to send control characters such as a linefeed to the output device. The following statement would accomplish this:
print CHR$(10)

-+*+-

9.2.9 CLK$ Predefined Function CLK$

Syntax:

```
CLK$
```

Where:

The CLK$ function returns an eight byte character string containing the time of day in HH:MM:SS format.

Example:

```
CLK$
PRINT "The time is ";CLK$
```

-+*+-

9.2.10 CLOSE Statement CLOSE

Syntax:

[<line number>] CLOSE #<expression> {; #<expression> }

Where:

The CLOSE statement causes the file specified by each <expression> to be closed. Before the file may be referenced again it must be reopened using a FILE statement. An error occurs if the specified file has not previously appeared in a FILE statement.

Example:

```
 CLOSE #1
150 CLOSE #I; #K; #L*M-N
```

Additional Considerations:

On normal completion of a program all open files are automatically closed by the interpreter. If the program terminates abnormally it is possible that files created by the program will be lost.

-+*+-

9.2.11 <constant> <constant>

Syntax:

[<sign>] <integer>. [<integer>] [e [<sign>] <exp>]
["] <character string> ["]

Where:

A <constant> may be either a numeric constant or a string constant. All numeric constants are stored as floating point numbers. Strings may contain any EBCDIC character except a carriage return (X'0D').

Numeric constants may be either signed or unsigned integer, decimal numbers (or expressed in scientific notation). Numbers up to 15 characters in length are accepted but the floating point representation of the number maintains approximately seven significant digits (1 part in 16,000,000). The largest magnitude that can be represented is approximately 3.6 times ten to the 38th power. The smallest non-zero magnitude that can be represented is approximately 2.7 times ten to the minus 39th power.

String constants may be up to 255 characters in length. Strings entered from the terminal, or read from a disk file may be either enclosed in quotation marks or delimited by a comma. Strings used as constants in the program must be enclosed in quotation marks.

Example:

```
 10
-100.75639e-19
"This is the answer"
```

-+*+-

9.2.12 COS Predefined Function COS

Syntax:

COS (<expression>)

Where:

COS is a function which returns the cosine of the <expression>. The argument must evaluate to a floating point number expressed in radians.

Example:

```
COS(B)
COS(SQR(X-Y))
```

-+*+-

9.2.13 COSH Predefined Function COSH

Syntax:

COSH (<expression>)

Where:

COSH is a function which returns the hyperbolic cosine of the <expression>. The argument must evaluate to a floating point number.

Example:

```
COSH(X)
COSH(X*2+Y*2)
```

-+*+-

9.2.14 DAT$ Predefined Function DAT$

Syntax:

DAT$

Where:

The DAT$ function returns an eight byte character string containing the date in YY/MM/DD format.

Example:

```
DAT$
PRINT "The date is ";DAT$
```

-+*+-

9.2.15 DATA Statement DATA

Syntax:

[<line number>] DATA <constant> {, <constant> }

Where:

DATA statements define string and floating point constants which are assigned to variables via a read statement. Any number of DATA statements may occur in a program.

Strings must be enclosed in quotation marks.

Example:

```
10 DATA 10.0,11.72,100
   DATA "XYZ",11.,"this is a string"
```

-+*+-

9.2.16 DIM Statement DIM

Syntax:

```
[ <line number> ] DIM <identifier> ( <subscript list> )
{ ,<identifier> ( <subscript list> ) }
```

Where:

The dimension statement dynamically allocates space for floating point or string arrays. String array elements may be of any length up to 255 bytes and change in length dynamically as they assume different values.

Initially, all floating point arrays are set to zero and all string arrays are null strings. An array must be dimensioned explicitly; no default options are provided. Arrays are stored in row major order.

Expressions in subscript lists are evaluated as integers when determining the size of the array. All subscripts have an implied lower bound of 1. When array elements are referenced a check is made to ensure the element resides in the referenced array.

Example:

```
    DIM A(10,20), B(10)
150 DIM B$(2,5,10)
```

Additional Considerations:

A <DIM statement> must be placed before the first reference to the array it describes.

-+*+-

9.2.17 EBC Predefined Function EBC

Syntax:

EBC (<expression>)

Where:

The EBC function returns the EBCDIC numeric value of The first character of the <expression>. The argument must evaluate to a string. If the length of the string is zero (null string) an error will occur.

Example:

```
EBC(A$)
EBC("X")
EBC(RIGHT$(A$,7))
```

-+*+-

9.2.18 END Statement — END

Syntax:

[<line number>] END

Where:

An END statement indicates the end of the source program. It is optional and, if present, it terminates reading of the source program. If any statments follow the END statement they are ignored.

Example:

```
10 END
   END
```

-+*+-

9.2.19 ENDIF Statement

ENDIF

Syntax:

[<line number>] ENDIF

Where:

An ENDIF statement is used to indicate the end of a nested IF statement list.

Example:

```
IF A EQ B                \
    IF C NE D            \
        LET X = 1        \
    ELSE                 \
        LET X = 2        \
    ENDIF                \
    PRINT X              \
ELSE                     \
    LET Y = 4            \
    PRINT Y
```

-+*+-

9.2.20 EXITFOR Statement EXITFOR

Syntax:

[<line number>] EXITFOR

Where:

The EXITFOR statement is used to transfer control out of a FOR-NEXT loop. Control is passed to the statement immediately following the NEXT statement of the current FOR-NEXT loop.

Example:

```
FOR I = 1 TO 10
    IF TABLE(I) = 0 THEN EXITFOR
    PRINT TABLE(I)
NEXT
```

-+*+-

9.2.21 EXP Predefined Function EXP

Syntax:

EXP (<expression>)

Where:

The EXP function returns 'e' raised to the power of the <expression>. The argument must evaluate to a floating point number. Note: e = 2.71828.....

Example:

```
EXP(X)
EXP(X*LOG(N))
```

-+*+-

EXPRESSION

9.2.22 <expression> <expression>

Syntax:

Where:

Expressions consist of algebraic combinations of variables, constants, and operators. The hierarchy of operators is:

1) ()
2) ^ or ** exponentiation
3) *, /
4) +, -, concat (+), unary +, unary -
5) relational ops <, <=, >, >=, =, <>
   LT, LE, GT, GE, EQ, NE
6) NOT
7) AND
8) OR, XOR

Relational operators result in a 0 if false and 1 if true. NOT, AND, OR, and XOR are performed on 32 bit two's complement binary representation of the integer portion of the variable. The result is then converted to a floating point number. String variables may be operated on by relational operators and concatenation only. Mixed string and numeric operations are not permitted.

Example:

```
X + Y
A$ + B$
(A <= B) OR (C$ > D$) / (A - A AND D)
```

-+*+-

9.2.23 FILE Statement FILE

Syntax:

```
[ <line number> ] FILE #<expression> , <expression>
{; #<expression> , <expression> }
```

Where:

A FILE statement opens files used by the program. The first expression must evaluate to a number between 1 and 16, inclusive. This value is used on PRINT and READ statements to perform I/O to the correct file.

The second expression must be of type string and is used to inform TIP/30's file control system of the file to be opened. Library, edit, and sequential files are supported, in input, update or output mode. The first three parameters are used to identify the file to be opened, the fourth governs access mode.

There may be any number of FILE statements in a program, but only 16 files may concurrently be open.

Example:

```
Library files:
        "file/elt,S,R"   - read a source module
        "file/elt,S,W"   - write a source module
        "file/elt,M,W"   - read a macro
        "file/elt,M,W"   - write a macro

Edit buffers:
        "buffer,,E,R"      - read an edit buffer
        "buffer,,E,W"      - write an edit buffer

Sequential files:
        "lfn,,,R"          - read a sequential file
        "lfn,,,W"          - write a sequential file
        "lfn,,,U"          - update a sequential file

100 FILE #2,"TIP/DATA,S,R"
    FILE #6,"SAMFILE,,,W"; #8,"BUFFER,,E,R"
    FILE #B+C,INPUT$; #D,OUTPUT$
```

-+*+-

9.2.24 FOR Statement FOR

Syntax:

```
[ <line number> ] FOR <index> = <expression> TO <expression>
[ STEP <expression> ]
```

Where:

Execution of all statements between the FOR statement and its corresponding NEXT statement is repeated until the indexing variable, which is incremented by the STEP <expression> after each iteration, reaches the exit criteria. If the step is positive, the loop exit criteria is that the index exceeds the value of the TO <expression>. If the step is negative the index must be less than the TO <expression> for the exit criteria to be met.

The <index> must be an unsubscripted numeric variable and is initally set to the value of the first <expression>. Both the TO and STEP expressions are evaluated on each cycle, and all variables associated with the FOR statement may change within the loop. If a STEP clause is omitted a value of 1 is assumed. A STEP of zero will loop indefinitely.

Example:

```
  FOR I = 1 TO 10
  FOR I = N TO 2 STEP -1
110 FOR INDEX = J + K * 6 TO W - X STEP D
```

-+*+-

9.2.25 GOSUB Statement GOSUB

Syntax:

[<line number>] GOSUB <line number>

[<line number>] GO SUB <line number>

Where:

The address of the next sequential instruction is saved on the run-time stack, and control is transferred to the subroutine labeled with the <line number> following the GOSUB or GO SUB.

Example:

```
10 GOSUB 300
   GO SUB 100
```

Additional Considerations:

The max depth of GOSUB calls allowed is controlled by the size of the run-time stack which is currently set at 32.

-+*+-

9.2.26 GOTO Statement GOTO

Syntax:

[<line number>] GOTO <line number>

[<line number>] GO TO <line number>

Where:

Execution continues at the statement labeled with the <line number> following the GOTO or GO TO.

Example:

```
100 GOTO 50
    GO TO 10
```

-+*+-

9.2.27 <identifier> <identifier>

Syntax:

<letter> { <letter> or <number> or . } [$]

Where:

An identifier begins with an alphabetic character followed by any number of alphanumeric characters, or periods. Only the first 10 characters are considered unique. If the last character is a dollar sign the associated variable is of type string, otherwise it is of type floating point.

Example:

```
A
STRING$
XYZ.ABC
PAY.RECORD
```

Additional Considerations:

All lowercase letters appearing in an <identifier> are converted to uppercase.

-+*+-

9.2.28 IF Statement IF

Syntax:

```
[ <line number> ] IF <expression> THEN <line number>

[ <line number> ] IF <expression> THEN <statement list>
                                      [ ENDIF ]

[ <line number> ] IF <expression> THEN <statement list>
                    ELSE <statement list> [ ENDIF ]
```

Additional Considerations:

If the value of the <expression> is not zero the statements which make up the <statement list> are executed. Otherwise the <statement list> following the ELSE is executed, if present, or the statement following the ENDIF is executed, if present, or the next sequential statement is executed.

In the first form of the statement if the <expression> is not equal to zero, an unconditional branch to the label occurs.

Example:

```
IF (A$<B$)                          \
    AND (C OR D)                    \
        THEN 300

IF B THEN X = 3.0 : GOTO 200

IF J                                \
    AND K                           \
        GO TO 11                    \
ELSE                                \
    GO TO 12
```

-+*+-

9.2.29 IF END Statement IF END

Syntax:

[<line number>] IF END #<expression> THEN <line number>

Where:

The IF END statement sets up an end of file pointer associated with the specified file number. If during a read to the file specified by the <expression>, an end of file is detected, control is transferred to the statement labeled with the line number following the THEN .

Example:

```
   IF END # 1 THEN 100
10 IF END # FILE.NUM - INDEX THEN 700
```

Additional Considerations:

After declaring an input file with the FILE statement an IF END statement should be executed for the specified file before processing it with a READ statement. An IF END statement does not have to be issued after each READ statement.

-+*+-

9.2.30 INPUT Statement INPUT

Syntax:

[<line number>] INPUT [<prompt string>;] <variable>

{, <variable> }

Where:

The <prompt string>, if present, is displayed on the terminal. A line of input data is read from the terminal and assigned to the variables as they appear in the variable list. The data items are separated by commas and/or blanks and terminated by a carriage return. Strings may optionally be enclosed in quotation marks.

If a string is not enclosed by quotes, the first comma terminates the string. If insufficent data is entered, the prompt is redisplayed, and additional input is read until all variables in the list have been filled. If non-numeric data is entered to a numeric field, an error message is displayed. Input is then read from the terminal and assigned to the variables, starting with the field in error.

Example:

```
10 INPUT A,B
   INPUT "size of array?"; N
   INPUT "values?"; A(I),B(I),C(A(I))
```

Additional Considerations:

Trailing blanks in the <prompt string> are ignored.

One blank is always supplied by the system.

-+*+-

9.2.31 INT Predefined Function INT

Syntax:

INT (<expression>)

Where:

The INT function returns the largest integer less than or equal to the value of the <expression>. The argument must evaluate to a floating point number.

Example:

```
INT (amount / 100)
INT(3 * x * SIN(y))
```

-+*+-

9.2.32 LEFT$ Predefined Function LEFT$

Syntax:

LEFT$ (<expression> , <expression>)

Where:

The LEFT$ function returns the 'n' leftmost characters of the first <expression>, where 'n' is equal to the integer portion of the second <expression>. An error occurs if 'n' is zero or negative. If 'n' is greater than the length of the first <expression> then the entire expression is returned. The first argument must evaluate to a string and the second to a floating point number.

Example:

```
LEFT$ (A$,3)
LEFT$(C$+D$,I-J)
```

-+*+-

9.2.33 LEN Predefined Function LEN

Syntax:

LEN (<expression>)

Where:

The LEN function returns the length of the string <expression> passed as an argument. Zero is returned if the argument is the null string.

Example:

```
LEN(A$)
LEN(C$ + B$)
LEN(LASTNAME$ + "," + FIRSTNAME$)
```

-+*+-

9.2.34 LET Statement LET

Syntax:

[<line number>] [LET] <variable> = <expression>

Where:

The <expression> is evaluated and assigned to the <variable> appearing on the left side of the equal sign. The type of the <expression>, either floating point or string, must match the type of the <variable>.

Example:

```
100 LET A = B + C
    LET X(3,A) = 7.32 * Y + X(2,3)
        W = (A<B) OR C$>D$)
    LET AMOUNT$ = DOLLARS$ + "." + CENTS$
```

-+*+-

9.2.35 <line number> <line number>

Syntax:

<digit> { <digit> }

Where:

<line numbers> are optional on all statements and are ignored by the compiler except when they appear in a GOTO, GOSUB, or ON statement. In these cases, the <line number> must appear as the label of one and only one <statement> in the program.

<line numbers> may contain any number of digits but only the first 10 are considered significant by the compiler.

Example:

```
100
1234567890
```

-+*+-

9.2.36 LOG Predefined Function LOG

Syntax:

LOG (<expression>)

Where:

The LOG function returns the natural logarithm of the absolute value of the <expression>. The argument must evaluate to a non-zero floating point number.

Example:

```
LOG (X)
LOG((A + B)/D)
LOGTEN = LOG(X)/LOG(10)
```

-+*+-

9.2.37 LOG10 Predefined Function LOG10

Syntax:

LOG10 (<expression>)

Where:

The LOG10 function returns the base 10 logarithm of the absolute value of the <expression>. The argument must evaluate to a non-zero floating point number.

Example:

```
LOG10(X)
LOG10((A + B)/D)
LOGTEN = LOG10(X)/LOG10(10)
```

-+*+-

9.2.38 MID$ Predefined Function MID$

Syntax:

MID$ (<expression> , <expression> , <expression>)

Where:

The MID$ function returns a string consisting of the 'n' characters of the first <expression> starting at the mth character. The value of 'm' is equal to the integer portion of the second <expression> while 'n' is the integer portion of the third <expression>. The first argument must evaluate to a string, and the second and third arguments must be floating point numbers. If 'n' is greater than the number of characters left in the string all the characters from the mth character are returned. An error occurs if 'm' or 'n' is zero or negative, or if 'm' is greater than the length of the first <expression>.

Example:

```
MID$(A$,I,J)
MID$(B$+C$,5,LENGTH)
```

-+*+-

9.2.39 NEXT Statement NEXT

Syntax:

[<line number>] NEXT <identifier> { ,<identifier> }

Where:

A NEXT statement denotes the end of the closest unpaired FOR statement. The identifier must match the index variable of the paired FOR statement being terminated. Multiple identifiers allow for pairing multiple FOR statements.

Example:

```
10 NEXT I
   NEXT I,J
   NEXT K
```

-+*+-

9.2.40 NEXTFOR Statement NEXTFOR

Syntax:

[<line number>] NEXTFOR

Where:

The NEXTFOR statement transfers control to the NEXT statement of the current loop.

Example:

```
FOR I = 1 TO 10
    IF TABLE(I) = 2 THEN NEXTFOR
    LET TABLE(I) = BALANCE
NEXT I
```

-+*+-

9.2.41 ON Statement ON

Syntax:

(1) [<line number>] ON <expression> GOTO
<line number> {, <line number> }

(2) [<line number>] ON <expression> GO TO
<line number> {, <line number> }

(3) [<line number>] ON <expression> GOSUB
<line number> {, <line number> }

(4) [<line number>] ON <expression> GO SUB
<line number> {, <line number> }

Where:

The <expression>, rounded to the nearest integer value, is used to select the <line number> at which execution will continue. If the <expression> evaluates to 1 the first <line number> is selected and so forth. In the case of an ON ... GOSUB statement the address of the next instruction becomes the return address. The next instruction is executed if the <expression> after rounding is less than one or greater than the number of <line numbers> in the list.

Example:

```
10 ON I GOTO 10, 20, 30, 40
   ON J*K-M GO SUB 10, 1, 1, 10
```

-+*+-

9.2.42 POS Predefined Function POS

Syntax:

```
POS
```

Where:

The POS function returns the current position of the output line buffer pointer. This value will range from 1 to the print buffer size.

Example:

```
PRINT TAB(POS + 3);X
```

-+*+-

9.2.43 PRINT Statement PRINT

Syntax:

(1) [<line number>] PRINT #<expression> , <expression> ;
<expression> {, <expression> }

(2) [<line number>] PRINT #<expression> ;
<expression> {, <expression> }

(3) [<line number>] PRINT <expression> <delim>
{ <expression> <delim> }

Where:

A PRINT statement sends the value of the expressions in the expression list to either a disk file (type(1) and (2)) or the terminal (type (3)).

A type (1) PRINT statement sends a random record specified by the second <expression> to the disk file specified by the first <expression>. An error occurs if there is insufficient space in the record for all values.

A type (2) PRINT statement outputs the next sequential record to the file specified by the <expression> following the #.

A type (3) PRINT statement outputs the value of each <expression> to the terminal. A space is appended to all numeric values and if the numeric item exceeds the right margin then the print buffer is dumped before the item is printed. The <delim> between the <expressions> may be either a comma or a semicolon.

The comma causes automatic spacing to the next tab position (14,28,42,56). If the current print position is greater than 56 then the print buffer is printed and the print position is set to zero. A semicolon indicates no spacing between the printed values. If the last <expression> is not followed by a <delim> the print buffer is dumped and the print position set equal to zero. The buffer is automatically printed anytime the print position exceeds 71.

PRINT Statement

Example:

```
100 PRINT #1,KEY; A,C,A$+"*"
    PRINT #FILE; A / B,D,"end"
    PRINT A, B, "the answer is "; X
```

Error Conditions:

Type 1 PRINT statement is currently not implemented.

-+*+-

9.2.44 RANDOMIZE Statement RANDOMIZE

Syntax:

[<line number>] RANDOMIZE

Where:

A RANDOMIZE statement initializes the random number generator.

Example:

```
10 RANDOMIZE
   RANDOMIZE
```

-+*+-

9.2.45 READ Statement READ

Syntax:

(1) [<line number>] READ #<expression> , <expression> ;
<variable> {, <variable> }

(2) [<line number>] READ #<expression>;
<variable> {, <variable> }

(3) [<line number>] READ <variable> {, <variable> }

Where:

A READ statement assigns values to variables in the variable list from either a file (type (1) and (2)) or from a data statement (type (3)).

Type (1) reads a random record specified by the second expression from the disk file specified by the first expression and assigns the fields in the record to the variables in the variable list. Fields may be floating point or string constants and are delimited by a blank or comma. Strings may optionally be enclosed in quotes. An error occurs if there are more variables than fields in the record.

The type (2) READ statement reads the next sequential record from the file specified by the expression and assigns the fields to variables as described above.

A type (3) READ statement assigns values from data statements to the variables in the list. Data statements are processed sequentially as they appear in the program. An attempt to read past the end of the last data statement produces an error.

Example:

```
100 READ # 1,I; PAY.REG,PAY.OT,HOURS.REG,HOURS.OT
    READ # FILE; NAME$,ADDRESS$,PHONE$,ZIP

    READ A,B,C$
```

Error Conditions:
Type 1 READ statement is currently not implemented.

-+*+-

9.2.46 REM Statement REM

Syntax:

[<line number>] REM [<remark>]
[<line number>] REMARK [<remark>]

Where:

A REM statement is ignored by the compiler and compilation continues with the statement following the next carriage return. The REM statement may be used to document a program. REM statements do not affect the size of a program that may be compiled or executed. An unlabeled REM statement may follow any statement on the same line, and the <line number> may occur in a GOTO, GOSUB or ON statement.

Example:

```
10 REM this is a remark
   REMARK this is also a remark
   LET X = 0 REM initial value of X
```

-+*+-

RESERVED WORD LIST

9.2.47 Reserved Word List

Reserved Word List

Syntax:

<letter> { <letter> } [$]

Where:

The following words are reserved by TIP/BASIC and may not be used as <identifiers>:

ABS	AND	ASC	ATN	CALL
CBRT	CHAIN	CHR$	CLK$	CLOSE
COS	COSH	DAT$	DATA	DIM
EBC	ELSE	END	ENDIF	EQ
EXITFOR	EXP	FILE	FOR	GE
GO	GOSUB	GOTO	GT	IF
INPUT	INT	LE	LEFT$	LEN
LET	LOG	LOG10	LT	MID$
MOD	NE	NEXT	NEXTFOR	NOT
ON	OR	POS	PRINT	RANDOMIZE
READ	REM	RESTORE	RET	RETURN
RIGHT$	RND	SEQ$	SGN	SIN
SINH	SQR	STEP	STOP	STR$
SUB	SYSTEM	TAB	TAN	THEN
TO	TRM$	USR$	VAL	XOR

Reserved words must be preceded and followed by either a special character or a space. Spaces may not be embedded within reserved words. Lowercase letters are converted to uppercase prior to checking to see if an <identifier> is a reserved word.

-+*+-

9.2.48 RESTORE Statement RESTORE

Syntax:

[<line number>] RESTORE

Where:

A RESTORE statement repositions the pointer into the data area so that the next value read with a read statement will be the first item in the first data statement. The effect of a RESTORE statement is to allow re-reading the data statements.

Example:

```
   RESTORE
10 RESTORE
```

-+*+-

9.2.49 RETURN Statement RETURN

Syntax:

[<line number>] RETURN

Where:

Control is returned from a subroutine to the calling routine. Subroutines may be nested up to sixteen levels deep. An error will occur if a RETURN is issued without the corresponding GOSUB.

Example:

```
130 RETURN
    RETURN
```

-+*+-

9.2.50 RIGHT$ Predefined Function RIGHT$

Syntax:

RIGHT$ (<expression> , <expression>)

Where:

The RIGHT$ function returns the 'n' rightmost characters of the first <expression>. The value of 'n' is equal to the integer portion of the second <expression>. If 'n' is zero or negative an error occurs; if 'n' is greater than the length of the first <expression> then the entire <expression> is returned. The first argument must produce a string and the second must produce a floating point number.

Example:

```
RIGHT$(X$,1)
RIGHT$(NAME$,LNG.LAST)
```

-+*+-

9.2.51 RND Predefined Function RND

Syntax:

RND

Where:

The RND function generates a uniformly distributed random number in the range 0 < n < 1.

Example:

```
RND
```

-+*+-

9.2.52 SEG$ Predefined Function SEG$

Syntax:

SEG$ (<expression> , <expression> , <expression>)

Where:

The SEG$ function returns a string consisting of the 'n' characters of the first <expression> starting at the mth character. The value of 'm' is equal to the integer portion of the second <expression> while 'n' is the integer portion of the third <expression>. The first argument must evaluate to a string, and the second and third arguments must be floating point numbers. If 'n' is greater than the number of characters left in the string all the characters from the mth character are returned. An error occurs if 'm' or 'n' is zero or negative or if 'm' is greater than the length of the first <expression>.

Example:

```
SEG$(A$,I,J)
SEG$(B$+C$,5,LENGTH)
```

Additional Considerations:

This function is provided for compatability with other versions of Basic. It is identical to the MID$ function.

-+*+-

9.2.53 SGN Predefined Function SGN

Syntax:

SGN (<expression>)

Where:

The SGN function returns 1 if the value of the <expression> is greater than 0, -1 if the value is less than 0 and 0 if the value of the <expression> is 0. The argument must evaluate to a floating point number.

Example:

```
SGN(X)
SGN(A - B + C)
```

-+*+-

9.2.54 SIN Predefined Function SIN

Syntax:

SIN (<expression>)

Where:

SIN is a predefined function which returns the sine of the <expression>. The argument must evaluate to a floating point number in radians.

Example:

```
X = SIN(Y)
SIN(A - B/C)
```

-+*+-

9.2.55 SINH Predefined Function SINH

Syntax:

SINH (<expression>)

Where:

SINH is a function which returns the hyperbolic sine of the <expression>. The argument must evaluate to a floating point number.

Example:

```
SINH(Y)
SINH(B<C)
```

-+*+-

SPECIAL CHARACTERS

9.2.56 Special Characters

Special Characters

Syntax:

Special Characters

Where:

The following special characters are used by TIP/BASIC:

^ circumflex
(open parenthesis
) closed parenthesis
* asterisk
+ plus
- minus
/ slant
: colon
; semicolon
< less-than
> greater-than
= equal
number-sign
, comma
' quote
cr carriage return
\ backslant

Any special character in the EBCDIC character set may appear in a string. Special characters other than those listed above, if they appear outside a string, will generate an error.

-+*+-

9.2.57 SQR Predefined Function SQR

Syntax:

SQR (<expression>)

Where:

SQR returns the square root of the absolute value of the <expression>. The argument must evaluate to a floating point number.

Example:

```
SQR (Y)
SQR(X^2 + Y^2)
```

-+*+-

STATEMENT

9.2.58 <statement> <statement>

Syntax:

[<line number>] <statement list> <cr>

[<line number>] IF statement <cr>

[<line number>] END statement <cr>

Where:

All TIP/BASIC statements are terminated by line end or carriage return (<cr>).

-+*+-

9.2.59 <statement list> <statement list>

Syntax:

<simple statement> {: <simple statement> }
Where a <simple statement> is one of the following:

CALL statement
CHAIN statement
CLOSE statement
DATA statement
DIM statement
EXITFOR statement
FILE statement
FOR statement
GOSUB statement
GOTO statment
INPUT statement
LET statement
NEXT statement
NEXTFOR statement
ON statement
PRINT statement
RANDOMIZE statement
READ statement
RESTORE statement
RETURN statement
STOP statement
SYSTEM statement
<empty> statement

Where:

A <statement list> allows more than one <statement> to occur on a single line.

STATEMENT LIST

Example:

```
LET i = 0 : LET j = 0 : LET k = 0
X = Y+Z/W : RETURN
:::::: PRINT "This is OK too"
```

-+*+-

9.2.60 STOP Statement STOP

Syntax:

[<line number>] STOP

Where:

Upon encountering a <STOP statement> program execution terminates and all open files are closed. The print buffer is emptied and control returns to the host system. Any number of STOP statements may appear in a program.

A STOP statement is appended to all programs by the compiler.

Example:

```
10 STOP
   STOP
```

-+*+-

9.2.61 STR$ Predefined Function STR$

Syntax:

STR$ (<expression>)

Where:

The STR$ function returns the EBCDIC string which represents the value of the <expression>. The argument must evaluate to a floating point number.

Example:

```
STR$(X)
STR$(3.141617)
```

-+*+-

SUBSCRIPT LIST

9.2.62 <subscript list> <subscript list>

Syntax:

<expression> {, <expression> }

Where:

A <subscript list> may be used as part of a <DIM statement> to specify the number of dimensions and extent of each dimension of the array being declared or as part of a <subscripted variable> to indicate which element of an array is being referenced.

There may be up to eight expressions but each must evaluate to a floating point number. A <subscript list> as part of a DIM statement may not contain a reference to the array being dimensioned.

Example:

```
X(10,20,20)
Y$(i,j)
COST(AMT(I),PRICE(I))
```

-+*+-

9.2.63 SYSTEM Statement SYSTEM

Syntax:

[<line number>] SYSTEM <expression>

Where:

The SYSTEM statement is used to call system routines.

Example:

```
100 SYSTEM "TLIB,PRINT,JCS/MYJOB,,AUX1"
    SYSTEM "WHOSON"
```

-+*+-

9.2.64 TAB Predefined Function TAB

Syntax:

TAB (<expression>)

Where:

The TAB function positions the output buffer pointer to the position specified by the integer value of the <expression> rounded to the nearest integer modulo 80. If the value of the rounded expression is less than or equal to the current print position, the print buffer is dumped and the buffer pointer is set as described above. The TAB function may occur only in print statements.

Example:

```
TAB(10)
TAB(I + 1)
```

-+*+-

9.2.65 TAN Predefined Function TAN

Syntax:

TAN (<expression>)

Where:

TAN is a function which returns the tangent of the expression. The argument must be in radians.

An error occurs if the <expression> is a multiple of pi/2 radians.

Example:

```
10 TAN(A)
   TAN(X - 3*COS(Y))
```

-+*+-

9.2.66 THEN Statement THEN

Syntax:

[<line number>] THEN <line number>

Where:

Execution continues at the statement labeled with the line number following the THEN.

Example:

```
    IF A = B THEN 200
110 THEN 220
```

-+*+-

9.2.67 TRM$ Predefined Function TRM$

Syntax:

TRM$

Where:

TRM$ is a function which returns a four byte string containing the terminal name on which the program is running.

Example:

```
 PRINT "You are logged on terminal ";TRM$
100 LET TERMNAME$ = TRM$
```

-+*+-

9.2.68 USR$ Predefined Function USR$

Syntax:

USR$

Where:

USR$ is a function which returns the userid of the person running the program.

Example:

```
 PRINT "Hello ";USR$
235 LET USERID$ = USR$
```

-+*+-

9.2.69 VAL Predefined Function VAL

Syntax:

VAL (<expression>)

Where:

The VAL function converts the number in EBCDIC passed as a parameter into a floating point number. The <expression> must evaluate to a string.

Conversion continues until a character is encountered that is not part of a valid number or until the end of the string is encountered.

Example:

```
VAL(A$)
VAL("3.789" + "e-07" + "This is ignored")
```

-+*+-

9.2.70 <variable> <variable>

Syntax:

<identifier> [(<subscript list>)]

Where:

A <variable> in TIP/BASIC may either represent a floating point number or a string depending on the type of the <identifier>. Subscripted variables must appear in a DIM statement before being used as a <variable>.

Example:

```
X
y$(3,10)
BAS.AMT(x(i),y(i),s(i-1))
```

-+*+-

9.2.71 SAMPLE TIP/BASIC PROGRAM **Sample Program**

The following is a sample program listing.

```
*
*     Guess the random number that the computer has chosen.
*
*     Hint: If you use a binary search technique you will
*           always get the answer in seven or less guesses.
*
100   PRINT "I have a number from 1 to 100"
      PRINT "You must try and guess it."
      LET ANSWER = INT ( RND * 100 ) + 1
      LET GUESSES = 0

200   INPUT "Your guess please";GUESS
      LET GUESSES = GUESSES + 1

      IF GUESS EQ ANSWER                                          \
          PRINT "You got the answer in ";GUESSES;" guesses. "; : \
          IF GUESSES GT 7                                         \
              IF GUESSES LT 10                                    \
                  PRINT "(You could have done better)"          : \
              ELSE                                                \
                  PRINT "(You could have done much better)"     : \
              ENDIF                                               \
          ELSE                                                    \
              PRINT "Well done!"                                : \
          ENDIF                                                   \
          INPUT "Another game";YESNO$                           : \
          IF YESNO$ EQ "Y"                                        \
              GO TO 100                                         : \
          ELSE                                                    \
              STOP

      IF GUESS LT ANSWER                                           \
          PRINT "Sorry but you are low"                            \
      ELSE                                                         \
          PRINT "Sorry but you are high"
      GO TO 200

      END
```

Figure 1

-+*+-

9.2.72 COMPILER STRUCTURE (BCOMP) BCOMP

The compiler structure consists of a table-driven parser which makes one pass over the input, checking statements for correct syntax, while concurrently generating code for the Interpreter to execute.

Floating point numbers are represented in standard OS/3 short format, 32 bits with one sign bit, a 7 bit exponent, and twenty-four bits of fraction. This provides slightly more than seven decimal digits of significance.

Variable length strings and n dimensional arrays are both dynamically allocated at run time.

The TIP/BASIC compiler requires 10k of memory for its code and a minimum of 12k of working storage assigned via the WORKSIZE parameter of the catalogue manager (CAT). The working storage is divided into six dynamic partitions managed by memory routines using a base/displacement concept for each region. Figure 3 illustrates this memory design. The first partition is of fixed size and is used to hold I/O buffers, address, counters, etc. The second region is used to hold the reserved word table and is also of fixed size. Partition three holds the program symbol table. It is actually an extension to the reserved word table as they are identical in format. Regions four and five hold object code and constants respectively. Partition six holds pointers used by the READ function. Regions three through six expand dynamically as required.

If the compiler terminates because of insufficient memory, memory can be added by simply re-cataloguing the compiler with a larger WORKSIZE value.

Compiler Memory Map

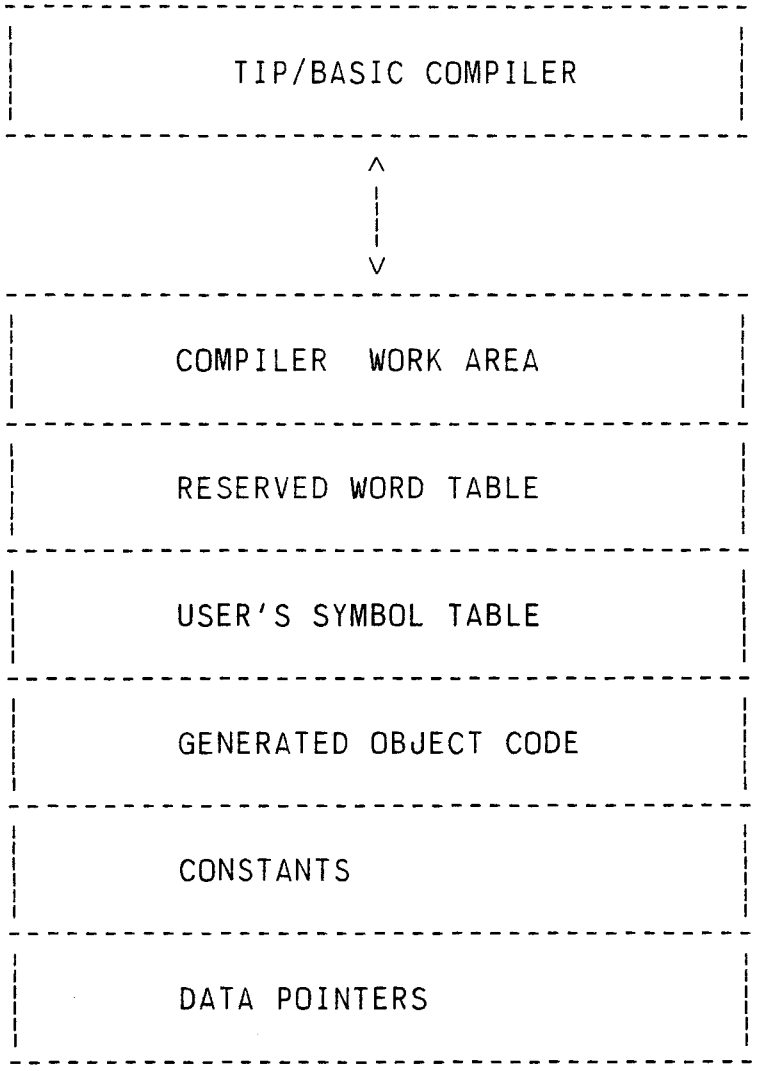


Figure 2

-+*+-

9.2.73 INTERPRETER STRUCTURE (BINT) BINT

The TIP/BASIC interpreter simulates a zero address stack computer with up to 32k of memory.

The interpreter requires 19k of memory for its code and requires free memory to hold the object module, and to build dimensions and strings. The interpreter will display an error message and terminate if free memory becomes exhausted. The interpreter must then be re-catalogued with more work area.

Free memory is divided into five partitions. Figure 4 illustrates the interpreter's memory regions. Partition one is used by the interpreter to hold I/O buffers, addresses, the stack, etc. It is a fixed size. Partition two is used to hold the object code generated by the compiler. This region is dynamically allocated at run time. Partition three, allocated at run time, is used to hold constants. Partition four is used to hold data pointers (used by READ statements) and is also of dynamic size. The remaining free memory is allocated to partition five where it is managed by a memory manager using the boundary tag allocation scheme. It is this region that can get exhausted at run time.

Interpreter Memory Map

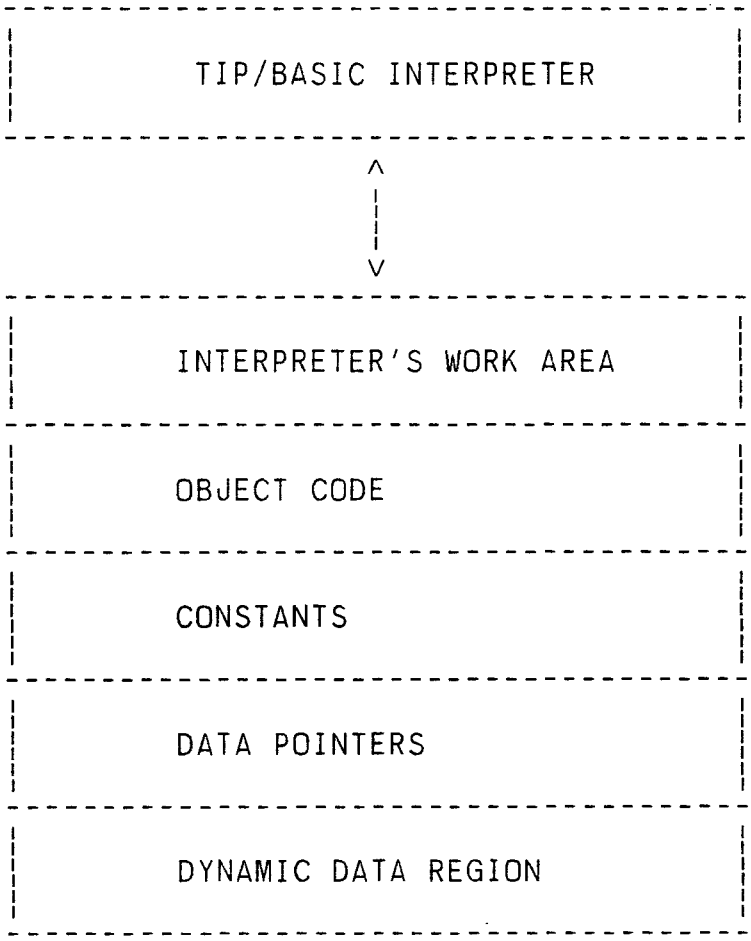


Figure 3

-+*+-

9.2.74 RUN-TIME MONITOR ERROR MESSAGES errors

BI#01 Line ____ Open routine detected invalid file number

BI#02 Line ____ Attempt to open a previously opened file

BI#03 Line ____ Fcs error during file open routine

BI#04 Line ____ Close routine detected invalid file number

BI#05 Line ____ Attempt to close a closed file

BI#06 Line ____ Fcs error during file close routine

BI#07 Line ____ Invalid file number on read function

BI#08 Line ____ Attempt to read an unopened file

BI#09 Line ____ End of file reached. No if end statement

BI#10 Line ____ I/O error during read function

BI#11 Line ____ Invalid file number on print function

BI#12 Line ____ Attempt to write to an unopened file

BI#13 Line ____ I/O error during print function

BI#14 Line ____ Attempt to read past end of record

BI#15 Line ____ Invalid file number on if statement

BI#16 Line ____ Insufficient memory

BI#17 Line ____ Unable to find object file

BI#18 Line ____ Invalid data, re-enter

BI#19 Line ____ Array index out of bounds

BI#20 Line ____ Attempt to read beyond available data

BI#21 Line ____ Attempt to read wrong data type

BI#22 Line ____ Return issued before a gosub

BI#23 Line ____ Invalid length for RIGHT$ function

BI#24 Line ____ Invalid length for LEFT$ function

BI#25 Line ____ Invalid length for MID$ function

BI#26 Line ____ Invalid displacement for MID$ function

-+*+-

10. KWIC INDEX **INDEX**

- A -

- B -

KWIC INDEX

- C -

KWIC INDEX

KWIC INDEX

- D -

KWIC INDEX

KWIC INDEX

- E -

KWIC INDEX

- F -

KWIC INDEX

KWIC INDEX

- G -

- H -

- I -

- J -

- K -

- L -

KWIC INDEX

- M -

- N -

- O -

- P -

KWIC INDEX

KWIC INDEX

- Q -

KWIC INDEX

- R -

KWIC INDEX

KWIC INDEX

KWIC INDEX

- U -

- V -

- E n d O f D o c u m e n t -