

BED ESD TR-67-372
ESTI FILE COPY

A DESCRIPTION OF THE INTERNAL OPERATION
OF THE ADAM SYSTEM

AUGUST 1967

J. A. Clapp

Prepared for
DEPUTY FOR COMMAND SYSTEMS
COMPUTER AND DISPLAY DIVISION
ELECTRONIC SYSTEMS DIVISION
AIR FORCE SYSTEMS COMMAND
UNITED STATES AIR FORCE
L. G. Hanscom Field, Bedford, Massachusetts



This document has been approved for public release and sale; its distribution is unlimited.

Project 502F
Prepared by
THE MITRE CORPORATION
Bedford, Massachusetts
Contract AF19(628)-5165

ADD 660581

When US Government drawings, specifications, or other data are used for any purpose other than a definitely related government procurement operation, the government thereby incurs no responsibility nor any obligation whatsoever; and the fact that the government may have formulated, furnished, or in any way supplied the said drawings, specifications, or other data is not to be regarded by implication or otherwise, as in any manner licensing the holder or any other person or corporation, or conveying any rights or permission to manufacture, use, or sell any patented invention that may in any way be related thereto.

Do not return this copy. Retain or destroy.

A DESCRIPTION OF THE INTERNAL OPERATION
OF THE ADAM SYSTEM

AUGUST 1967

J. A. Clapp

Prepared for
DEPUTY FOR COMMAND SYSTEMS
COMPUTER AND DISPLAY DIVISION
ELECTRONIC SYSTEMS DIVISION
AIR FORCE SYSTEMS COMMAND
UNITED STATES AIR FORCE
L. G. Hanscom Field, Bedford, Massachusetts



This document has been approved for public release and sale; its distribution is unlimited.

Project 502F
Prepared by
THE MITRE CORPORATION
Bedford, Massachusetts
Contract AF19(628)-5165

FOREWORD

This document was prepared by The MITRE Corporation for the Deputy for Command Systems, Computer and Display Division, of the Electronic Systems Division, Air Force Systems Command, L. G. Hanscom Field, Bedford, Massachusetts.

REVIEW AND APPROVAL

This technical report has been reviewed and is approved.



for CHARLES A. LAUSTRUP, Colonel, USAF
Chief, Computer and Display Division

ABSTRACT

This report summarizes the internal operation of the ADAM system. It describes the organization of functions among the system routines. Appendix I lists the sizes of the primary routines. Appendix II describes the internal format of ADAM files and rolls.

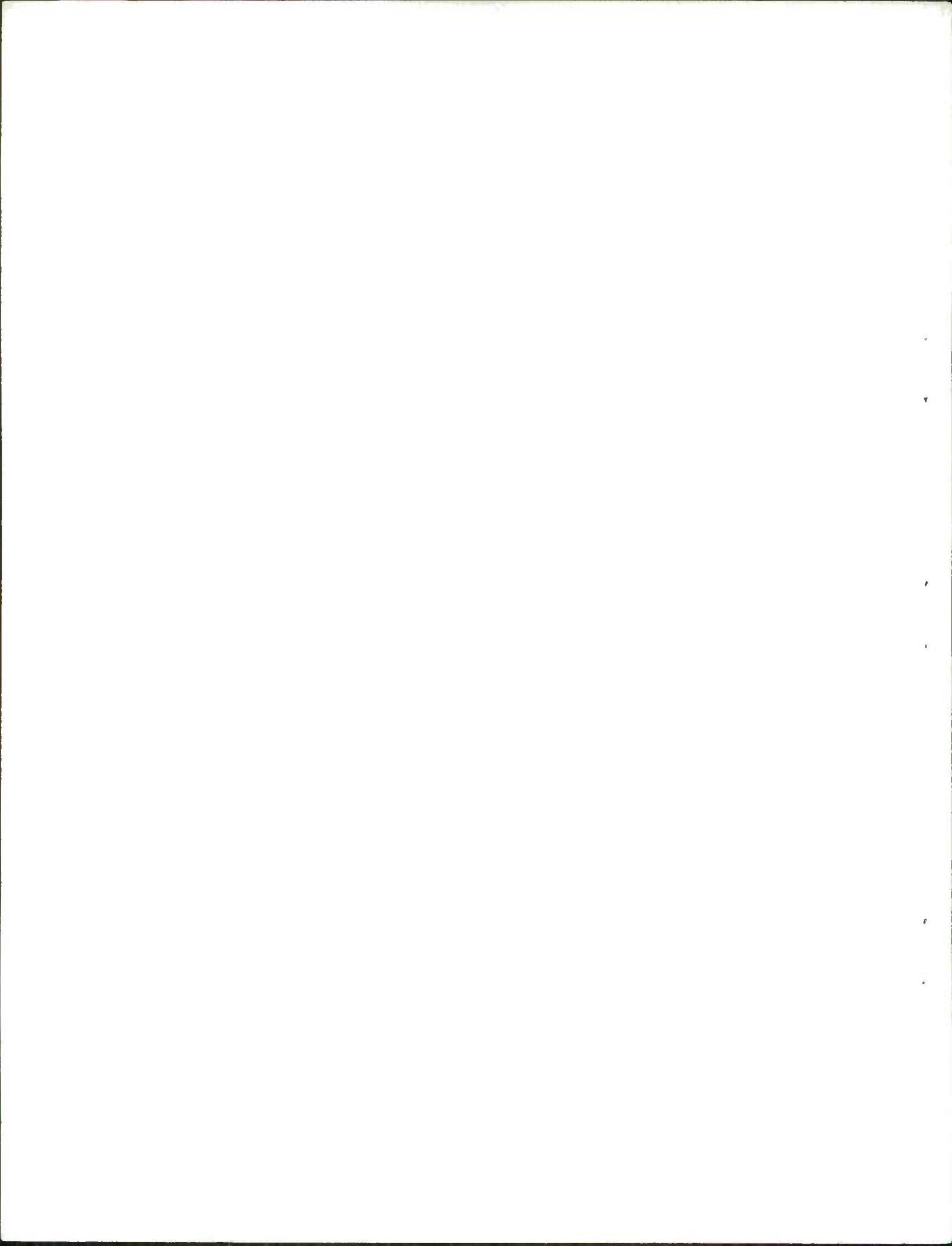


TABLE OF CONTENTS

	<u>Page</u>
SECTION 1.0 BASIC OPERATION OF THE ADAM SYSTEM	1
SECTION 2.0 INITIALIZATION AND MODIFICATION	3
2.1 INITIALIZATION	3
2.2 MODIFICATION	5
SECTION 3.0 INPUT-OUTPUT	8
3.1 INPUT MESSAGES	8
3.2 DISPLAY INPUTS	12
3.3 OUTPUT MESSAGES	13
SECTION 4.0 TASK CONTROL	14
SECTION 5.0 TASK PROCESSING	17
5.1 TYPES OF TASKS	17
5.2 MESSAGE PROCESSING ROUTINES	17
5.3 THRUPUT PROCESSING ROUTINES	18
5.4 DISPLAY INPUT PROCESSING ROUTINES	20
SECTION 6.0 DYNAMIC STORAGE ALLOCATION	22
6.1 CORE MEMORY ALLOCATION	22
6.2 DISK ALLOCATION	26
SECTION 7.0 FILE AND ROLL PROCESSING ROUTINES	27
7.1 FILE PROCESSING ROUTINES	27
7.2 ROLL PROCESSING ROUTINES	30
SECTION 8.0 LANGUAGE PROCESSORS	33
8.1 DAMSEL	33
8.2 LAP	33
8.3 COMFORT MONITOR AND POSTPROCESSOR	33

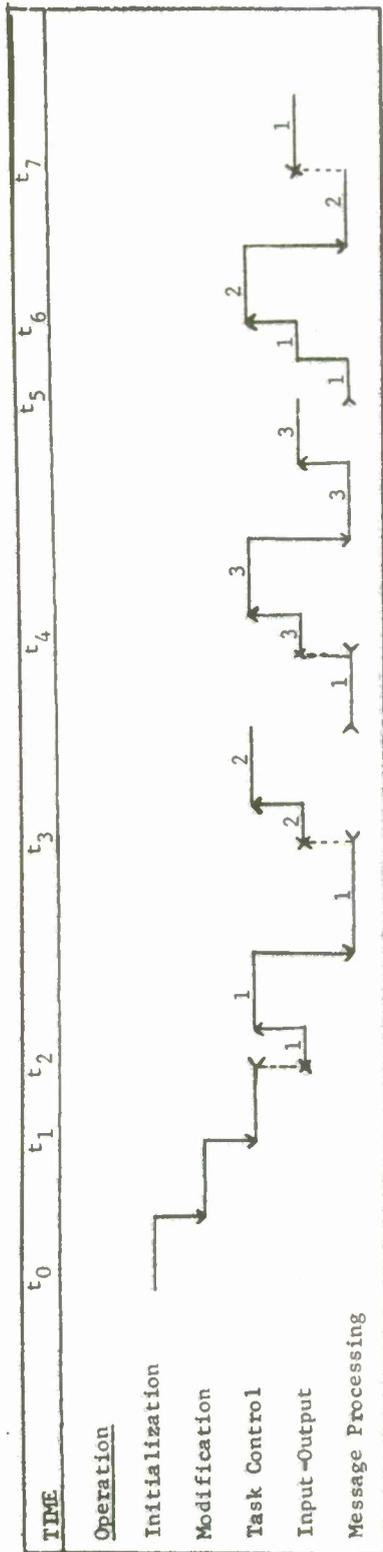
TABLE OF CONTENTS (CONTINUED)

	<u>Page</u>
APPENDIX I SIZES OF PRIMARY ADAM ROUTINES	35
APPENDIX II DESCRIPTION OF FORMATS OF ADAM FILES AND ROLLS	38

SECTION 1.0

BASIC OPERATION OF THE ADAM SYSTEM

Figure 1 shows the basic behavior of the ADAM system which operates as a single job in the job queue of MCP, the monitor control program of the IBM 7030. After the system is 'ready' (see time t_2) it operates under user control based on input messages which may be entered on-line via input devices or off-line as a part of his job deck. Each phase of operation is discussed briefly in the following sections in terms of the general flow of control and data and the names of the primary routines involved. This is intended to orient the reader in a general way to the internal organization of the ADAM system.



Key

- Time in which an operation occurs.
- ↓ Change of control to a new operation.
- ← Operation interrupted by input/output.
- Operation resumed after an interrupt.
- ⋯ Change of control due to an interrupt.
- 1 Number of message being handled.

- t₀ Start of ADAM job. Initialization and modification of the ADAM system takes place
- t₁ Start of ADAM Operating System. Task Control waits for inputs.
- t₂ Time of receipt of first message. This interrupts the Task Control. Because no messages were waiting to be processed, message 1 is processed (as indicated by 1 over Message Processing).
- t₃ Time of receipt of message 2. This interrupts Message Processing. Task Control determines that message 2 is not of higher priority than message 1, so Message Processing is resumed where it was interrupted.
- t₄ Time of receipt of message 3. In this case Task Control decides that message 3 is of higher priority and hence a bypass task which is completely processed before processing of message 1 is resumed at t₅.
- t₅ Time of completion of message 3 processing.
- t₆ Time of completion of processing of message 1. Task Control then starts the processing of message 2.
- t₇ Time of interrupt indicating that hardware is ready for more of output from message 1.

Figure 1. Basic Operation of the ADAM System

SECTION 2.0

INITIALIZATION AND MODIFICATION

2.1 INITIALIZATION

When the ADAM system is not being operated, it exists as a system tape containing a core image and data to be stored on the disk. Data in the ADAM system is organized into two basic structures: files and rolls. Rolls are dictionaries or directories containing names and values related to the names. Almost all of the routines and data of the ADAM system are contained in the files and rolls which are stored on tape. For example, there is a routine file which contains binary images of the routines which comprise the ADAM system.

When an ADAM job starts, the deck which is used contains a tape reading routine called TWRT which reads a specific tape IOD and expects a specific format. It reads one file and transfers control to a location in the last record.

The next set of routines to operate are BS1 and BS2. Together they read from tape and generate a core image containing initialized versions of routines, and store on the disk the data for files and rolls. BS1 and BS2 routines may be used whenever the ADAM system is loaded from tape, e.g., on recovery from an error or under user control via an input message.

Following system generation, control is given to IPLOAD. IPLOAD is primarily a card loading routine. It can load binary cards and octal correction cards. In this way locations in core memory can be modified. In addition, IPLOAD reads and interprets special control cards as follows:

(1) T LOAD,RFILE ---

This is a command to load a specific routine from the routine file into core memory. IPLOAD calls on CLOD, the ADAM system routine loader to perform the loading.

(2) T END ---

This is a command to give control to a specified location in memory where, hopefully, there is a routine to be executed. The command can also specify parameters for the routine. In this way the user may operate various routines before starting the ADAM Operating System.

(3) T OUT ---

This command specifies a device and a message to be sent to that device and IPLOAD causes the transmission of the message.

(4) T LOAD,GO

This command causes transfer of control to TIP first to read any control cards for off-line messages and process them as specified (see Section on Input-Output). Then control is given to COP, the task control routine for the ADAM Operating System. Subsequent to this command input messages may be processed by the ADAM system.

IPLOAD may also be used after system operation has begun by a user message (\$RESUME) so that further card reading may be done.

(5) T LOAD,READ

This is similar to T LOAD,GO except for the manner in which the off-line input messages which follow it are specified. IPLOAD reads the next card, and checks to see if it contains \$CONTINUE. If so, it is sent to TIP, as an off-line input message. If not, IPLOAD itself creates a \$CONTINUE message which it sends to TIP. Control is then given to COP for Task Control.

2.2 MODIFICATION

Modification of the ADAM system is usually accomplished by a command to load a routine and a command to execute it using control cards described above. Below is a summary of some of the routines which may be used to modify the system.

2.2.1 DABS

DABS is a routine which reads and writes specified parts of a data base to and from tape. DABS deals with a data base tape, as opposed to a systems tape described earlier. A data base tape contains images of data to be stored on the disk but no core images. DABS acts under control of cards in the deck which are of two basic forms:

T RESTORE ---

which reads and stores on disk the files and rolls which are on the specified data tape, overwriting the same files or rolls if they are already in the system.

T SAVE ---

which writes a data tape with the files and rolls specified, which exist in the ADAM system.

DABS may also be called during system operation via a routine called USAVE with entry points to save and restore a data base tape, e.g., DO TSAVE(DATA,5,0) is a FABLE message to save the list of files and rolls and routines named in the file called DATA.

2.2.2 RUE

RUE updates and edits the Routine File and its associated rolls, the Routine Roll and the Compiler Roll. It reads control cards indicating the name of the routine and the operation desired (DELETE, ADD, CORRECT) and reads the appropriate cards, if ADD or CORRECT, which specify the changes. RUE itself is called by a card beginning

T RUE ----.

After the Routine File has been updated, subsequent loading of these routines into core will cause the modified versions to be loaded.

2.2.3 CHANGER

CHANGER is a routine which is used to make changes to the language file, which contains, for each language, specifications of the language diagram (syntax and semantics), the literals (words which are in the language) and character types (alphanumeric or punctuation). CHANGER reads cards containing one or more parts of a language specification and adds to or modifies the language file. Language specifications of a diagram and literals may be produced by the LAP routine, an assembly program which must be operated before the ADAM job because CHANGER can only accept cards which must be physically inserted into the ADAM job deck after they are produced by LAP.

2.2.4 FORUP

FORUP is a routine which adds and replaces formats in the Format File. It reads cards containing a compiled format which consists of the format name and binary code for the format.

2.2.5 DEMON

This routine accepts specifications for message identification and message monitoring which name the devices to be monitored, the devices on which monitoring takes place, and the devices whose messages shall be identified (see Input-Output). DEMON may also be requested to terminate any such specifications.

SECTION 3.0

INPUT-OUTPUT

3.1 INPUT MESSAGES

Input messages may be entered on-line from a variety of devices such as a typewriter, a teletype, pushbuttons and a light pencil. Inputs may also be entered off-line via messages on punched cards with control cards indicating the simulated source and time at which the message is to be entered. The two basic sources are handled differently. Messages may also be internally generated by a routine and then processed as if they came from an external source.

3.1.1 On-Line Input Messages

Input messages from real time devices cause hardware I/O interrupts. Such interrupts are handled by a system routine called TOP, which edits the message as follows:

- (a) Characters are converted from the code used by the particular device to A8 code, a standard code used by routines in the system which subsequently operate on the message. This makes such routines independent of the type of input device.
- (b) During conversion, the message is cleaned up by removing control characters (carriage return, line feed) and interpreting backspace and delete codes.
- (c) The proper ending character is appended to the message (i.e., Partial Message or End of Message).
- (d) A source code is inserted before the message, indicating the device from which it came.

After TOP has edited the input message it is sent to a routine called TIP. TIP's primary function in handling on-line input messages is to collect partial messages and concatenate them with subsequent message fragments from the same source until the entire message is received. Each complete message is sent by TIP to COP, the Task Control routine, to give it a priority and place it in the Task Queue. (See description of Task Control below.)

3.1.2 Message Monitoring and Identification

A user may optionally specify message monitoring and/or message identification for on-line input devices. Message monitoring means all inputs and/or outputs of a specified device are reproduced on one or more other output devices, e.g., the inputs from a remote teletype may be monitored on a printer at SDL and the outputs to the teletype may be monitored on the same or a different printer.

Monitoring of input from a device is accomplished as follows:

After editing an input message, TOP checks its source against an indicator for that device which tells whether its input is being monitored. If so, the input is passed, as usual, to TIP and, in addition, an indicator called TOPMO\$A is set. On sensing that indicator, TIP calls a fixed routine named TIPSI which calls TOP to send the message to the monitoring device(s).

Monitoring of outputs is accomplished within TOP by a check to see if the device to which the message is directed is being monitored. If so, the message is also sent to the monitoring device(s).

Message identification means that each input message from a specified device or devices is given a unique number. This number is sent back to the user. If the processing of a message results in an error, and the message has been identified by a number, the number will be included in the error message.

Message identification is accomplished as follows:

After editing an input message, TOP checks its source against an indicator for that device which tells whether message identification has been specified. If so, the input is passed, as usual, to TIP and in addition an indicator called TOPID\$A is set. On sensing that indicator, TIP calls a fixed routine named TIPID which calls TOP with a message to the input device containing the new query number. The number is also stored in the Task Queue and is available during the processing of that message.

A routine called DEMON may be called via FABLE to specify or cancel message identification and message monitoring. It sets the indicators which TOP senses when inputs and outputs are received.

3.1.3 Off-line Input Messages

The deck used in an ADAM job may contain a T LOAD,GO card to start operation of the ADAM system following initialization and modification. Control is given to TIP, a routine which, in addition to the functions described in the preceding sections, also handles off-line input messages. Off-line messages are each preceded by a control card, called a TIPSIM card. It is

recognized by TIP from a 1-2-3 punch in column 1. The card contains a device number, indicating the simulated source of the message and a simulated present time as well as a time at which the message should be read. TIP calculates the time difference and uses it to set the interval timer. If there is no difference, the time is set to one millisecond. One special TIPSIM card, the END card, allows TIP to bypass setting the interval timer since there are no further messages. Any deck in which there are no off-line messages contains at least a TIPSIM card with a long delay or a TIPSIM END card.

When the interval timer interrupt occurs, TIP reads cards until the next TIPSIM control card. All cards between TIPSIM cards are treated as one message. TIP edits the message as follows:

- (1) The message is converted from card code to A8 code.
- (2) The device number specified on the TIPSIM card is inserted before the message.
- (3) An End-of-Message is inserted after the message.

TIP then sends the message to COP to be added to the Task Queue. The next TIPSIM card is then used to set (or not set) the interval timer before TIP exits.

Another method of entering off-line messages is through the utility message \$CONTINUE and an ILOAD message I LOAD,READ (see preceding discussion in Section 2.1).

A \$CONTINUE message is processed by a subroutine in RULE, a system routine containing subroutines for some of the system utility message processing. This subroutine reads the cards containing a message, determines the end of the message (card has blanks in columns 73-80) and edits the message in exactly

the same manner as TIP usually does. In this case the device number is implied from the \$CONTINUE card. The subroutine then sends the message to TIP as an internally generated message (see the next section), and also sends another \$CONTINUE message to COP so it may again be called to process subsequent off-line input messages. In this way \$CONTINUE causes each message to be processed before the next message is stacked as input to the system.

3.1.4 Internally Generated Messages

Any routine may generate a message and by calling TIP, cause the message to be stored and given to COP for addition to the Task Queue. The message must be in core. No editing is done by TIP.

3.2 DISPLAY INPUTS

Light pen and display inputs are handled by a routine called BURLEQUE. Recognition rules (see Task Control) for these devices send inputs to BURLEQUE when the I/O-interrupt occurs. The routine stacks the message segments, since a complete message may consist of a sequence of display inputs combined with typewriter inputs.

Special action sequences are used to terminate the message. When the terminal action occurs, BURLEQUE calls TIP with the sequence of inputs as an internally generated message. The message is stacked at the head of the queue of input messages and when unstacked, it is processed by a routine called GHOUL and becomes a typewriter-like message. (See Message Processing.)

For light pen inputs TOP supplies BURLEQUE with data associated with the point which was light-penned. This data is in a Light Pen Input Stream created by the formatting routine.

3.3 OUTPUT MESSAGES

All output messages are sent to TOP along with the identification of the device(s) for which they are intended. TOP queries the messages by device and I/O channel. If the specified channel and device is free, TOP transmits the message. Whenever there is an I/O interrupt TOP checks to see if any device for which a message is queued has become free and, if so, initiates the output transmission. TOP rotates among the devices assigned to a particular channel to provide equal opportunity for all devices to receive queued output.

TOP edits the output message by transforming the A8 characters to the character code of the output device and the correct format required by the hardware. MCP, the 7030 control program, is used to perform the actual output operations.

Although a device is specified for each output, several output devices may be slaved to a device. If the slaved devices are of the same kind, e.g., two printers, then TOP will only format the message once and send it to all the slaved devices, thereby saving time.

SECTION 4.0

TASK CONTROL

A message becomes a task when it is assigned a priority and a routine to process it, and is added to a Task Queue. This process is called message recognition and is performed by a routine called COP. To 'recognize' a message COP uses sets of recognition rules which specify parts of the message to be examined. The rules are arranged as sieves; they are applied to a message in a given order. When a rule successfully completes the recognition of the message, that rule also specifies the priority of the message which designates when (with respect to other tasks in the system) processing of this task should begin and specifies the identity or location and input of a routine to process the message.

The recognition rules are outside of COP and may be dynamically changed. A fixed location, called RCGRUL\$A, points to a control word for the first set of rules. A control word is an index word which contains the location of the first rule, the number of rules and, in its refill field, the location of the control word for the next set of rules.

A fixed table called RCGRUL contains the recognition rules used in ADAM. There are three sets. The first set, called SYSTEM RULES, is intended for rules which are necessary to system operation. It normally contains only one rule which always fails so the next set is applied. This next set is called the USER set and consists of a sequence of rules for each device. The USER set has a control word per device number pointing to the location of the rules for that device. There is a set of rules for typewriter and teletype input and another set for

light pen and push button input.

The card reader, typewriter and teletype input rules consist of a set to test for what are called utility messages (see Task Processing) which are not a part of any input language defined in the Language File and another rule to test for a file generation language message.

The third set of rules is called the COMMON set. It is not indexed by device and contains one rule which is applied when rules in the USER set fail. It recognizes any message as a FABLE message.

When the message is recognized, its location and the values of priority processing routine, entry point, and routine parameters from the rule are placed on the task queue.

There are three kinds of priority:

Immediate: Processing of the task should interrupt the current task. Such a task is called a bypass task.

Normal: The priority may be an indication that the task is put at the top or bottom of the queue. Tasks are unstacked from the top of the queue. As a result messages stacked at the top are unstacked in the reverse order of their receipt while messages stacked on the bottom are unstacked in the order of receipt.

Continue: The task is put on the Continue queue. Tasks on this queue are only unstacked on the request of a routine.

Task processing is initiated by COP whenever a normal task is completed and the processing routine returns control to COP. Bypass task processing is initiated when it has been recognized or at the completion of another bypass task. A bypass task may only interrupt a normal task.

SECTION 5.0

TASK PROCESSING

5.1 TYPES OF TASKS

Task Processing begins when COP, the Task Control routine, gives control to the message processing routine associated with the input message. Based on priority, there are two kinds of tasks: Normal and Bypass. A Bypass task is always operated in autostack mode and is therefore restricted by ADAM convention for autostacked routines as follows:

A routine which operates while in autostack may not cause the location of data or routines in core memory to change. This is necessary so the interrupted task may proceed without being affected by the Bypass task. As a result, a Bypass task may not make any new allocations or request routines not already in core. For this reason Bypass tasks are reserved for operations which do not require lengthy or complex processing.

5.2 MESSAGE PROCESSING ROUTINES

There are three groups of message processing routines. The first group consists of a set of small, relatively simple subroutines which handle what are called Utility messages. These subroutines are contained in a routine called RULE and in the RCGRUL table. A complete list can be found in the operating instructions of the ADAM system. Examples are \$TIME, which prints the time, and \$EOJ which ends an ADAM job.

A second group of processing routines handles display inputs, i.e., inputs from push buttons and light pens. These include routines called BURLEQUE and GHOUL.

A third group of routines, called THRUPUT routines, process messages in FABLE and File Generation Language which require translation and execution and generally involve file manipulation. THRUPUT message processing is a normal (as opposed to Bypass) task.

5.3 THRUPUT PROCESSING ROUTINES

5.3.1 THRUCON

THRUCON is the routine to which COP gives control for a THRUPUT task. The entry-point number, taken from the recognition rule, indicates to THRUCON which sequence of routines should be operated and the routine parameter, also from the recognition rule, specifies which language is needed. THRUCON is the THRUPUT control program. It contains a table of task chains each of which describes in tabular form a sequence of routines to be operated and the inputs to these routines.

THRUCON prepares for translation by reading the appropriate language specification into core memory from the LANGUAGE file and allocates work space which is necessary for communication between the routines in the selected task chain. THRUCON then operates each routine. The routines in FABLE and File Generation Message Processing are described below:

5.3.2 SEPCAN

SEPCAN scans the input message character by character using the specifications for characters kept in the language file for the language of the message. SEPCAN is commonly used to separate alphanumeric strings from punctuation and punctuation from adjacent punctuation by a single space. Spaces are

introduced where necessary and eliminated where redundant. However, the specification for characters may indicate that a character is to be removed, replaced by another, preceded and followed by a space, or that successive occurrences of a character be compressed to one occurrence.

5.3.3 SUBSCAN

Next THRUCON calls SUBSCAN which scans the message for keywords for which string substitutions have been defined. If any are found, the message is altered to replace the keywords by their defined strings.

5.3.4 TRANSLAT

TRANSLAT is the Translator. It is functionally divided into a syntax analyzer and a code generator. The syntax analyzer compares the language specification provided by THRUCON with the message and produces a description of the syntactic structure of the message in the SCAN table, a list of operators called generators which can transform the message into a sequence of operations; and a dictionary called LEX containing parameters for the generators. These parameters, extracted from the message, may be constants or values from rolls.

The final output of the generators is a procedure which can be interpretively operated. This procedure is stored in a stream called SINTAB which consists of operation codes and parameter references. The code generator portion of the translator produces SINTAB. Each generator is a subroutine of the translator which communicates with other generators via common storage and calls on a sequence of composers each of which

constructs one particular kind of SINTAB entry. Associated with SINTAB is SININ, a stream into which the generators store parameters for SINTAB code taken from LEX and FALLOUT, an area used by SINTAB entries for allocation information.

5.3.5 PROCESS

PROCESS is the name of the routine which is called by THRUCON after translation. It acts as an interpreter which operates the code in SINTAB. As such, it decodes SINTAB entries and reformats the parameters as input for system routines which allocate storage, handle files and rolls, and formats.

5.4 DISPLAY INPUT PROCESSING ROUTINES

5.4.1 BURLEQUE

As described in the section on Input-Output, BURLEQUE receives light pen, push button and special typewriter inputs. It saves the sequence of such inputs, checks for an action terminating the sequence, and, on receiving it, causes an interrupt. At that time it calls TIP with an internally generated message from an imaginary source which will be recognized by the rule for that source and be placed as a task on the top of the Normal Task Queue.

5.4.2 GHOUL

The message generated by BURLEQUE is sent to GHOUL. Display inputs are used in conjunction with entries in the CEMETERY file. Each entry is a skeleton message with instructions for inserting into it, as parameters, the display inputs. The inputs must identify which skeleton message should be used. GHOUL retrieves that message and initiates a call to THRUCON

with a special task chain and language entry. The task chain consists of SEPSCAN (see above) and STRIPPER (see below). The message consists of the entry from the CEMETERY file. SEPSCAN is used to separate punctuation and alphanumeric characters in the skeleton message according to the rules specified for the language file for the display language.

5.4.3 STRIPPER

STRIPPER is given the skeleton message and the display inputs. It scans the message and whenever the message indicates an action to insert a display input it executes the action. The fully edited message with its parameters is finally sent to TIP, in the normal fashion, as a typewriter input.

SECTION 6.0

DYNAMIC STORAGE ALLOCATION

The resources which are allocated by ADAM system routines are core memory and disk memory.

6.1 CORE MEMORY ALLOCATION

Core memory is divided into a part at the low-numbered end of memory which is used for routines, and the remainder which is used for data. The location of the boundary between the two parts is stored in LINE\$A and may be dynamically moved to meet the varying demands for each kind of allocation. The primary differences between routine and data allocation are:

- (a) Storage is allocated for routines in 100-word increments, sometimes called glitches, in a direction from low to high-numbered locations while storage is allocated for data in 512 word increments called pages in the opposite direction.
- (b) Once a routine is allocated, it may not move from its assigned location as long as it is in use, while data allocations may be moved when subsequent demands for core allocations occur.
- (c) Allocations for data may be made and then increased. The addition will always be contiguous to the previous allocation although the entire allocation may move to a new location.

6.1.1 Routine Allocation

A routine is designated as fixed or allocatable. Fixed routines are in core memory at the time ADAM system operation begins and remain throughout operation unless a utility message

called \$EXPAND is used to remove them. All routines are entries in the Routine File. When a routine which is being executed requests another routine to be loaded, a system routine called CLOD allocates space for it, and performs the necessary relocation of the code as it is read from the Routine File into the assigned location. A table called PAT contains allocation information about the fixed and allocatable routines. The output to the routine which calls CLOD to load another routine is the location in PAT assigned to that routine. All allocatable routines are accessed through the PAT table. A branch to the PAT location will branch from there to the routine being called or to the loader portion of CLOD if the routine has not yet been loaded.

CLOD may call a subroutine called SPACER in order to request that LINE\$A be moved to allow more room for routine allocation.

A routine which is being executed sets its IN-USE bit in the PAT entry for itself so that it will not be overwritten. When a routine finishes with a routine it has requested, it may dismiss that routine by setting its DISMISS bit in the PAT table (or some routines dismiss themselves). CLOD will reallocate the space allocated to dismissed routines provided it is contiguous with other available space adjacent to the location in LINE\$A, i.e., CLOD will reuse space at the end of its allocation but not fill in holes within the area available for routines with new assignments of routines.

6.1.2 Data Allocation

A routine called BASAL handles requests for data storage allocation in core memory. BASAL, in turn, calls on a routine

named MARASS for the actual assignment of space. A table called ALLOT contains an entry for each allocation which is assigned and for each available sequence of one or more pages. A request for n pages means n consecutive pages. To meet requests for allocations or increases in allocations, BASAL and MARASS may change the location of allocations. All routines which deal with allocatable data space must follow conventions for addressing that space in order to correctly locate the data, even if it is dynamically moved. These conventions involve locating data indirectly through the ALLOT entry which is always updated when the data moves, and using index registers which point to data with a special indicator so the index registers are also updated.

6.1.3 Types of Data Allocation

Allocation of core for data may be made directly through BASAL or by calling other system routines which understand the kind of data or the way in which the data will be addressed.

(a) Area

An area is an allocation which is made by specifying the number of pages needed. Its contents may be anything the user desires. Areas are allocated by BASAL.

(b) File Data

Allocation for a file or an entry from a file may be made by specifying the file and, if desired, the entry. Allocation of core space is then determined from the file description. In the case of file data, allocation also includes reading into core memory the appropriate parts of the file and, when a user steps

through a file, refilling the contents of the core allocation when necessary. BASAL handles allocation of file data.

(c) Streams

A stream is a disk allocation which may be associated with a core allocation by 'attaching' it. Stream handling routines, SHP and STRCON, allow the user to consider a stream as a single series of addressable locations even though a small portion of those locations may actually be in core. SHP and STRCON call BASAL to make core allocations (and SESCON for disk allocations) and then handle the movement of data to and from the disk as the user addresses locations in the stream.

(d) Rolls

Data for a roll is stored in streams. However requests for allocation of a roll are made to ROLCON which determines how much core to allocate based on the size of the roll, core requirements described in the Roll Pointer Set for each roll, and the amount of core space available. ROLCON may also decide to release space in core occupied by other rolls in order to satisfy a request when little core space is available. Users address rolls through ROLCON and the roll routines since a roll whose streams are attached may be detached from core without the user's knowledge. Information about the core and disk assignment of rolls is also contained in the Roll Pointer Set, which is itself part of a roll, called \$ROLL which contains names and descriptive data for all rolls.

6.2 DISK ALLOCATION

SESCON is the ADAM routine in charge of disk allocations. It accepts requests for a disk allocation of a specified number of arcs (512 words) and for increases in existing allocations. Each new allocation is assigned an 'ID' which identifies it. SESCOON also handles disk-to-core and core-to-disk transfers of data. The user of SESCOON specifies the ID of the allocation and the relative locations he wishes to transfer from or to. The actual disk arcs are assigned as needed according to which arcs are available. An allocation may consist of a sequence of nonconsecutive disk arcs which appear to the user to be contiguous. SESCOON maps the user's addressing into the actual arc assignments.

SECTION 7.0

FILE AND ROLL PROCESSING ROUTINES

Data base manipulation involves the processing of data within the two basic data structures available in ADAM: files and rolls. To facilitate the manipulation of such data a number of routines are available. Another form of data manipulation is performed during file generation in order to transform the input data into a form suitable for storing in an ADAM file.

7.1 FILE PROCESSING ROUTINES

7.1.1 FILDEF

The FILDEF routine is used to define new files. It performs two functions. The first creates a new file by assigning it a disk allocation and adding its description to the roll, called \$FILE, containing names and descriptions of all files in the system. In addition, it creates a property roll and an object roll for the file.

The second function adds property descriptions to the property roll of a file and updates the file description.

FILDEF is designed to operate in the environment of the translator and hence its input formats conform to the formats of data available at translation time.

7.1.2 FILGEN

FILGEN is a routine which is used during the processing of input data for file generation. It performs the necessary operations to read data from cards or tape and locate positions in the data corresponding to particular columns or characters. Positions

may also be located by scanning characters and comparing for match or nonmatch with one or more specified strings of characters.

FILGEN also handles conversion of character strings for a set of 'built-in' conversion routines (i.e., not user-supplied) such as card code to A8 or card decimal to binary.

7.1.3 FILMOD

FILMOD consists of a collection of subroutines which perform modification of files and if necessary, modification of associated rolls and allocations in core and on disk. If a new entry is added to a file, entry points to FILMOD exist for functions such as: adding the new object name to the object roll of the file; allocating core and disk space for the new entry and initializing its contents; and, adjusting the contents of the entry, the file descriptions, and the length of the entry after all data has been stored in it. Another called ADD REPETITION handles the addition of a repetition by increasing the size of the entry if necessary. Other functions delete repetitions, delete an object, delete an entire group, and delete a file.

7.1.4 FETCH AND STORE (FS)

FS is a system routine which fetches a value from a file or stores a value in a file. The input to FS is a property description taken from the property roll, and the location in core memory of the entry or repetition from which a value is to be fetched, or into which a value supplied as input should be stored.

7.1.5 BASAL

Although **BASAL** is primarily an allocation routine, it performs additional functions on allocations for file data. **BASAL**, when given the PV of a file, will determine the disk location of file data, the amount of core space required and will read into core the requested data. Since it is more efficient to transfer data to and from the disk at arc boundaries, **BASAL** will provide the calling routine with the location within the allocated core of the requested entry since it is not likely to be the first location. In sequencing through a sequence of entries in a file or a series of repetitions of an entry, **BASAL** may be used to find the absolute location of each entry or repetition via entry points such as **FOR FILE**, **STEP FILE**, **FOR RG**, and **STEP RG**. Because the entire file may not be in core, **STEP FILE** may cause refilling of the core allocation from the disk. This is handled by **BASAL** automatically.

7.1.6 OUTFOP

OUTFOP is the output formatting routine. Its role as a file processing routine is to take a format description and a file and transform the file data into formatted output for a specified device.

7.1.7 FILCO

The deletion of entries or repetitions from a file by **FILMOD** merely sets an indicator to show null data, but the space originally occupied by the data is still in the file. **FILCO** is a housekeeping routine which will compress a file to remove unused space formerly occupied by deleted entries or repetitions.

7.1.8 PRESORT and SORTRT

PRESORT sorts the entries of a file on the values of up to 20 properties which are not in a repeating group. SORTRT sorts the repetitions in the entries of a file on the values of up to 20 properties in the same repeating group. Both routines are given the option of producing the sorted version of the file to replace the original or as a new file.

7.1.9 TALLYRT

TALLYRT is a routine used to tally the values of one or two properties in a file. If a property is arithmetic, the routine is first called with the ranges for the tally. If a property is logical, the first 25 values found in the data are tallied. The routine is initialized with number of properties, types, and ranges, if any. It is then called with one or two values at a time (for one-or-two-way tally) and builds either counts for the tally totals or sums. A final call transfers the results to a file which may be formatted and presented as output. TALLYRT is used in ADAM in conjunction with a tally message in FABLE.

7.2 ROLL PROCESSING ROUTINES

7.2.1 ROLCON

ROLCON is primarily responsible for the allocation of core memory for rolls and their associated roll routines. When a user appeals to ROLCON to open a roll, ROLCON allocates space based on the requirements for that roll, the availability of core space and the amount of space already occupied by rolls. If it is not in core, ROLCON will load the roll routine for that roll. Users are given, by ROLCON, the location of the

roll routine which will process roll operations on the specified roll. ROLCON can also read into the core allocation data from the roll. Other ROLCON functions release the allocation for specific rolls or all rolls or sufficient rolls to meet a given request for space from an allocation routine. ROLCON uses BASAL to allocate the core space.

7.2.2 ROLLR, SROLLR

There are two types of roll structures in general use in the ADAM system (see 7.2.3 for a third type). To the user of rolls the interface presented by all routines is the same. Only the roll routine deals with the roll structure. For a limited number of rolls which are basic to system operation, called special rolls, the format consists of a standard roll and an additional portion called the pointer set containing subvals indexed by PV¹. The three special rolls are the FILES, ROLLS, and ROUT rolls. Their pointer sets remain in allocatable core during system operation and are easily accessed. SROLLR is the roll routine which handles them. ROLLR is the routine for standard rolls. The basic roll operations are:

LOOKUP	which relates a name to PV's and, if specified, subvals.
EVALUATE	which delivers subvals for PV's.
NAME	which delivers names for PV's.
UPDATE	which has various options for modifying a roll.

¹PV stands for Principal Value, a fixed length integer identifying the element in the roll associated with an entry of roll data.

7.2.3 MROLLR

MROLLR is a roll routine used during the file generation of large amounts of data from tape. When presented with inputs such as names or subvals, MROLLR saves the roll operation and values in its input as well as the location in the file where its output would have been stored. Periodically or when all input data have been processed, MROLLR processes the list of roll operations and input parameters which had been saved. The roll operations are executed in an order which minimizes the accessing of portions of a roll and the disk transfers of roll information in order to minimize processing time. Finally, MROLLR makes one pass through the file to store PV's in their proper places. Another routine, MRMC, arranges the environment in which MROLLR operates and restores the environment when MROLLR is finished. The environment changes include loading MROLLR over ROLLR and SROLLR.

7.2.4 MIMIC

MIMIC is a routine which copies the contents of a property roll into another roll. It is used in producing an output file with the same properties as an existing file.

7.2.5 EXPAND and REDUCE

These two routines handle the subvals of a property in a property roll. EXPAND separates the packed fields of a subval into a form which is easier to use. REDUCE takes an expanded set of values for the fields of a subval and packs them into the format used in the roll. Many routines which use property rolls use these routines to simplify handling of subvals.

SECTION 8.0

LANGUAGE PROCESSORS

8.1 DAMSEL

The DAMSEL compiler is a series of routines which operate within the ADAM environment to translate a data-base-sensitive statement language into SMAC code to be assembled by the SMAC and STRAP assembly programs of the 7030. Because SMAC and STRAP may not be called by any routine but MCP, it is not possible to compile a DAMSEL routine in a single job. Instead, the output of the DAMSEL compiler is a tape of input to a second job in which SMAC and STRAP operate.

Associated with DAMSEL is a routine called DAR which may be called when a routine written in DAMSEL language is executed. It interprets calls to other ADAM system routines by the DAMSEL-compiled routine.

8.2 LAP

LAP is a routine which accepts language specifications and produces a binary output for language diagrams and other associated information. LAP is not compatible with the ADAM system and must operate as a separate job outside the ADAM environment.

8.3 COMFORT MONITOR AND POSTPROCESSOR

Routines written in FORTRAN may be operated in the ADAM system provided the routine includes several special subroutine calls to make it compatible with ADAM. The binary deck produced by the FORTRAN compiler must be processed by the COMFORT Post-processor, a nonsystem compatible routine which revises the binary deck to a format suitable for inclusion in the ADAM routine file.

At execution time the COMFORT Monitor must be loaded before any FORTRAN routines may be executed. Along with the Monitor, a table must be loaded containing a list of routines to be operated. The Monitor loads all routines on the list and transfers control to the first routine.

APPENDIX I

SIZES¹ OF PRIMARY ADAM ROUTINES

Initialization and Modification

TWRT	24
BS1	615
BS2	825
IPLOAD	1785
DABS	2783
RUE	1484
CHANGER	319
FORUP	318
DEMON	418
USAVE	337

Input-Output

TOP	3438
TIP	880
TIPSI	63
TIPID	52

Task Control

COP	437
RCGRUL	491

¹Size may include data space as well as executable code.

Task Processing

RULE	1460
BURLEQUE	448
GHOUL	791
STRIPPER	7873
THRUCON	980
SEPCAN	274
SUBSCAN	1413
TRANSLAT	9050
PROCESS	1766

Allocation

CLOD	1608
BASAL	1167
MARASS	942
SHP	307
STRCON	549
SESCON	2071

File and Roll Processing

FILDEF	974
FILGEN	1696
FILMOD	1278
FS	208
OUTFOP	3457
FILCO	915
PRESORT	163
SORTRT	2107
TALLYRT	427
ROLCON	481

File and Roll Processing (Continued)

ROLLR	1745
SROLLR	1439
MROLLR	1792
MRMC	52
MIMIC	335
EXPAND	127
REDUCE	134

Language Processors

DAMSEL	18596
DAR	815
LAP	3050
COMFORT Monitor	1123
COMFORT Preprocessor	550

APPENDIX II

DESCRIPTION OF FORMATS OF ADAM FILES AND ROLLS

The structure of files in the ADAM system is completely serial, that is all the data for one object is considered to be contiguous. The format of a file is shown in Figure 2. In the following descriptions an asterisk (*) indicates parts which were not used by the rest of the system.

1.1 MISCELLANEOUS SECTION

The first section is called MISCELLANEOUS (or MISC). Data which occurs on a "once per file" basis is stored in a Miscellaneous section. The Miscellaneous section is divided into four subsections, and the first four words are four pointers to the four subsections in the order shown below. These pointers have the following format:

- (a) Each pointer is a full word.
- (b) Bits 0-24 of each pointer contain a bit location in value field format (B,25,1). The remaining bits may be given different uses in each pointer.
- (c) The location is a relative bit location from the start of the Miscellaneous section.

The pointers are the first four words of MISC, in the following order: Storage Monitoring, Query-valued*, Event Detection, and File Properties.

The four subsections are listed below.

1.1.1 Storage Monitoring

At least two kinds of storage monitoring are controlled here -- file protected data and event detection.* Each is 1-bit per property and is indexed by property PV. There are 4-bits per PV.

1.1.2 Private Query*

These are stored queries which are referenced by query valued properties in the file.

1.1.3 Event Detection*

These are descriptions of what is to be done in case a property with a set event detection bit is updated.

1.1.4 Property*

These are user defined "once per file" data. This subsection is organized like an entry. These properties are described in the property roll.

1.2 MAIN SECTION

The Main section consists of the entries. The format of each entry is shown in Figure 2. An entry always starts at a full word boundary.

1.2.1 Standard Data

A set of properties for each entry. See Figure 3 for the format and description of these properties.

1.2.2 Fixed Data

This portion of an entry is the same length for each entry in a file. It contains values for fixed length properties which are not in repeating groups (see below) and pointers to variable length properties.

1.2.3 Variable Data

All variable length values and all data for repeating groups are stored in the variable data section of an entry. Pointers to Variable Data from the Fixed Section include a count of the number of bits being pointed to and may also indicate the location of a continuation pointer which, in turn, points to additional data and may also indicate the location of a continuation pointer. Hence a variable data property may be stored in groups of consecutive bits with discontinuities within the data. The format and use of pointers are described in 1.6 on page 46.

1.2.4 Slop

Slop is between fixed and variable data and allows expansion of variable data without increasing the size of entry, and hence dislocating all subsequent entries. If necessary, however, additional Slop can be obtained.

1.3 FORMATS OF PROPERTY VALUES

Data can, in general, be divided into two types -- Fixed and Variable.

Fixed data is of a predictable length and therefore has the same relative position in each entry or repetition. Fixed data types are:

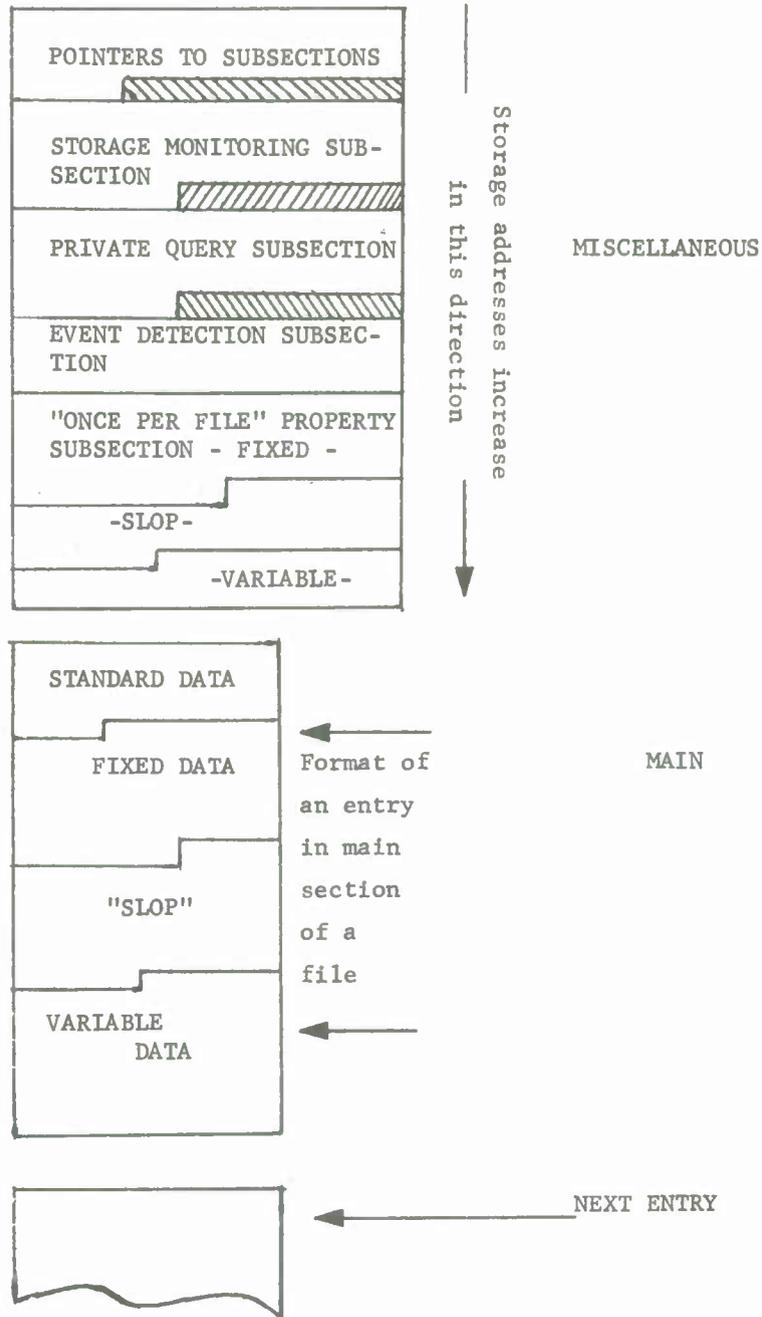


Figure 2. Format of A File

STANDARD DATA FOR AN ENTRY

0	23 24 27 28	42 43 46 47	58 59 62 63
SIZE OF OBJECT IN BITS	FLAGS	PV OF OBJECT	PV FLAGS
		DEAD SPACE BIT COUNT	FLAGS D

0	3 4	27 28 31 32	46 47 50 51	63
D FLAGS	BEGINNING OF VARIABLE DATA (RELATIVE TO END)	FLAGS	STANDARD CLASSIFICATION PV	ALTERNATE CLASSIFICATION

0	1 2	5 6 7	10 11	34 35 38 39	63
	FLAGS	0	FLAGS	LENGTH OF SLOP IN BITS	START OF FIXED DATA
			FLAGS		

- SIZE OF OBJECT The size is actually in words since every entry starts at a full word boundary. The flags are always 0.
- PV OF OBJECT PV in object roll.
- DEAD SPACE BIT COUNT* To indicate when garbage collection is needed.
- D Delete bit. If set, this entry is deleted.
- BEGINNING OF VARIABLE DATA The number of bits used for variable data.
- STANDARD CLASSIFICATION PV of security classification of data in this entry.
- ALTERNATE CLASSIFICATION* PV of security classification of data in this entry with V flag set.
- 0 * Override bit.
- LENGTH OF SLOP Number of bits remaining unused in the entry.

STANDARD DATA FOR A REPETITION

0	1	15 16 19
D	NAME PV	FLAGS

- D Delete bit for the repetition.
- NAME PV PV of name of this repetition if named repeating group.
- FLAGS Bit 17=1 if no name or if unnamed repeating group.

Figure 3. Format of Standard Data

- (a) FP = Floating Point (64 bit standard 7030 format).
Such data is guaranteed to start at a full-word boundary -- otherwise it would be declared as compressed floating point (q.v.).
- (b) CFL = Compressed Floating Point*
An VFL quantity which, when placed in the accumulator at the prescribed offset, becomes a floating point quantity.
- (c) VFL = Integer (never greater than 48 bits long).
- (d) L = Logical (or roll valued)
The values of these properties are principal values in the specified roll, i.e., a 15-bit integer.

Other data types are variable in size and must therefore be pointed to by a 64-bit pointer (see 1.6 on page 47).

- (a) NQV = Numeric Query Valued*
LQV = Logical Query Valued
The value of each of these properties is computed by retrieving and honoring an associated query. NQV and LQV always start at a full-word.
- (b) RAW = Raw data (an arbitrary string of bits).
Raw data always starts at a full-word.
- (c) NRG = Named Repeating Group
URG = Unnamed Repeating Group

1.4 REPEATING GROUPS

Repeating groups are sufficiently complicated to require an elaborate description. A repeating group may be considered as a named collection of properties. A set of values comprising one value for each property in the group is called repetition. A repeating group may have an arbitrary number of repetitions. The properties included in a repeating group may be any of the seven types (including other repeating groups). A variable data type in a repeating group will, of course, be pointed to yet again -- in this system, to access data, a sequence of pointers corresponding to the number of degrees of variability must be used.

A named repeating group has a property called 'NAME' and hence each repetition may have a name.

One repetition in a repeating group consists, then, of a fixed part which contains fixed properties and pointers to variable properties, and variable parts pointed to by the pointers. The following rules hold.

All repetitions are within the Variable Data portion of an entry. The pointer to the first repetition is in the Fixed section when the repeating group property is 'prime' level, (i.e., not itself in a repeating group). There will never be a discontinuity within the fixed section of a repetition.

1.4.1 Format of a Repetition

(a) Standard Data

Figure 3 shows the format of the standard data. If the repeating group is unnamed, the NAME value will be deleted and there will be no NAME property in the property roll.

(b) Fixed Data

The fixed data for a repetition is exactly like fixed data for an entry. The location of fixed properties in a repeating group are specified in the property roll as relative to the start of a repetition rather than an entry. As noted above, there will never be a discontinuity in the fixed data for a repetition. If a repeating group contains any floating point property, each repetition will begin at a full-word boundary.

(c) Variable Data

Variable data has the same use as in an entry.

1.5 FLAGS

Each property has a sign and three flag bits. In fixed length properties these are the last four bits. Variable length properties have flag bits in the first pointer (see 1.6 on page 46). These bits are used as follows:

- bit 0 (S) Sign bit. Always 0 in nonnumeric properties.
- bit 1 (T) If set, this value has been deleted or is nonexistent.
- bit 2 (U) *Override bit. Intended to prevent contamination of the data base during checkout.

bit 3 (V) *If set, this value has the alternate security classification found in standard data for the entry.

1.6 POINTERS TO VARIABLE DATA IN FILES

The described pointer is used for all pointed to data (viz., Repeating Groups, Query Valued*, and Raw Data Properties) both as original and continuation pointers.

Data Location (B, 25, 4) Value	Flags ØTUV	Count (bits) (BU, 18)	Location of Continuation Pointer (BU, 21)
-----------------------------------	---------------	--------------------------	---

All pointers are 64 bits long (but need not start at a word boundary). These 64 bits are divided into three fields:

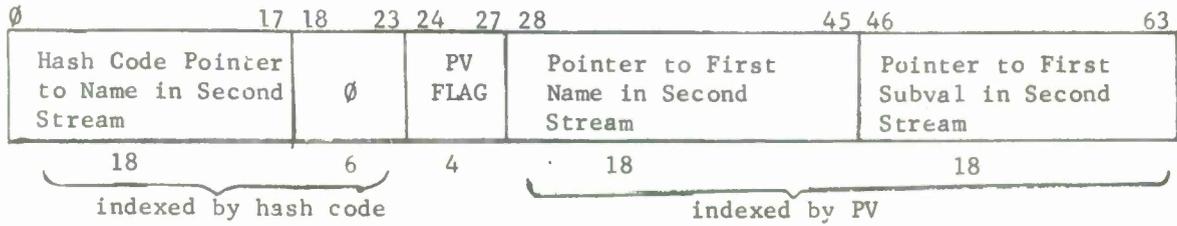
Data Location-----bits 0-21 where bit 21 = Ø
 Flags-----used in the first pointer only with
 same meaning as for other properties
 Count-----bits 25-42
 Continuation
 Pointer Location---bits 43-63

The location field points to the beginning of the data relative to the beginning of the next object; in other words, the location pointed is:

ORIGIN OF ENTRY + LENGTH OF ENTRY - VALUE OF POINTER
 LOCATION FIELD.

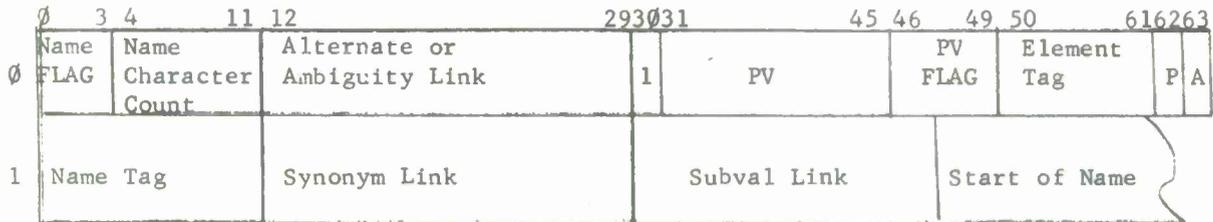
This means the values in pointers do not change when the length of the entry is increased.

FIRST STREAM FORMAT

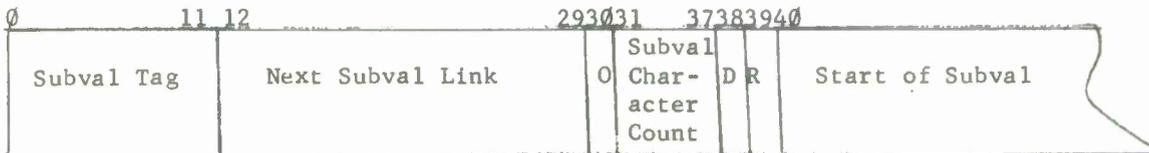


SECOND STREAM FORMAT

(A) NAME ELEMENT



(B) SUBVAL ELEMENT



- P = 1 Name is a prefix
- A = 1 Name is ambiguous
- D = 1 Subval deleted
- R = 1 Subval is recirculation

Figure 4. Format of a Standard Roll

2.1 FORMAT OF A ROLL

The format of a roll is illustrated in Figure 4.

2.1.1 Standard Roll

The standard roll consists of two streams. The first has a dual use. It may be accessed by the hash code for a name and points to the list of names in stream 2. It may also be accessed by PV and points to the location of names and subsidiary values (subvals) in stream 2. The second stream contains names and subvals. The ambiguity link chains elements together which have the same name. An element is all the data associated with one PV. The synonym link chains multiple names for the same element. The alternate link chains names which map into the same hash code.

2.1.2 Special Rolls

Frequently a few rolls have a special format and are called special rolls. A special roll has the same format as a standard roll with the addition of a third stream containing subvals arranged in order of PV, with a fixed number of words per subval. The number of words and the fields are different for each roll. This third stream is called a pointer set and all pointer sets reside in core during system operation to provide rapid access to the subval information.

DOCUMENT CONTROL DATA - R & D

(Security classification of title, body of abstract and indexing annotation must be entered when the overall report is classified)

1. ORIGINATING ACTIVITY (Corporate author)		2a. REPORT SECURITY CLASSIFICATION	
The MITRE Corporation Bedford, Massachusetts		Unclassified	
		2b. GROUP	
3. REPORT TITLE			
A DESCRIPTION OF THE INTERNAL OPERATION OF THE <u>ADAM</u> SYSTEM			
4. DESCRIPTIVE NOTES (Type of report and inclusive dates)			
N/A			
5. AUTHOR(S) (First name, middle initial, last name)			
Clapp, Judith A.			
6. REPORT DATE		7a. TOTAL NO. OF PAGES	7b. NO. OF REFS
August 1967		54	0
8a. CONTRACT OR GRANT NO.		9a. ORIGINATOR'S REPORT NUMBER(S)	
AF 19(628)-5165		ESD-TR-67-372	
b. PROJECT NO.		9b. OTHER REPORT NO(S) (Any other numbers that may be assigned this report)	
502F			
c.			
d.		MTR-276	
10. DISTRIBUTION STATEMENT			
This document has been approved for public release and sale; its distribution is unlimited.			
11. SUPPLEMENTARY NOTES		12. SPONSORING MILITARY ACTIVITY	
		Deputy for Command Systems, Computer and Display Division; Electronic Systems Division, L. G. Hanscom Field, Bedford, Massachusetts	
13. ABSTRACT			
<p>This report summarizes the internal operation of the ADAM system. It describes the organization of functions among the system routines. Appendix I lists the sizes of the primary routines. Appendix II describes the internal format of ADAM files and rolls.</p>			

KEY WORDS

LINK A		LINK B		LINK C	
ROLE	WT	ROLE	WT	ROLE	WT

COMPUTER SYSTEMS

ADAM

Internal Operation, Description

System Routines, Organization of
Functions