ESD-TR-66-644
ESTI FILE COPY

ESD-TR-66-644 MTR-268

1 of 2

ESD RECORD COPY
RETURN TO
SCIENTIFIC & TECHNICAL INFORMATION DIVISION
(ESTI), BUILDING 1211

A USER'S GUIDE TO THE ADAM SYSTEM

ESRC

ESD ACCESSION LIST
ESTI Call No. AL 58899
Copy No. 1 of 2 cys

DECEMBER 1967

ADAM Project Staff

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

AD664337

When U.S. 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.

ESD-TR-66-644

MTR-268

A USER'S GUIDE TO THE ADAM SYSTEM

DECEMBER 1967

ADAM Project Staff

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

Project 502F

Prepared by

THE MITRE CORPORATION
Bedford, Massachusetts

Contract AF19(628)-5165

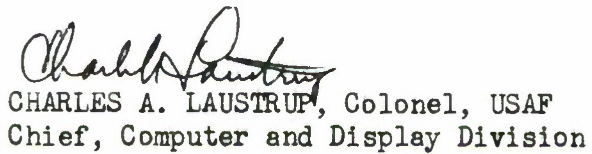
This document has been approved for public release and sale; its distribution is unlimited.

FOREWORD

The work reported in this document was performed by The MITRE Corporation, Bedford, Massachusetts, for the Deputy for Command Systems, Computer and Display Division, Electronic Systems Division, of the Air Force Systems Command under Contract AF19(628)-5165.

REVIEW AND APPROVAL

This technical report has been reviewed and is approved.



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

ABSTRACT

This report describes the kinds of capabilities available in the ADAM system and the way in which they are used. The processes for creating and maintaining a data base, specifying formats, modifying the form of the input, and specifying procedures are described. The FABLE, IFGL, and DAMSEL languages are also described.

TABLE OF CONTENTS

		Page No.
SECTION I	INTRODUCTION	1
SECTION II	FABLE	4
	BASIC QUERY STRUCTURE OF FILE PROCESSING STATEMENTS	6
	'For' Part	6
	Boolean	7
	Output Part	7
	BOOLEAN IN DETAIL	8
	Operands in Boolean Primaries	8
	Property Names	9
	Complex Boolean Primaries	11
	Action Phrases	12
	OUTPUT PART OF FILE PROCESSING STATEMENTS IN DETAIL	15
	Titles and Formats	15
	Repeating Groups	16
	New Properties	16
	ALL	16
	Null Output Files	17
	LOOPING IN FILE PROCESSING STATEMENTS	17
	Loops	17
	Global Rule	18
	'For' Clauses	18

TABLE OF CONTENTS (Continued)

		Page No.
SECTION II (Cont.)	TALLYING WITH FABLE STATEMENTS	19
	General Description	19
	Detailed Discussion	20
	USING FABLE TO EXAMINE OR MODIFY ROLLS	22
	Specifying a Roll	22
	Retrieving Contents of a Roll	23
	Modifying a Roll	24
	OPERATING ROUTINES WITH FABLE STATEMENTS	25
	Requirements for Routines Which May be Operated Through FABLE statements	25
	References to Routines in FABLE	26
	SORTING A FILE	29
SECTION III	FILE GENERATION LANGUAGE AND PROCEDURES	32
	GENERAL CHARACTERISTICS	33
	Source of Input	33
	Data Fields	33
	Names	34
	Data Modifications	34
	File Classification	34
	Logical Rolls	34
	Protection	35
	Format	35
	Legality Checks	35

TABLE OF CONTENTS (Continued)

		Page No.
SECTION III (Cont.)	INITIAL FILE GENERATION LANGUAGE DECK	36
	Preparation	36
	Notation	36
	FILE DESCRIPTION	38
	Syntax	38
	Discussion	39
	PROPERTY DESCRIPTION	43
	Logical Property Description	43
	Numeric Property Description	45
	Raw Property Description	47
	Repeating Group Property Description	48
	Sequence Check	49
	FIELD DESCRIPTION	50
	Position	50
	Length Description	53
	Set Sequence Counter Command	54
	CONVERT DESCRIPTION	55
	Syntax	55
	Discussion	55
SECTION IV	DAMSEL	59
	GENERAL CHARACTERISTICS	59
	System Names	59
	System Specifiers	60
	Variables and Type Specifiers	61
	Card Format	62
	System Defined Variables	63

TABLE OF CONTENTS (Continued)

		Page No.
SECTION IV (Cont.)	DAMSEL DECLARATIONS	66
	Routine Declaration	67
	Table Declaration	68
	Parameter Declaration	69
	Entry Declaration	69
	Variable Declaration	71
	Tail Declaration	72
	Begin Globals Declaration	72
	End Globals Declaration	73
	End Declaration	73
	DATA MANIPULATION STATEMENTS	73
	File Processing Statements	74
	Area Statements	80
	File Generation Statements	82
	DATA TRANSMISSION STATEMENTS	82
	Block Designators	83
	Fetch Statements	84
	Store Statements	85
	ASSIGNMENT STATEMENT	85
	Variables	86
	Constants	88
	Simple Assignment Statements	93
	Assignment Vectors	96
	Compound Assignment Statements	97
	Generalized Assignment Statements	98

TABLE OF CONTENTS (Continued)

		Page No.
SECTION IV (Cont.)	BRANCH STATEMENTS	98
	Unconditional Branch Statements	98
	Conditional Branch Statement	99
	Entry Branch Statement	100
	Enter Statement	100
	Exit Statement	100
	ROUTINES AND FUNCTIONS	101
	Call Statement	101
	Function Reference	103
	DO Statement	104
	DATA STATEMENTS	104
	Real Data Statement	105
	Integer Data Statement	105
	Roll Data Statement	106
	Location Data Statement	106
	Spring Data Statement	107
	Switch Data Statement	107
SECTION V	STRING SUBSTITUTION	108
	DEVICE DEPENDENCY	109
	OPERATION	109
	Scan Option	110
	Rescan Option	111
	PARAMETERS	112
	Insert Option	114
	Reinsert Option	115

TABLE OF CONTENTS (Continued)

		Page No.
SECTION V (Cont.)	CAUTIONS	115
	Messages Not Subject to Substitution	115
	Punctuation and Separation	115
	DEFINITION AND SYNTAX	117
	String Substitution Definition	117
	Removing String Substitutions	118
	SUBTABLE FILE	118
SECTION VI	OUTPUT FORMATTING	120
	BASIC PRINCIPLES	121
	An Example	121
	Which File	122
	Output Devices	122
	Printed vs Display Output	123
	Next Field	123
	Next Object	123
	Next Property	123
	CATEGORIES OF OPERATORS	124
	Print Operators	124
	Spacing Operators	124
	Display Operators	124
	Iteration-Control Operators	125
	Mode-Setting Operators	125
	Margin-Definition Operators	125
	Special Routines	126
	Miscellaneous Operators	126
	Macro-Control Operators	126

TABLE OF CONTENTS (Continued)

		Page No.
SECTION VI (Cont.)	FORMAT TYPES - COL, ROW, AND RAW	127
	FIELD DEFINITION AND FIELD OVERFLOW	128
	Deleted Names and Values	129
	Variable Width	129
	Floating Point Property Values	130
	Field Underflow	131
	Field Overflow	131
	Right-Margin Overflow	132
	MARGINS, PAGINATION, AND HEADINGS	132
	Margins	132
	Pagination	133
	Headings	134
	Page Numbers and Classification-Implied Top Margin	135
	FILE DATA	135
	Objects and Properties	135
	Standard Properties	135
	Repeating Groups	136
	Alignment of Repetitions	136
	Repeating Group Stepping	139
	MACRO AIDS	140
	Begins and End	140
	Labeling or Tagging	141
	STRAP or SMAC Code Intermixed	141

TABLE OF CONTENTS (Continued)

		Page No.
SECTION VI (Cont.)	SOME EXAMPLES	142
	Example of Column Format - SF1	142
	Example of Raw Format	145
	Example of Row Format	147
SECTION VII	WRITING ROUTINES	149
	ADAM ROUTINES	149
	STANDARD BINARY DECKS	150
	Roll Data Subdeck	150
	Routine Data Subdeck	151
	ROUTINE FILE	151
	ROUTINE FILE UPDATING (RUE)	151
	Delete Options	152
	Add Option	152
	Correct Option	152
	ROUTINE LOADING (CLOD)	152
	Fixed Routine Loading	153
	Allocatable Routine Loading	153
	Allocatable Routine Dismissal	153
	General Release	154
	SYSTEM CONVENTIONS FOR ADAM ROUTINES	154
	Conventions for Called Routine	154
	Conventions for Calling a Routine	156
	Specifying Roll Data for Standard Binary Decks	157
	References to Data	159

TABLE OF CONTENTS (Concluded)

		Page No.
SECTION VII (Cont.)	Consideration for Code Operated in Autostacked Mode	164
	SYSTEM CONVENTIONS FOR FORTRAN ROUTINES	165
	Preparation	165
	Execution	166
	Comfort List	166
	Restrictions on FORTRAN Statements	167
	Required Heading	168
	FORTRAN Calls to DAMSEL Routines	169
	DAMSEL Calls to FORTRAN Routines	170
	Library Routines	172
APPENDIX I	SYNTAX OF FABLE	173
APPENDIX II	TWO SAMPLE FILES USED IN THE EXAMPLES	179
APPENDIX III	AN IFGL FILE DESCRIPTION EXAMPLE	180
APPENDIX IV	SOME EXAMPLES	184
APPENDIX V	MESSAGES TO USER FROM STRING SUBSTITUTION	190
APPENDIX VI	FORMATTING OPERATORS	192
APPENDIX VII	PAGE SIZES FOR VARIOUS DEVICES	220
APPENDIX VIII	OUTPUT FORMAT BY SPECIAL ROUTINES – DO AND VC OPERATORS	222
APPENDIX IX	LIGHTPENCIL INPUT STREAM	232
APPENDIX X	DESIGN FEATURES NOT IMPLEMENTED	235

SECTION I

INTRODUCTION

Since the first large command and management systems were built, there has been an obvious need for a constantly improving capability to plan, design, and evaluate command and management systems. This capability must include the best available methods for generating alternative system designs and precise techniques for rapidly evaluating existing, prototype, or proposed system designs or design characteristics.

System designers must be provided with both improved operating information processing concepts and improved techniques for use in design and evaluation. These tools and techniques must be capable of rapidly reflecting the latest technology, experimental proposals, and designs.

Similarly, systems to be produced for the field are requiring an ever-increasing degree of flexibility not provided by conventional programming technology. Lead times and reprogramming costs frequently exceed acceptable limits. New techniques must be devised or refined and applied to these problems.

The objectives of the ADAM project were to use, develop, and evaluate advanced information-processing techniques for use in the system design and implementation processes, and to make the means for realizing these techniques in an experimental setting rapidly available.

In late 1962, The MITRE Corporation began designing the ADAM System* for the Electronic Systems Division, Air Force Systems Command.

---

* The ADAM System is described in T. L. Connors, ADAM, A Generalized Data Management System, The MITRE Corporation, MTP-29, April 1966. This report should be read to provide background information.

The system became operational in early 1965. By 1966, ADAM was fully developed to function as a tool for designers of data management systems during the conceptual, design, and evaluation phases. ADAM allows a system designer to build and operate a functional prototype of a proposed system in a laboratory environment.

The ADAM System operates in real time with on-line inputs from a number of users and a variety of devices, as well as off-line batched inputs. It is implemented on the IBM 7030 in the Systems Design Laboratory.

ADAM is an example of a generalized data management system in which as many user and system functions as possible were generalized, including the following:

(1) translation of input messages independent of the language

(2) data base generation and maintenance independent of the form or actual contents of the data

(3) creation, formatting, and presentation of outputs independent of the particular formats or data

(4) dynamic allocation of computer storage resources

(5) input-output routing and handling for a variety of devices.

A user builds his particular system by specifying what the generalized ADAM routines should do. The system description becomes a part of the data base along with the data itself and hence the ADAM System is modified by use of its file maintenance capability. Among the things which a user may specify are:

(1) the language or languages he wishes to use

(2) his data base structure

(3) the formats of his reports

(4) special procedures specific to his application.

ADAM has more than one kind of user: The system designer uses ADAM to build a prototype of all or part of his own system. The designer may have programmers who program his specific calculations, and ultimate users who communicate with the prototype itself. This report is directed primarily to the system designer, but may be of interest to user programmers, ultimate users, or those responsible for creating large system programs.

SECTION II

FABLE

FABLE is an input language available in the ADAM system. Messages containing one or more FABLE statements may be entered into the ADAM system for processing via any on-line device or off-line through card input.

There are FABLE statements for the following kinds of operations:

(1) Execution of a routine

(2) File processing

This includes creation of a file from existing files, deletion of files, addition to file data, modification of file data, sorting of entries and repetitions, tallying and summing of values, and retrieval and presentation of data.

Conditional phrases permit the user to select the data within a file or files which qualify for the specified file processing. A single FABLE statement may perform several file processing operations and reference data in several files, but only one file may be modified and one file created. The user may specify the format and output devices for the presentation of retrieved data.

(3) Roll processing

Information in rolls may be retrieved and modified.

FABLE is the only available facility in the ADAM system for the on-line specification of procedures other than file generation. The ability to specify string substitutions (see Section V) enables a user to modify the form

of the language in which he communicates with the ADAM system, but the result of string substitution must be valid statements in FABLE or IFGL, the file generation language.

The ADAM system processes one FABLE message at a time. First, the message is edited to remove control characters such as carriage returns and backspaces. Then, it is edited to remove superfluous spaces and to introduce a single space between each alphanumeric sequence of characters and punctuation and between successive punctuation.

The ADAM translator scans the resulting message to analyze the syntax, and, if it is sytactically correct, it is transformed into a sequence of operations, along with values from the message, which are then interpretively executed.

If a syntactic error is found, processing of the message is terminated and the user receives an error message with some indication of the kind of error, showing the remainder of the message as it appears after string substitution, beginning with the rightmost word scanned before the syntax analysis failed.

The syntax of FABLE statements is given in Appendix I. Judicious use of this information should permit a user to construct statements of the correct forms. The meaning of those forms and what can be accomplished with FABLE messages is described in the succeeding paragraphs of this Section. The parts of the syntax which relate to any discussion are referenced by the names assigned in Appendix I.

The examples in this section reference two files which are described in Appendix II. Other examples of the use of FABLE may be found in the following documents:

C. Baum and L. Gorsch, Editors, Proceedings of the Second Symposium on Computer-Centered Data Base Systems, SDC, TM-2624/100/00, 1 December 1965, pp 3-87 to 3-121.

O. Beebe, B. Char, J. Penney, Implementation of ADAM-AFLC Experiment - Phase I, MITRE, MTR-109, 24 January 1966.

O. Beebe, J. Penney, Implementation of ADAM-AFLC Experiment - Phase II, MITRE, MTR-262, 15 July 1966.

BASIC QUERY STRUCTURE OF FILE PROCESSING STATEMENTS

FABLE queries generally consist of three parts: the 'for' part, which names the file or objects to query; the 'Boolean', which performs actions and selects objects and repetitions for output; and the 'output' part, which names properties to be printed. Either the Boolean or the output part, but not both, may be omitted.

Example:

FOR AIRFIELD . IF ALT EQ LOGAN, PRINT LAT, LONG.

'For' Part

The 'for' part may name a file, a file and an object, or a file and a list of objects. The file is called for 'for file'.

Example:

FOR AIRFIELD DOW.
FOR AIRFIELD DOW, REMY.

Boolean

Boolean Primaries

Booleans are constructed from basic components called Boolean primaries (boolprm in the syntax in Appendix I). The simplest form of a Boolean primary is property name, a relation, and a value, e.g.,

LAT EQ 70

The permissible relations are defined in the relation phrase of the syntax, in Appendix I. Any relation may be preceded by NOT.

And, Or, and ( )

The Boolean is made up of Boolean primaries connected by AND and OR, with AND taking precedence.

LAT GR 70 AND ALT EQ LOGAN OR CITY EQ BOSTON

The grouping of the Boolean primaries may be altered by parentheses to any number of levels. A series of Boolean primaries in parentheses is also a Boolean primary.

LAT GR 70 AND (ALT EQ LOGAN OR CITY EQ BOSTON)

Not

A Boolean primary may be negated by preceding it with NOT.

NOT LAT GR 70 AND NOT (ALT EQ LOGAN OR CITY EQ BOSTON)

Output Part

Output File

The output from a FABLE query is placed in a new file with the objects and repetitions taking the same names that they had in the 'for file'. The output file may be produced on an output device, or saved, or both. The device is indicated by OUTPUT followed by device names, or TYPE, DISPLAY, or PRINT.

Examples:

OUTPUT P1 D1 ALT.

TYPE ALT, CITY.

Saving Output File

The output file may be saved by ending the output part with NAME and the name to be given to the new file, e. g.,

PRINT ALT, CITY. NAME NEWFILE.

To save the file with no visible output use SAVE with NAME.

SAVE ALT, CITY. NAME NEWFILE.

Property List

The properties in the output file are named in the property list. Properties in a repeating group must be grouped together.

ALT, LENGTH, WIDTH, COLOR, NUMBER

BOOLEAN IN DETAIL

Operands in Boolean Primaries

Matching Operands

The operands on each side of a relation must be of the same type, either logical or arithmetic; the operands may not be raw.

Logical Operands

Only EQ or NOT EQ, or their equivalents, may be used with logical operands. The operands must use the same roll. The left-hand operand may be either a property name or a logical function; if the left-hand operand is a property name, the right-hand operand may be a property name, a value, or a logical function.

Examples:

ALT EQ OBJECT NAME

COLOR EQ GREEN

COLOR NOT EQ LOGICFN (OBJECT NAME, ALT)

If the left-hand operand is a function, the right-hand operand may be a property name or a function.

Arithmetic Operands

Each operand may be an arithmetic expression. An arithmetic expression is made of property names, functions, and numbers. These may be connected by +, -, *, and / , with * and / taking precedence. Again, parentheses may be used to alter the grouping to any number of levels.

Example:

LENGTH GR LONG * (LAT + EXP(LAT) - 7.3) /6

Property Names

Cross File Reference

Normally property names used in a Boolean primary are from the 'for file'. In FABLE there are four ways to reference properties in files other than the 'for file'. These cross file references may be used in place of property names in Boolean primaries:

Another File. Using a file name and a property name causes the Boolean to be evaluated against each object in the other file.

FOR AIRFIELD . IF LENGTH EQ CITY LAT, PRINT LENGTH.

Here, each LENGTH is checked against each LAT in the CITY File.

Another Object in the 'For File'. To refer to another object in the 'for file', use the object name and the property name.

FOR AIRFIELD . IF ALT = LOGAN ALT, PRINT LAT, LONG.

An Object in Another File. To refer to an object in another file, use the file name, the object name, and a property.

FOR AIRFIELD . IF ALT GR CITY BOSTON LAT, PRINT ALT.

An Indirect Object in Another File. When object names of another file are the values of a property (the indirect object) use the other file name, the indirect object in parentheses, and a property name in the other file to refer to data from the selected object. This is called indirect object cross file reference.

CITY (CITY) POPU

AIRFIELD (ALT) CITY

Indirect object cross file reference may be compounded any number of times.

CITY (AIRFIELD (ALT) CITY) POPU

CITY (AIRFIELD LOGAN CITY) POPU

Ambiguous Property Names

ADAM allows files with ambiguous property and repeating group names. Ambiguous property names must be qualified by enough repeating group names to resolve the ambiguity. Unambiguous names may also be qualified for ease of reading.

RUNWAY NAME

LIGHTS NAME

RUNWAY LIGHTS COLOR

COLOR

RUNWAY COLOR

Properties in Named Repeating Groups

To deal with only certain repetitions in a named repeating group, qualify the properties in those repetitions by the group name and the repetition name or names.

RUNWAY 33L, 4R LENGTH

RUNWAY 33L, 4R LIGHTS APPROACH COLOR

Complex Boolean Primaries

Null, Else, Until

NULL and a property name (logical or numeric) in parentheses is a Boolean primary which is true when the property is deleted.

NULL (ALT)

ELSE is a Boolean primary that is always true. The output of a query may be limited by ending the Boolean with UNTIL and an integer.

FOR AIRFIELD . ELSE UNTIL 4, PRINT ALT.

This query prints only the first four ALT's. UNTIL counts the number of repetitions or objects that qualify and stops output when the count is exceeded, although the querying goes unhindered to completion.

Double Boolean Primaries

Two relations may be connected by AND or OR with a single left-hand operand.

LAT LS 40 AND GR 10

Compound Boolean Primaries

The operands on either side of a relation may be compound. Multiple operands separated by commas and ALSO and grouped by parentheses are permitted. A comma means OR, and ALSO means AND.

CITY EQ BOSTON, HARTFORD, NEW YORK

LAT ALSO LONG GR 7 AND LS 13

Boolean primaries that are compound on both sides are equivalent to a series of Boolean primaries that are compound on the left only, using the right-hand operands and separated by the right-hand operators.

LAT, LONG EQ 7 ALSO 3

is equivalent to

LAT, LONG EQ 7 AND LAT, LONG EQ 3.

Any and All

Ordinarily, in FABLE queries, the Boolean must be satisfied completely within the same repetition. Sometimes it is desirable to see if one repetition satisfies one condition and another repetition satisfies a different condition (ANY), or to see if all repetitions satisfy the same condition (ALL). (See Looping in File Processing for further discussion.)

ANY LENGTH GR 10000 AND ANY LENGTH LS 7000

ALL (LENGTH GR 10000 AND WIDTH GR 100)

ANY and ALL also work with cross file reference.

ANY (LAT EQ CITY POPU) AND ANY LONG EQ CITY POPU*2

Action Phrases

Position in Query

Action phrases alter files and operate routines. An action phrase may appear after a Boolean primary or in place of a Boolean primary and the phrase is equivalent to ELSE followed by the phrase. Several action phrases may be joined by commas.

FOR AIRFIELD . IF COLOR EQ GREEN CHANGE COLOR TO RED OR DELETE REPETITION RUNWAY.

FOR AIRFIELD . IF (COLOR EQ RED OR NUMBER EQ 3) CHANGE
    COLOR TO GREEN.

Operation of Action Phrases

An action phrase is performed whenever the Boolean primary preceding it is evaluated true. FABLE evaluates as few Boolean primaries as possible in determining the truth of a Boolean expression. When one Boolean primary in a series connected by AND's is evaluated false, the rest are not evaluated and the whole series is false. When one Boolean primary in a series connected by OR's is evaluated true, the rest are not evaluated and the whole series is true.

Types of Action Phrases

Change. A property may be changed to NULL (which deletes it), a value, the value of another property of the same type, or an arithmetic expression where appropriate. Raw properties may be changed to the values of other raw properties only. More than one change may be specified, where each is separated from the next by a comma and CHANGE is not repeated.

CHANGE COLOR TO GREEN, ALT TO NULL

CHANGE LAT TO LAT +7, LONG TO CITY BOSTON LONG.

ADD Repetition. Repetitions may be added to groups by phrases such as

ADD REPETITION RUNWAY (NAME = GEORGE, LENGTH = 7,
    WIDTH = DOW RUNWAY 4R WIDTH)

The repetition is always added following the last repetition in the group. More repetitions may be added, each separated by two commas, and a repetition may be added within a repetition.

ADD REPETITION RUNWAY (NAME = GEORGE, , NAME = MIKE,
    LIGHTS (COLOR = GREEN))

Delete Repetition or Group. Repetitions or groups may be deleted by DELETE REPETITION or DELETE GROUP followed by the group name with repetition names if desired.

DELETE REPETITION RUNWAY 4R, 33L

DELETE GROUP RUNWAY LIGHTS.

Delete Object. This will delete the current object.

Add Object. The ADD OBJECT phrase is similar to ADD REPETITION. The object is added to the end of the file.

FOR AIRFIELD . ADD OBJECT OBJECT NAME = CITY
OBJECT NAME, LAT = 7, RUNWAY (NAME = GEORGE,,
NAME = MIKE)

There are four restrictions on the ADD OBJECT alter phrase:

(1) There may be no output part in the statement.

(2) There may be no other kinds of action phrases, except DO routine.

(3) No property may be fetched from the file to which data is being added.

(4) If object name is set to a property value, the property must use the object roll of its file.

Do Routine. A routine may be operated by DO and a routine name – DO routcall, in the syntax – and its parameters.

DO F(ALT).

Cross File ALTER

While querying one file it is possible to alter objects in another file by action phrases. At the end of the 'for' part of the statement add ALTER, a file name (called the alter file), and either an object name or an indirect object name. (See An Indirect Object in Another File, p. 10.)

FOR AIRFIELD ALTER CITY BOSTON.

FOR AIRFIELD ALTER CITY (CITY).

The values of the alter file are changed and used for the output file; i. e., properties and repetitions that are changed, added, or deleted are in the alter file, and the proplist or ALL in the output part of the statement refers to the alter file. All other property names in the Boolean are assumed to be in the 'for file' unless explicitly specified as another file.

For example:

FOR AIRFIELD ALTER CITY (CITY). IF ANY (LENGTH GR CITY (CITY) L$\underset{*}{\text{A}}$T)

CHANGE L$\underset{*}{\text{A}}$T TO CITY (CITY) L$\underset{*}{\text{A}}$T + LAT,

PRINT L$\underset{*}{\text{A}}$T, 'OLD LAT' = L$\underset{*}{\text{A}}$T - AIRFIELD LAT.

LAT's marked with an * underneath are in the City (Alter) file. Other LAT's are in the Airfield ('for') file. This query illustrates the places where cross file reference must be made and where the file of a property reference will be inferred.

The cross file ALTER phrase for the add object action names only the file.

FOR AIRFIELD ALTER CITY.

OUTPUT PART OF FILE PROCESSING STATEMENTS IN DETAIL

Titles and Formats

Formats and titles may be specified by FORMAT and a format name, and TITLE followed by the title enclosed in primes.

DISPLAY FORMAT F1 TITLE 'THIS IS A TITLE' ALT , COLOR.

Repeating Groups

In the property list, the group name may be placed before the properties in a group and must be so placed if there is ambiguity.

LAT, ALT, RUNWAY NAME, LENGTH, LIGHTS COLOR.

LAT, ALT, LIGHTS COLOR.

If only the group name is specified, all properties from the group will be put into the output file.

New Properties

A new property may be introduced into the output file by naming the new property enclosed in primes and following it by an equals sign and then an arithmetic expression or a property name,

ALT, 'SUM' = LAT + LONG + 7, 'POP' = CITY BOSTON POPU.

New properties in groups must have OF and the group name after the property name and they must follow either the group name or a property in the group.

RUNWAY 'AREA' OF RUNWAY = LENGTH*WIDTH.

LENGTH, 'AREA' OF RUNWAY = LENGTH*WIDTH.

All

Instead of the property list, one may write ALL, which will transfer a complete object into the new file if the Boolean is true for the object, or for some repetition in the object, or if it is true for any other reason. The object gets transferred every time the Boolean is true. If the object is being altered by action phrases in the Boolean, the object in the output file will reflect the values of the object the last time the complete Boolean was true.

Null Output Files

The output file is produced even though there are no objects that satisfy the Boolean and hence no data in the output file. This makes an easy way to build an empty file with almost any structure desired.

```
FOR AIRFIELD LOGAN.  NOT ELSE.  SAVE 'X' = 0, 'Y' = 0,
     RUNWAY 'Z' OF RUNWAY = CITY OBJECT NAME.  NAME NEW.
```

LOOPING IN FILE PROCESSING STATEMENTS

Loops

In evaluating a query, FABLE steps through the files, groups, and objects in nested loops. First, each is opened in the order of appearance in the query. Then, the Boolean is evaluated using the values in the currently open objects and repetitions. If the Boolean is true, the properties named in the output part of the query are transferred to the output file. Then, true or false, the innermost loop is stepped and the process is repeated until all the loops are completely stepped through.

In FABLE, ANY, ALL, ADD REPETITION, ADD OBJECT, and the output part are local looping phrases (LLP's). When the evaluation of a Boolean comes to an LLP, a set of nested loops is set up for those files, objects, and groups mentioned within the LLP. Then, the loops are stepped just as in the main Boolean. In an ANY phrase, the stepping stops after the first true instance, and the ANY phrase is true. If the ANY phrase is completely stepped through without any true instance, it is false.

In an ALL phrase it is just the opposite: The stepping stops after the first false instance, and the ALL phrase is false. If the ALL phrase is completely stepped through without a false instance, the ALL phrase is true.

Global Rule

Not all files, objects, and groups mentioned in a local looping phrase (LLP) are stepped through at that point; because of this, they are called global to the LLP. A file, object, or group is global to an LLP if it is mentioned to the left of the LLP and is outside of all LLP's which do not contain the LLP in question.

Files, objects, and groups previously mentioned to the left carry into ANY's and ALL's; but a file, object, or group that is not global to an ANY or ALL does not carry beyond the end of it.

For example:

FOR X. X ANY(Y) X ANY (X ANY(Z)Y)Y.

an object, file, or group is global to the ANY containing Z, if it is mentioned at a point marked by an X, and it is not global if it is mentioned at a place marked by Y. An example of the usefulness of the global rule is

FOR AIRFIELD. IF RUNWAY LENGTH GR 1000
AND ANY(LENGTH EQ CITY LAT) AND ANY (WIDTH EQ CITY LONG),
PRINT LAT, LONG, RUNWAY.

Here, RUNWAY is global to both ANY's, and each ANY refers to the same runway. The effect of the query is to find runways whose length is equal to some CITY LAT and whose WIDTH is equal to some CITY LONG. IF RUNWAY LENGTH GR 1000 were not there, the effect would be to find airfields with one runway equal to some CITY LAT and another – or the same runway – equal to some CITY LONG.

For Clauses

When it is necessary to mention a file, object, or group, and the use of a Boolean primary is inconvenient, 'for' clauses may be used: FOR is followed by one or more files, a file and objects, indirect objects, objects in

the 'for file' or groups in the 'for file' (see forcl in the syntax in Appendix I). The objects and files may be followed by groups.

FOR CITY, CITY BOSTON, CITY(CITY), DOW RUNWAY, REMY LIGHT, LOGAN RUNWAY LIGHT, RUNWAY.

'For' clauses may appear before ANY's and ALL's and before Boolean primaries. The 'for' clause is used to cause looping when there is no mention of the file, object, or group in the Boolean or to cause a name to become global.

FOR AIRFIELD, FOR RUNWAY. ANY (LENGTH EQ CITY LAT) AND
    ANY (WIDTH EQ CITY LONG), PRINT ALL.

TALLYING WITH FABLE STATEMENTS

General Description

FABLE contains a type of statement that permits a tally on one or two values which may be logical type property values or arithmetic expressions. If a value is arithmetic, ranges for the tally must be specified. For logical properties, the first 25 unique values found in the file data being tallied are used as ranges. If two values are used to tally, a two-dimensional matrix is produced with a 'tally value' for each combination of the two range specifications.

The tally values may be counts (an increment of one for each time a particular tally range is found) or else the user may specify an arithmetic expression by which the appropriate tally value is incremented. In the latter case, a total is obtained by tally range.

Examples:

FOR CITY. TALLY FOR STATE. TYPE TALLY.

will count the instances of the first 25 states found in the CITY file.

FOR CITY. TALLY FOR STATE. IF STATE EQ MASS, NEW YORK, MAINE, PRINT FORMAT TALLY TALLY.

will count the number of cities in the states of Mass., New York, and Maine.

FOR CITY. TALLY FOR STATE AND POPU/1000, LS 10, 20, 30. TYPE FORMAT TALLY TALLY.

will tally POPU in ten thousands up to 30,000 for each of the first 25 states encountered in the data.

Typical output might be:

	10	20	30
MASS	5	7	15
CALIF	3	6	12
KANSAS	1	3	7
⋮			

The message

FOR CITY. TALLY FOR STATE. PRINT FORMAT TALLY TALLY POPU.

will sum the population values by state for the first 25 states.

Detailed Discussion

The tally statement (as defined in the syntax in Appendix I) consists of five parts:

for

tally

Boolean and/or action

output

tally increment

The 'for' phrase initially selects the file and, optionally, the objects from which the logical or numeric property values will be fetched.

The "Boolean" phrase, if used, will further qualify and select objects and property values for tallying. Actions may also be named within this phrase. Note that the Boolean phrase must be used if selection is desired for logical value tallies, since value qualification is not allowed within the tally phrase itself.

The tally phrase defines the arithmetic expression or logical property to be tallied. If a two-way tally is desired, the phrase will be compounded, i.e., the tally definitions will be joined by the literal 'AND'. The syntax requires that each arithmetic expression tally be followed by a range specification of the form:

a relation and a list of ranges, e.g.,

EQ 10, 20, 30

or

a relation, a $limit_x$, an increment, and a $limit_y$, e.g.,

LS 10$10$30

In the latter specification, the tally routine uses the increment to compute a list of ranges from $limit_x$ to $limit_y$. The maximum allowable number of ranges in either specification is 25. Any list of ranges exceeding this number will be truncated at 25, and, consequently, those ranges will be lost.

Only one relation (EQ, LS, GR, GQ, or LQ) per specification is allowed and, therefore, applies to each range in the list. The ranges are sorted in descending order if the relation is GR or GQ and in ascending order if the relation is LS or LQ. Tests are applied in those orders and the first test results equal to true (if any) determines the tally range of a value.

The output phrase directs the tally output file to be typed, displayed, printed, or saved. The file may be named, in which case, it is entered in the system file roll and available for additional processing. An appropriate

output format (TALLY) may be requested and, although optional, is more desirable for printing a tally file than the standard ADAM output format.

The "tally increment" option allows the user to tally for each qualification by any valid arithmetic expression. If an arithmetic expression is not present, the implied increment is 1.

The TALLY syntax is illustrated in Appendix I.

USING FABLE TO EXAMINE OR MODIFY ROLLS

Specifying A Roll

A roll may be modified or its contents may be output by using FABLE statements in which the user specifies the roll. The syntax for identifying a roll is slightly different in each type of statement but allows for the types of specification discussed in this subsection.

By Name

Some rolls have names; if the user knows the name, it may be specified, e.g.,

PRINT ELEMENT NAMES OF COLOR.

By Property

A user may specify the roll used by a particular property by specifying the file name and property name, e.g.,

PRINT LOGICAL VALUES OF AIRFIELD ALT.

FOR AIRFIELD. ADD VALUE KENNEDY TO ALT.

Property and Object Rolls

Property and object rolls do not have individual names and, therefore, must be identified by naming the file and indicating PROPERTY or OBJECT.

The property roll contains the names and descriptions of properties. The object roll contains names and locations of entries.

Examples:

PRINT PROPERTY NAMES OF AIRFIELD.

ADD SYNONYM BOS FOR OBJECT NAME CITY BOSTON.

Retrieving Contents of a Roll

FABLE allows a user to retrieve the names in a roll. These may be typed, displayed, etc., on any specified output device. The format of the output is not variable. All of the names including synonyms are printed from the specified roll with the corresponding principal value (PV) for each name. If the name is a prefix it will be indicated. It is important to realize that a query of the form

PRINT LOGICAL VALUES OF AIRFIELD ALT.

uses the 'AIRFIELD ALT' to identify the roll and print the names in it, and if that roll is also used for another property, its values will be printed as well. When the roll is used by only one property, the names in the roll constitute a list of the unique values of the property.

In addition to the property names, a FABLE user may retrieve additional information about the properties in a file with a message of the form:

PRINT PROPERTY ROLL CONTENTS OF FN.

The names and PV's of the properties and the type of each property are printed in a format which shows the file structure by indenting the properties within a repeating group.

To retrieve other values from a roll, roll dumping routines exist that may be executed by using FABLE (e.g., RDMP).

Modifying A Roll

New names may be added to a roll which is associated with a property of logical type, e.g.,

FOR CITY . ADD VALUES ALASKA, HAWAII TO STATE.

Such additions are necessary in order to make those names legal values that may then be referred to in queries, e.g.,

FOR CITY. IF STATE EQ ALASKA, PRINT ALL.

cannot be translated unless ALASKA is in the roll associated with STATE.

Another form of modification or rolls that is possible through FABLE is the addition and removal of synonyms. Synonyms are additional names associated with one element in a roll. By adding synonyms, the user may reference a logical value, a property name, an object name, etc., by more than one name.

For example,

ADD SYNONYM BOS FOR OBJECT NAME CITY BOSTON.

may then be followed by queries such as

FOR CITY BOS. PRINT POPU.

or

FOR CITY BOSTON. PRINT POPU.

A third form of roll modification involves renaming elements of a roll. Renaming means replacing the original name with a new name, i.e., replacing the name corresponding to a value in the file data with a new name. This has many uses. For example, a new file may be created from the contents of the AIRFIELD file and given the name AF. Then, a FABLE message may be used to delete the AIRFIELD file and rename AF as AIRFIELD. In that way the structure of the AIRFIELD file can be modified through FABLE.

Another use of RENAME is to change an object name. The usual query

FOR AIRFIELD IDLEWILD . CHANGE OBJECT NAME TO JFK.

will not work because OBJECT NAME is file protected. However,

RENAME OBJECT AIRFIELD IDLEWILD AS JFK.

will accomplish the change and will also make all references to IDLEWILD in other logical property values, e.g., ALT, also use JFK instead of IDLEWILD. This change is made in the roll and no modification of file data is necessary.

If RENAME is used to change a logical value, e.g.,

RENAME LOGICAL VALUE GREEN OF AIRFIELD RUNWAY LIGHTS TO BLUE.

all values in the file that had the value GREEN in the roll used by RUNWAY LIGHTS will automatically be changed to BLUE without modification of the file data.

OPERATING ROUTINES WITH FABLE STATEMENTS

Requirements for Routines Which May Be Operated through FABLE Statements

A routine may be operated within a FABLE statement provided it is stored as an entry in the routine file and its name and/or entry point names are listed in the associated ROUT and COMP rolls. The RUE routine should be used to automatically update these rolls and the routine file. A routine must be compiled with the necessary information about its name and/or entry point names and also a description of its parameters, if any. DAMSEL routines should use DAMSEL statements that contain this information. For other routines, macros exist to generate the routine description in the proper format.

FABLE will accept routines whose input parameters are any of the following types:

REAL, INTEGER, ROLL and STRING.

These types are described more completely in Section IV. Briefly, the REAL and INTEGER types are for numeric values, the ROLL type is for values from a specified roll, and the STRING type is for a string of characters.

References to Routines in Fable

Notation

The syntax of a reference to a routine is shown in the syntactic phrase 'routcall' (see Appendix I). The following points should be noted.

If a routine has no input parameters it must be written as RT ( ). If a routine has input parameters, the number of input values specified must be less than or equal to the number of inputs the routine expects. No output parameters are specified in the 'routcall'.

If the parameter type is REAL or integer, the input value may be an 'ae'. If it is a ROLL-type parameter it may be 'scpn'* or RN. If RN is used, the value is a name in the specified roll. If 'scpn' is used, it may be a logical type property that uses the roll specified in the input parameter description; or, if the PV of the specified roll is zero, it may be a logical property using any roll. In addition, an 'scpn' may be another 'routcall'. If the type is STRING the value is expressed as RV and hence must be enclosed in primes. The following are examples of parameters:

REAL: 1.2 LAT AIRFIELD(ALT)LAT LAT*100 + 50

INTEGER: 100 5*POPU

*See the FABLE syntax in Appendix I for scpn, ae, RN and RV definitions.

ROLL: RED LOGAN are examples of RN
ALT AIRFIELD ALT are examples of scpn

STRING: 'THIS IS A TEST'

Unconditional Routine Operation

The simplest FABLE statement to execute a routine has the following syntax:

DO routcall.

No reference is made to a file. Values of input parameters may be 'ae', provided the arithmetic expression does not contain any references to file data. For ROLL type parameters 'scpn' may not be used, but RN is legal.

For example:

DO REPCO (CITY, Ø)

is permissable because CITY is a RN in the roll of file names (FILES ROLL). However,

DO FCN(CITY STATE).

where CITY is a file and STATE is a logical property and the input to FCN is ROLL type, is not legal because CITY STATE is syntactically an 'scpn'.

Note that the referenced routine should not have any output parameters.

A second method of unconditional execution of a routine is the FABLE statement of the form:

for boolcl.

where the 'boolcl' consists of a 'routcall', e.g.,

FOR CITY. DO FCN(POPU).

The routine FCN will be operated once per entry in the CITY file and will be given the value of POPU for each entry.

In this form of statement the routine may have any of the legal syntactic forms for its parameters. The routine is executed once for every instance of data in entries specified by the 'for' phrase and any repetitions referenced by the parameters. For example, in a CITY file with one entry per city and a repeating group called SCHOOLS containing NUMBER OF PUPILS, the following query

FOR CITY. DO FCN (NUMBER OF PUPILS).

would operate FCN for every entry and every repetition of SCHOOLS in each entry. In this case, as in the first, the referenced routine may not have declared outputs.

Conditional Routine Operation

Among the legal actions which may be associated with a Boolean term is the phrase:

DO routcall

where routcall is defined as a routine name and its parameters. This is shown as a form of 'altfz' in the syntax of FABLE in Appendix I. The routine parameters may be any legal form. The routine is executed whenever the preceding Boolean term is true. Inputs to the routine that are file data will come from the qualifying entry and/or repetition.

For example,

FOR CITY. IF POPU GR 10000 DO FCN(POPU).

The routine may not have output parameters.

Routines in Arithmetic Expressions

A routine may be used as an operand in an arithmetic expression wherever such expressions are legal in FABLE. The only restriction on the routine is that it must have one output which will be treated as a floating point number.

For example,

FOR CITY. CHANGE POPU TO FCN(POPU).

FOR CITY. IF POPU GR FCN(POPU) CHANGE POPU TO Ø.

Routines with Logical Output

The syntactic phrase 'scpn' may be a 'routcall', i.e., a routine call. In this case, the routine must have an output which is a PV stored as an integer. Examples of the use of a routine with logical output are the following:

FOR AIRFIELD. IF ALT EQ F(CITY) CHANGE ALT TO NULL.

FOR AIRFIELD. CHANGE ALT TO F(CITY).

SORTING A FILE

Using FABLE, it is possible to sort a file so that it is ordered on a maximum of 20 property values within the file. For a single SORT, the values may be for prime level properties; i.e., each property has one value per entry, or may be for properties in one group. The reordered file may be the original file or a new file with a user-specified name. Entries are reordered on prime level properties. Sorting on property values within a repeating group reorders the repetitions within each entry.

The SORT capability is available in two types of FABLE statements, the file processing statement which can retrieve and modify as well as sort, and a simple SORT statement. In the first case, the output file is sorted before it is saved or output. In the second case, the file, or a list of entries if the sort is in a group, to be sorted is specified. In either case, the properties are named and an ordering (ascending or descending) is specified for each. If no ordering is specified, ascending order is assumed.

For alphabetical sorts, the collating sequence is:

ascending ⟶

blank + $ = * (/) . ; , A . . . Z Ø . . . 9 -

⟵ decending

Any other character is collated as a / . Null values are last in the sort sequence of both alphabetic and numeric values. For numeric values, ascending order is the order of increasing numerical value, starting with the largest negative number and ending with the largest positive number.

If more than one property is specified as a sort key, ordering is performed on the first property-ordering specified, and within the ordered set all those having the same values are ordered on the second property-ordering pair, etc.

Examples:

FOR CITY IF POPU GR 10000, PRINT POPU. SORT ON POPU

This sorts the entries in the output file in increasing order of POPU before the file is printed.

SORT CITY ON STATE ASCEN, POPU DESCN.

This sorts the CITY file by state and for all cities in each state by population. In this case the CITY file itself is reordered. Due to the implementation of SORT in the ADAM system, a new object roll is created for the CITY file and the old roll is deleted. For this reason any logical properties which used the object roll of the original file cannot be referenced, e.g., CITY in the AIRFIELD file.

SORT AIRFIELD RUNWAY LIGHTS ON INTENSITY DESCN, COLOR.
NAME AIRSORT.

This will sort the LIGHTS group in descending order by intensity and lights with the same intensity in ascending alphabetical order by COLOR. The AIRFIELD file will be unmodified. All sorting will be done in a new file called AIRSORT.

SECTION III

FILE GENERATION LANGUAGE AND PROCEDURES

The ADAM file generation package generates ADAM data files on disk. The files and the associated property and object rolls can be saved on tape for later restoration within the system.

An ADAM file is a collection of information about a similar set of entities, called objects. The various pieces of information which pertain to an object are called properties. The set of all property values for an object is called an entry. The data falls into two general classes of property types: fixed length and variable length. Fixed length property types have floating-point, integer, and decimal values. Variable length property types, using as much space as is necessary in that particular object, are logical valued and raw valued. A repeating group, a property without any value, is a collection of properties which may be any of the types mentioned above, including other repeating groups. A repeating group may have an arbitrary number of repetitions, i.e., sets of values comprising one value for each group property.

Associated with each file, but physically separated from it, are two rolls, a property roll and an object roll, which serve as a dictionary-directory for the file.

An ADAM file can be generated by using the Initial File Generation Language (IFGL), the only 'built-in' capability for file generation from card or tape input data. File generation may also be accomplished by using FABLE, the retrieval language in ADAM, provided the input data is either supplied in the message or is available in an existing ADAM file.

Theoretically, a third method of generating files is available. In this case, the user must define his own file generation procedures in DAMSEL, SMAC, or a new language by using available system routines as building blocks.

IFGL is described in the following subsections.

GENERAL CHARACTERISTICS

File generation input consists of specifications and, optionally, file data. The specifications name and describe the properties of the ADAM file being generated and may describe the location and length of the corresponding values in the file input data; also described is the processing necessary to convert these values to internal representations in the ADAM file.

Source of Input

A file can be generated with or without input data. A null data file generation creates a complete property roll with no values in the file and a null object roll.

The input data can be on cards or tape. The card input data is in card code. The tape input data is assumed to be in 6-bit BCD code where the physical record length must not exceed 3189 characters (6-bit bytes). The logical record length, analogous to card length, must be less than or equal to the physical record length and must not exceed 1056 characters.

Data Fields

Data fields in the input may be variable or fixed length. The data in each entry must be grouped together and any field in that entry must be a

contiguous string. Input fields may be ignored. Individual properties may be described without any input data.

Names

Names may be prefixed (i. e., consist of more than one word) with the exception of file, roll, and routine names. Each name may have a number of synonyms, any one of which may be specified as the output PRINT name.

Data Modifications

Conversion

By using a conversion routine, the input data may be, and generally is, modified before being stored in the file. The system contains some conversion routines, or optionally, a user-defined conversion routine can be used.

Scale Factor

For numeric properties (floating-point, integer, and decimal), the property value may be multiplied with a scale factor, before it is stored in the file.

File Classification

A security classification may be assigned for the file. If the classification is other than UNCLASSIFIED, the classification is printed automatically at the top and bottom center of every page of the output.

Logical Rolls

Logical values are optionally added to one of the following rolls:

(a) object roll of this file

(b) object or property roll of another file

(c) a new or existing roll.

Protection

A property value may be protected thereby preventing any future change of that value except by special coding.

Format

The user may define his own output format for the file generation printout. Otherwise, the standard ADAM output format will be used.

Legality Checks

Numeric Range Check

For numeric properties (floating-point, integer, and decimal) a range check is made, following the optional multiplication of a scale factor, according to the user's specification in the property description. If the value is outside the range, it is rejected.

Sequence Check

A sequence counter may be set and compared with a data field. If the comparison fails, file generation is ended.

Optional Roll Additions

When the value for a logical property is not found in an existing roll, the 'ADDITIONS ALLOWED' option determines whether or not to add the value to the roll. Thus, if the 'ADDITIONS ALLOWED' option is not specified, the value is rejected if it is not in the roll.

Conversion

Illegal characters may be detected by the conversion routine.

INITIAL FILE GENERATION LANGUAGE DECK

Preparation

In the Initial File Generation Language (IFGL), the file description may be punched on cards or typed on-line. On-line message input is generally not feasible, since the description tends to be too long; therefore, card specifications are usually used.

A file description in IFGL must be a separate input message. It is punched or typed in any desired format, with any number of sentences per card or line, in columns 1 through 80, with the exception that the first character of the message must be nonblank and in column 1 of the first card.

Previously defined string substitutions may be used in the file description.

The file data itself may be on cards or tape in any desired format. The restrictions of the tape input data are given above in the paragraph entitled Source of Input. The card file input data is terminated by B, EOF punched in columns 1-5 of the last card. The tape file input data is terminated by an end-of-file mark on the tape. The tape input data is located on a tape whose IOD is assumed to be 7.

Notation

In this account of the Initial File Generation Language, we have used a system of notation similar to the one used to describe COBOL. It's salient features are outlined below.

Key Words

Key words are all upper case words that are underlined. These words must be used precisely as specified.

Example: BEGIN OBJECT

Optional Words

Optional words are all upper case words that are not underlined. They may be used to improve readability, or may be omitted for succinctness.

Example: CONVERT USING

User-Defined Phrases

User-defined phrases are all lower case words or groups of words that represent values to be supplied by the user, following procedures delineated elsewhere in the document.

Example: GENERATE FILE, name.

Braces

When two or more phrases are enclosed within braces, a choice must be made from the entries enclosed therein.

Example:

$$\left\{ \begin{array}{c} \underline{\text{FLOATING}} \\ \underline{\text{INTEGER}} \\ \underline{\text{DECIMAL}} \end{array} \right\}$$

Brackets

Information enclosed within square brackets is optional. It may be included or omitted, as required.

Example: [PROTECT]

Punctuation

All punctuation must be used precisely as specified.

FILE DESCRIPTION

The language description that follows is a series of statements in the notation described in the previous subsection, augmented by explanatory paragraphs where the language is not self-describing. Appendix III gives an example of file generation description.

Syntax

A file description consists of the series of statements outlined below:

GENERATE FILE, name,

[NULL DATA,]

[{ PRINCIPAL VALUE | PV } number,]

[ESTIMATED LENGTH number PAGE [S] ,]

[{ NO PRINT | PRINT FORMAT name* } ,]

{ SCR | TAPE, number CHARACTERS }

[, CLASSIFICATION name] . **

BEGIN OBJECT [, number PERCENT SLOP]

[set sequence counter command]

[position. . . [position]]

[property description. . . [property description]]

END OBJECT. [position. . . [position]]

* Must be an element in the format roll.
**Must be an element in the classification roll.

Discussion

Name

name

prefixed name [(PRINT)] . . . [,prefixed name [(PRINT)]]

The names of the file, properties, and new rolls (if any) are entered as elements of their corresponding rolls. The first prefixed name is considered the principal name. All subsequent prefixed names are synonyms. If one synonym is followed by the notation (PRINT), that synonym will be used in any output referencing the data denoted by this set of names. If (PRINT) is not specified, the principal name is used. Any of the names may be used interchangeably in FABLE and DAMSEL.

The following restrictions apply to the choice of names for properties:

(1) The following names may not be used for properties:

OBJECT SIZE	DEAD SPACE	DELETE BIT
VAR. DATA START	OVERRIDE BIT	LENGTH OF SLOP

(2) The following names must be used if the values are in the data base for an object:

OBJECT NAME PRINCIPAL CLASSIFICATION

ALTERNATE CLASSIFICATION

These are all LOGICAL type properties and their property descriptions should specify the use of OBJECT or CLASSIFICATION rolls. NAME is used only for names of repetitions in a repeating group.

(3) Properties may have the same name if they are either members of different repeating groups or one is a non-repeating-group property and the others are members of different repeating groups.

The following restrictions apply to the choice of names for the new files:

(1) The name may not be ROUT.

(2) The name may not be the same as any other file name in the data base.

(3) The name may not be prefixed.

The following restrictions apply to the choice of names for new rolls:

(1) The name may not be the same as any existing named roll.

(2) The name may not be prefixed.

<u>Prefixed Name</u>

prefixed name

simple name...[simple name]

Any number of spaces may be used between simple names, but they will be reduced to one space before the name is added to the roll.

<u>Simple Name</u>

A simple name is a string of arbitrary length formed from the characters A, B, C,...Z, Ø, 1,...,9.

<u>Null Data</u>

The phrase <u>NULL DATA</u> implies no file data. Neither the B, EOF termination card nor the end-of-file tape mark is required.

<u>Principal Value</u>

The principal value or PV number must be an available (but not deleted) PV and not greater than 127. If two files are saved during separate file generation tasks, the same PV may be assigned to both by FILDEF (the initiate file routine). Later, if both files are to be restored during the same run and if they have the same PV, the allocation for the first file restored is released

and therefore destroyed by DABS, the data base save and restore program. By assigning a unique PV to each file, this can be avoided.

Number

number

[{±}] integer [. integer] [([{±}] integer)]

The positive or negative integer in parentheses is the power of 10 by which the mixed decimal is to be multiplied.

Integer

An integer is a string of arbitrary length formed from the characters Ø, 1, 2, ..., 9.

Estimated Length Number

The ESTIMATED LENGTH number is an estimate, in pages (arcs), of the space that the new file will occupy. In large file generations, the estimate must be made to avoid an internal table overflow. The number of estimated arcs is a function of the amount and character, i.e., property types, of the input data. There is no penalty for overestimation provided the estimate plus the system startup requirement* is less than the disk IOD request.

No Print

The NO PRINT option suppresses printing of the generated file and is used when the file generation is very long, thus avoiding excessive data print-out. If a print specification is not given, the standard ADAM output print format is used.

---

*Includes any disk used by the data base restoration program.

SCR and TAPE

SCR implies that the file input data is from the extended system card reader. The logical record length is assumed to be 80 characters. If the file input data is from tape, the 'number CHARACTERS' is the number of 6-bit BCD characters per logical record on the input data tape.

Percent Slop

Slop is the empty space between fixed and variable data, per entry, and allows the addition of variable data to an entry after file generation without increasing the size of the entry and thereby causing noncontiguous disk allocation for the expanded entry. If the PERCENT SLOP of the object is not specified, a standard percentage (10) will be used.

Positions

The first set of 'positions' is a sequence of instructions to the file generation program that positions an imaginary input pointer to the first object of the data base. It is an initial position and executed only once.

The second set of 'positions' is a sequence of instructions that positions the imaginary input pointer to the start of the next object. It is executed after data for an object has been processed. See the paragraph entitled Position, below.

Set Sequence Counter Command

The 'set sequence counter command' is used to set the sequence counter to check the validity of the input data and is discussed in the paragraph entitled Set Sequence Counter Command, below.

PROPERTY DESCRIPTION

Each of the following qualifies as a property description:

logical property description

floating-point property description

numeric property description

raw property description

repeating group property description

sequence check

Logical Property Description

LOGICAL properties have their values listed in a roll. It is possible to use an existing roll, or to create a new roll, that is shared by several logical properties of the file being generated.

Syntax

LOGICAL, OBJECT NAME,

field description [CONVERT USING {code conversion name / entry point name}]

USE OBJECT ROLL

or:

LOGICAL, name,

{ NULL DATA / field description [CONVERT USING {code conversion name. / entry point name.}] }

[PROTECT.]

USE { OBJECT ROLL. | NEW ROLL name. | ROLL name [(ADDITIONS ALLOWED)]. | OBJECT ROLL OF name FILE. | PROPERTY ROLL OF name FILE. }

Discussion

If no data for the property 'OBJECT NAME' is to be read in, the property must not be described.

The phrase 'NULL DATA' is used when values for the property described are not part of the input data base, and will be added to the file later.

The optional 'CONVERT USING' phrase specifies the form of the data in the input data base, and the type of code conversion to be used when moving the data into the new ADAM file (see CONVERT DESCRIPTION).

The 'PROTECT' verb is used to specify file protection for this property. The value of a protected property cannot be changed after the file has been generated.

The 'USE···ROLL' phrase names the roll that will be used to relate the logical values to their alphanumeric representation. If two or more logical properties are to use the same roll, the 'NEW ROLL' phrase should be used in the description of the first of these properties, and the 'ROLL..' phrase in the others.

'ADDITIONS ALLOWED' is used to indicate that values for this property in the data base will be automatically added to the specified roll if not already there. If the 'ADDITIONS ALLOWED' option is not chosen and a value for the

property is not in the roll, the following message will be sent to the user and processing will continue:

VALUE NOT IN NAMED ROLL. NO VALUES STORED.

For 'field description' see paragraph of that title, below.

Example

LOGICAL, OBJECT NAME,
LENGTH IS 20 COLUMNS.
CONVERT USING CAA.
USE OBJECT ROLL.

LOGICAL, CITY,
SPACE 10 COLUMNS.
LENGTH IS VARIABLE, SCAN UP TO '**'.
PROTECT.
USE NEW ROLL LOCATION.

Numeric Property Description

Syntax

{ FLOATING | INTEGER | DECIMAL } , Name,

{ NULL DATA. | field description [CONVERT USING { code conversion name. | entry point name. }] }

[PROTECT.]

[CONVERT IN entry point name.] [CONVERT OUT entry point name.]

[MAX number,] [MIN number,]

[SCALE FACTOR number,] integer DIGIT [S].

Discussion

The 'CONVERT IN entry point name' (if any) specifies a user-defined routine to convert values of the numeric property expressed in FABLE from their external form to their internal representation in the file. An output conversion routine (if any) is used to convert the internal representation of the value to an output form in response to a FABLE query, and is specified in the 'CONVERT OUT entry point name' phrase.

The 'MIN' and 'MAX' phrases specify the expected minimum and maximum values for this property after it has been multiplied by the scale factor (if any). If a value being stored is not within the specified range, the following message will be sent to the user and processing will continue:

DATA OUT OF RANGE. VALUE NOT STORED. PROPERTY NAME IS name.

If a 'SCALE FACTOR' has been specified, the numeric quantity in the input data base will be multiplied by this factor before it is stored in the new file. The scale factor is not limited to powers of ten.

The 'integer DIGIT[S]' phrase specifies the maximum number of significant decimal digits which will be used to create values for this property and determines how much space should be provided for the property in the new file. In addition, if a 'MIN'and/or 'MAX' is not given, the 'integer DIGIT' is used as the expected minimum and/or maximum value(s) for the property.

Example

INTEGER NUMBER OF RUNWAYS,
SPACE 2 COLUMNS. SPACE TO NON ' '.
LENGTH IS 3 COLUMNS.
CONVERT USING CDB.
MAX 100, MIN 1, 3 DIGITS

DECIMAL, NUMBER OF FLIGHTS,

SPACE TO NON' '.

LENGTH IS 4 COLUMNS.

4 DIGITS.

Raw Property Description

Syntax

RAW, name,

{ NULL DATA. | field description [CONVERT USING {code conversion name. | entry point name.}] }

[PROTECT.]

PRINT {STANDARD. | entry point name.}

Discussion

If standard ADAM output printing is not requested, a user-defined print routine is executed. The entry point name to the routine must be defined in the compiler roll.

Example

RAW, CODE NAME,

SPACE TO '*'. SPACE TO 'A' OR 'B'.

LENGTH IS VARIABLE, SCAN UP TO 'Ø' OR '*'.

CONVERT USING CAA.

PROTECT.

PRINT STANDARD.

Repeating Group Property Description

Syntax

```
                          ( NULL DATA
BEGIN GROUP, name,       <  TERMINATED BY [NON] 'string of characters'...
                          (                  [OR 'string of characters'].

[set sequence counter command]
[position...[position]]
BEGIN REPETITION.
[property description...[property description]]
END REPETITION.
[position...[position]]
END GROUP, name.
```

Discussion

The phrase 'NULL DATA' is used when the values for the repeating group are not part of the input data base. All property descriptions in a null repeating group must, therefore, use the 'NULL DATA' phrase.

A 'string of characters' is a string of arbitrary length formed from the A8 character set excluding a prime (').

The phrase beginning 'TERMINATED BY...' is used to specify a string or a set of possible strings that the file generation program should look for in the input data base, indicating the end of values for a repeating group property. The test for the end of a repeating group property is made before each repetition is processed (including the first). If the modifier 'NON'* is used, the file generation program will look for any string of characters except the string following NON. For example, NON ' ' means that a single nonblank character

---

*Applies to all character strings.

signals the end of this group. Note that when a repeating group is terminated, the imaginary position pointer is positioned to the first character of the terminating string.

If one of the properties described within the group is the logical property NAME, then the group is considered to be a named repeating group. If not, the group is unnamed, and repetitions will be numbered.

The first set of 'positions' is executed before the first repetition is processed and sets the imaginary input pointer to the place where the test for the terminal string(s) is to be made.

The second set of 'positions' is executed after each repetition is processed and also sets the imaginary input pointer to the place where the test for the terminal string(s) is to be made.

The name that follows '<u>END</u> <u>GROUP</u>' must agree with the name that follows '<u>BEGIN</u> <u>GROUP</u>'.

<u>Example</u>

An example of the repeating group is given in the file described in Appendix III.

<u>Sequence Check</u>

<u>Syntax</u>

<u>SEQUENCE</u> <u>CHECK,</u> field description

<u>CHECK</u> <u>FOR</u> <u>SEQUENCE</u> <u>NUMBER</u> relation <u>PREVIOUS.</u>

<u>Discussion</u>

If sequence checking is specified, a field in the data is compared with the current contents of the sequence counter according to the specified relation. The imaginary position pointer is moved to the first character after the data field.

If the comparison is successful, the value in the data replaces the value in the sequence counter. If unsuccessful, the file generation process is terminated.

The following six phrases denote the relations allowed:

LESS

EQUAL

GREATER

NOT LESS

NOT EQUAL

NOT GREATER

Example

SEQUENCE CHECK,

LENGTH IS 3 COLUMNS.

CHECK FOR SEQUENCE NUMBER GREATER PREVIOUS.

FIELD DESCRIPTION

The syntax for field description is:

[position...[position]] length description

[set sequence counter command]

Position

Syntax

SPACE TO NEXT CARD.

RESET TO OBJECT START.

SPACE BACKWARD {integer CARD[S] [, integer COLUMN[S]] | integer COLUMN [S]}

SPACE [BACKWARD] TO [NON]* 'string of characters'...
[OR 'string of characters'].

An integer is a string of arbitrary length formed from the characters Ø, 1, 2, ...9. A 'string of characters' is a string of arbitrary length formed from the A8 character set excluding a prime (').

Discussion

One CARD is equivalent to 80 COLUMNS. The words CARD[S] and COLUMN[S] are used even in file generations with tape input. In this case, one CARD is equivalent to the number of characters per logical record given in the file description. COLUMN[S] now is equivalent to character[s] (6-bit byte[s]).

POSITION moves an imaginary pointer to point to the beginning or end of a data field. Thus, it locates the start of an object or the value of a property. At the beginning of a file generation the pointer points to column 1 of the first card or, for tape input, to byte 1 in the first record. The pointer is moved by position statements, length descriptions, and sequence checks. All positioning statements, apart from 'RESET TO OBJECT START', move the pointer relative to the previous pointer location.

The 'integer COLUMNS' phrase is currently limited to 2047 columns (characters), or the equivalent number of cards (logical records) and columns (characters).

In no case is it possible to space backward past the beginning of the data corresponding to the current object.

---

*NON is applied to all following character strings.

Examples

SPACE TO NEXT CARD.

The imaginary pointer is moved to point at column 1 or byte 1 of the next card or record.

RESET TO OBJECT START.

The imaginary pointer is moved to point at the previous pointer location at which the current object was started.

SPACE 1 CARD, 14 COLUMNS.

The imaginary pointer is moved forward 1 card and 14 columns (that is, 80 + 14 columns) or, for tape input, 1 logical record and 14 characters (that is, the number of characters per logical record + 14 characters).

SPACE TO 'ABC' OR 'XYZ'.

The positioning routine examines three characters starting with the one at which it is currently pointing. If they are 'ABC' or 'XYZ', the imaginary pointer is not moved. Otherwise, the pointer is moved forward one character at a time and the test for 'ABC' or 'XYZ' is made again until either a match or the end of the input data is found. If the test is successful the pointer is positioned at the character A in 'ABC' or X in 'XYZ'. If unsuccessful, the file generation is terminated.

SPACE BACKWARD TO NON 'A'.

If the pointer is not currently pointing to a NON 'A', it is moved backward to point at the first NON 'A' character found.

SPACE BACKWARD TO NON 'A' or 'B'.

Since NON applies to both A and B, a match is always found immediately and the imaginary pointer is not changed.

Length Description

Syntax

LENGTH IS
- integer CARD[S] [, integer COLUMN[S]]
- integer COLUMN[S]
- VARIABLE, SCAN UP TO [NON]* 'string of characters'... [OR 'string of characters']

An integer is a string of arbitrary length formed from the characters Ø, 1, 2,..., 9. A 'string of characters' is a string of arbitrary length formed from the A8 character set excluding a prime (').

Discussion

One CARD is equivalent to 80 COLUMNS. For tape input, one CARD is equivalent to the number of characters per logical record, and COLUMN[S] is equivalent to character[s].

The length description statement describes a data field. The data field starts with the character at which the imaginary position pointer is pointing. The length of the data field is either fixed, given directly as the number of characters, or variable, in which case the last character is the one immediately before the terminating string of characters. The length description moves the imaginary pointer to point to the character following the defined field; hence the SCAN UP TO phrase positions it to point to the first character in the found string of characters. Therefore no position statement is necessary when the data is packed and in the sequence in which the properties are defined.

---

*NON is applied to all character strings.

The 'integer COLUMNS' phrase is limited to 2047 columns (characters) or the equivalent number of cards (logical records) and columns (characters).

Examples

LENGTH IS 20 COLUMNS.

The length of the data field is 20 columns (bytes), starting with the character at which the imaginary position pointer is pointing. The pointer is moved forward 20 columns (bytes).

LENGTH IS VARIABLE, SCAN UP TO '*'.

If the position pointer is pointing to a '*', this would be an error, since the length would be zero. Otherwise, the length of the data field is variable, starting with the current location of the imaginary position pointer and ending with the last character before the '*'. The pointer is moved forward to point at the found '*'.

LENGTH IS VARIABLE, SCAN UP TO NON 'A' OR 'B'.

This would be an error with zero length as a result, since the NON applies to both 'A' and 'B', therefore, the scanning finds an immediate match.

Set Sequence Counter Command

Syntax

SET SEQUENCE COUNTER TO integer.

Integer is defined as a string of arbitrary length formed from the characters Ø, 1, 2,..., 9.

Discussion

The sequence counter may be set at the start of file data, data for an object, repeating group, or property. It is used to make a sequence check

to check the validity of the input data. The sequence check is described in the section by that name.

Example

SET SEQUENCE COUNTER TO 040.

CONVERT DESCRIPTION

Syntax

CONVERT USING $\left\{\begin{array}{l}\text{code conversion name.}\\\text{entry point name.}\end{array}\right\}$

Discussion

The CONVERT description describes which conversion to apply to the input data field before storing it into the new ADAM file. The system contains a number of standard conversions. If these are not sufficient, the user has the option of coding a specific code conversion routine and then naming it in the 'CONVERT USING ..' phrase (entry point name). The user-defined conversion routine must be coded in accordance with the standard ADAM conventions for conversion routines. The user's routine entry point names must first be defined in the compiler roll and can then be used in the CONVERT USING... phrase. A code conversion name is one of the following:

Name	Conversion*
NULL	Don't convert
CDB	Card decimal to binary
CAA	Card alpha to ADAM alpha
CFA	Card floating to ADAM floating

*'Card", in this context means 'card' if the input data is on cards or 'tape' if input is on tape.

COB	Card octal to binary
CBB	Card binary to binary
CHB	Card hex to binary
LLC	Lat-long to ADAM floating

Additional code conversions may be added to this list as the need arises.

File data is read from the extended system card reader or from a tape, with IOD 7, in binary form. If NULL Conversion, the data is passed on directly. In CAA, data is converted to ADAM alphabetic. All the other conversions are composed of a card or tape code - ADAM alpha conversion - followed by a specific conversion. A description of the individual conversions follows, including the limitations of the corresponding data.

NULL - The input string is to be left alone; no conversion operations are performed. Data may be any length up to 272 cards.

CDB - The input string is a decimal number in card or tape code. It is to be converted to the equivalent binary number. A sign may optionally precede the number. There are at most 15 decimal digits in the string.

CAA - The input string is card code or A6 text. It is to be converted to equivalent A8 text. It may be any length up to 255 characters.

CFA - The input is a card or tape code string representing a floating point number; it is to be converted to an internal floating point number. Decimal integers, decimal fractions, and mixed decimal numbers (I or I., .F, or I.F) are acceptable provided no more than 15 decimal digits appear in the integer and fraction portions combined. A preceding sign is optional. A following exponent is optional; if it appears it must be in the decimal radix

and preceded by the character E. A sign between E and the exponent is optional.

COB - The input string is an octal number in card or tape code. It is to be converted to the equivalent binary number. A sign preceding the number is optional. The maximum number of octal digits allowed is 20.

CBB - The input string is a binary number in card or tape code. It is to be converted to the equivalent binary number. A preceding sign is optional. The number may not exceed 48 bits.

CHB - The input string is a hexadecimal number in card or tape code. It is to be converted to the equivalent binary number. A sign preceding the number is optional. The maximum length of the number is 12 hexadecimal digits.

LLC - The input is a six-character string, in card or tape code, of the form LDDDMM

where

(a) L is one of the letters N, E, S, or W;

(b) D and M are decimal digits representing degrees and minutes of arc.

The string is to be converted to a floating point number equal to the equivalent number of minutes of arc positive for N or E, negative for S or W.

CAA and NULL accept all fields (including blanks) as valid data. The other routines will accept a completely blank field, but create a NULL data value after which normal processing is resumed. Any other invalid data will cause a null value to be stored, after which, processing is resumed and an appropriate comment is printed.

If there is no 'CONVERT USING...' phrase, the system uses one of the standard conversion routines; which routine is used is contingent upon the property type as indicated below:

Property Type	Conversion Applied
Logical	CAA
Floating	CFA
Integer	CDB
Decimal	CDB
Raw	CAA

SECTION IV

DAMSEL

DAMSEL (ADAM SEnsitive Language) is used to signify both a compiler and the DAMSEL language which it translates. This section describes the DAMSEL language, a statement language that is ADAM oriented. It allows a user to specify file manipulation operations and, if he wishes, to reference file data by name. The DAMSEL compiler operates within the ADAM System and, therefore, has access to data base descriptions. The input to the DAMSEL compiler may be DAMSEL, SMAC, or STRAP statements. The output is SMAC code which must be compiled. Because the SMAC compiler may not be called after ADAM operation in a single job, the compilation of a DAMSEL input requires two linked jobs. The first uses the DAMSEL compiler to produce a tape containing SMAC statements. The second job is a SMAC compilation which uses the output tape from the first job. In addition, the second job may operate the ADAM System, add the routine to the system, and execute it within ADAM.

GENERAL CHARACTERISTICS

System Names

System names are used in DAMSEL programs to refer to ADAM data structures. The following types of system names can occur:

file-name:	defined by the file pointer set
roll-name:	defined by the roll pointer set
unit-name:	defined by the unit roll

group name:	defined by a property roll and of type NRG or URG
property-name:	defined by a property roll and not of type NRG or URG
object-name:	defined by an object roll

A system-name has one of the following forms: label; or 'word'. Label is any alphanumeric string of characters starting with a letter, and 'word' is any string of characters within primes ('). 'Word' may thus contain blanks. Any number of consecutive blanks is equivalent to a single blank. Thus

'NEWVVVVVYORK'

is equivalent to

'NEWVYORK'

System Specifiers

Specifiers are used in DAMSEL statements to specify file properties, rolls, and units. System specifiers are constructed using system names. Syntactically correct system specifiers may be undefined in a particular ADAM system.

A property specifier defines a file property and has one of the following forms: (a) file-name, property-name, which defines a prime-level properly; and (b) file-name, group-name,. . . , property-name which defines a property which belongs to a repeating group. All ancestral group names must appear in descending order.

A roll specifier defines a roll and has one of the following forms: roll-name, which specifies a name defined by the roll pointer set; PROP (file-name) and OBJ (file-name), which specify the property role and object roll of a file; and property-specifier, which specifies the name roll of a property (which must be of type LOG or NRG).

A unit specifier defines a unit and has one of the following forms: unit-name, which specifies a name defined by the unit roll; and property-specifier, which specifies the unit associated with a property (must be of type FP, CFL, or INT).

Variables and Type Specifiers

Routines perform computations on variables whose values are full-word (64-bit) quantities stored at full-word locations. All variables must have an associated description. In some cases, the description is "built-in" (e.g., a variable which is a file property gets its description from a property roll). In other cases, the description must be defined by the user. The type specifier can be used for this purpose and has the following forms:

(a) (1) REAL

    (2) REAL (unit-specifier)

(b) INTEGER

(c) (1) ROLL

    (2) ROLL (roll-specifier)

(d) LOCATION

(e) STRING

A variable of type REAL is a normalized floating point number. In REAL (unit-specifier) a unit name is associated with the variable by unit-specifier.

A variable of type INTEGER is an unnormalized floating point number with the exponent, +38.

A variable of type ROLL is an unnormalized floating point number with the exponent, +38. In ROLL (roll-specifier) a roll name is associated with the variable by roll-specifier.

A variable of type LOCATION is an index word having the following form:

XW, value, Ø, Ø, Ø

A variable of type STRING is an index word having the following form:

XW, location, bit-count, Ø, Ø

and defines a string of A8 characters.

Type specifiers are used to declare variables in the following statements:

parameter declarations
variable declarations
data statements.

A particular variable may be declared more than once but all of its declarations must be consistent. For example, it is permissible to declare a variable to be

REAL and REAL (MILES)

but it is not legal to declare a variable to be

ROLL and INTEGER

or even

ROLL (PROP(AIRFIELD)) and ROLL (PAIR)

Card Format

SMAC and STRAP cards must contain a blank or a minus (-) in column 1. They are translated by the SMAC and STRAP compilers.

DAMSEL cards must contain a dollar sign ($) or a plus (+) in column 1. The location field - columns 2 through 9 - may contain a label which identifies the statement. The actual statement may begin in column 10 or any following

column. DAMSEL statements may occupy more than one card; any continuation cards must contain an asterisk (*) in column 1. Columns 2 through 9 are ignored on continuation cards.

Comments may occur on DAMSEL cards; the start of the comment is denoted by a prime (') and the comment continues until the card ends. Comments may be continued. A comment should not contain a prime ('), since any string of characters between two primes is presumed to be a system name.

Blanks on DAMSEL cards terminate words and are not ignored. For example,

DTAIL X

is a legal statement, since DTAIL is a legal keyword; but

DTA IL X

is not a legal statement, since DTA is not a legal keyword.

System Defined Variables

Variables occurring in a program must normally be defined by the program (e.g., by occurring in a location field). The variables described in this section are:

(a) predefined by the ADAM system

(b) may be used on SMAC and STRAP cards as well as DAMSEL cards

System Tailed Symbols

The STRAP pseudo-operations, TAIL and UNTAIL, are not allowed in DAMSEL decks and have been replaced by a DAMSEL declaration, DTAIL, which permits single-level tailing.

Single character tail labels have been reserved for use by the ADAM system. The following "system tails" have been defined:

$P If label is a routine name or an entry name, then the value of
    label $P
    is the PV of label or the PV of the routine defining label in the routine roll.

$E If label is an entry name, then the value of
    label $E
    is the entry number of label.

$G If label is a global symbol, then the value of
    label $G
    is the value of the global symbol.

$I If label is a global symbol, then the value of
    label $I
    is label $G plus the starting core location of the routine which defines label, i.e., the current absolute location of that label. I-tailed globals cannot be evaluated until load time.

$A If label is a global symbol defined by the PAT or FAVOR tables, then the value of
    label $A
    is the value of label $I.

$F If system-name is a SYSTEM NAME, q.v., the value of system-name $F
    is defined to be the PV of the system-name in the File Pointer Set.

$R Analogously the value of

system-name$R

is defined to be the PV of system-name in the roll pointer set.

The following notation can be used to look up names in arbitrary rolls. The value of the symbol is defined to be the PV of the specified system name in the specified roll:

system-name$(roll-specifier)

Index Symbols

The following single letters can be used to refer to certain index registers: I, J, K, L, M, and N. Their meaning is defined as follows:

I:	$7	J:	$8	K:	$9
L:	$10	M:	$11	N:	$12

Parameter References

References to routine parameters have the form

P.label

where label is defined by a parameter declaration. Parameter references are normally replaced by expressions of the following form:

integer . ($3) (input parameter)

integer . ($4) (output parameter)

The index registers involved ($3 and $4) are automatically housekept if the routine uses the enter statement and the exit statement.

DAMSEL DECLARATIONS

A declaration specifies general information about a program and, in general, does not produce executable code. A DAMSEL declaration may not have a label in the location field.

Every deck must begin with a routine declaration or a table declaration and end with an end declaration.

Tables contain data and are not called for execution. Routines contain entry-name options - defined by entry declarations - which may be called for execution. Each entry-name option has an associated parameter configuration; parameters are defined by parameter declarations.

The first statement to be executed in a routine is specified on the routine declaration and is normally an enter statement. The various entry-name options are entered, at the conclusion of code common to all options, by the entry branch statement. The last statement to be executed by a routine must be an exit statement.

The following example illustrates the macro structure of a routine (called SINCOS), which has two entry options (called SIN and COS), each having one input parameter and one output parameter.

```
$          ROUTINE   SINCOS, ST(START)   starting location is START
$          INPAR     IN,REAL             input parameter IN is real
$          OUTPAR    OUT,REAL            output parameter OUT is real
$          ENTPAR    SIN (IN,OUT)        entry name SIN
$          ENTPAR    COS (IN,OUT)        entry name COS
$  START   ENTER
           code common to SIN and COS
$          ENTRY     BRANCH              go to various entry options
```

```
$  SIN     code for entry SIN
$          EXIT
$  COS     code for entry COS
$          EXIT
$          MEND
```

Routine Declaration

The first card in a routine deck must be a routine declaration which has the following forms:

ROUTINE	routine-name
ROUTINE	routine-name, routine-description

routine-name defines a name and a mod-number for the routine and has the forms:

label	name only
label (unsigned-integer)	name and mod-number

routine-description describes the routine and consists of a sequence of terms, separated by commas, chosen from the following terms:

(1) routine PV designator - specifies a PV for the routine and has the form:

PV (unsigned-integer)

If it is omitted, the routine does not have a PV associated with it.

(2) routine type designator - defines the type of the routine and has the forms: FIXED and FIXED (unsigned-integer), which define fixed routines; and ALLOC and ALLOC (unsigned-integer), which define allocatable routines.

ALLOC and ALLOC (unsigned-integer) associate with the routine a relative location in the fixed routine subsection of the program allocation table (PAT). If the routine type designator is omitted, ALLOC is obtained.

(3) listing designator - specifies the listing option to be used for the compilation and must be one of the following key words:

LISTALL	list all cards
LISTDAM	list DAMSEL statements only
NOLIST	list only cards containing errors

If the listing designator is omitted, LISTDAM is obtained.

(4) The starting location designator - specifies the location field symbol of the first statement to be executed in the routine and has the form:

ST (label)

If it is omitted, this label is assumed to be START.

Examples of the routine declaration are:

$	ROUTINE	SINCOS
$	ROUTINE	SINCOS, PV(7), NOLIST, FIXED(8)
$	ROUTINE	SINCOS, ST(S1C1), ALLOC

Table Declaration

The first card in a table deck must be a table declaration which has the following forms:

TABLE	table-name
TABLE	table-name, table-description

table-name has the same structure as routine-name and table-description has the same structure as routine-description. A table should not have a starting location designator since tables do not contain executable code.

Parameter Declaration

Parameter variables must be declared before being referenced either by input parameter declarations:

```
INPAR       label, type-specifier
```

or output parameter declarations:

```
OUTPART     label, type-specifier
```

label (a label) specifies the name of the parameter; type-specifier specifies the type of the parameter. Parameter declarations are used by entry declarations to define parameter configurations.

Examples:

```
INPAR       PI,REAL(MILLIMETERS)
INPAR       P2,REAL(BUG, LENGTH)
OUTPAR      P01,ROLL(PROP(BUG))
OUTPAR      P02,ROLL(BUG,'LEG GROUP')
```

Entry Declaration

The entry options of a routine are defined by entry declarations which have the following forms:

```
ENTPAR      entry-specifier
ENTPAR      entry-specifier ()
```

ENTPAR entry-specifier(label$_1$. , . . . , label$_n$) n $\geq$ 1 where entry-specifier is a tailed label and label is a previously defined parameter name.

The entry declaration defines an entry name, an entry starting location and a parameter configuration. The entry name and entry starting location are defined by an entry-specifier which has the following forms:

label

label$

label$tail

The entry name, label, is used in call statements and function references to execute the entry option. Executable code must occur in the routine for each entry option, and the first executable statement is specified by the tailed label. The tailing expressed on the entry declaration must be relative to the entry branch statement used to enter the option (see the following example).

A parameter configuration is an ordered list of parameter names. ENTPAR specifies <u>no</u> parameters. ENTPAR ( ) specifies <u>all</u> parameters, and the order specified is the order in which the parameter declarations occur. ENTPAR (label$_1$, . . . , label$_n$) explicitly specifies an ordered list of parameters. The parameter configuration specifies the order in which arguments are written in the call statement and the function reference.

Example:

```
$    ROUTINE    RI, ST(START$T1)
$    ENTPAR     E1$
$    ENTPAR     E2$T2
$    ENTPAR     E3
$    DTAIL      T1
```

```
$START   ENTER    T1
         code common to all entries
$        ENTRY    BRANCH
$        DTAIL
$E1      code for entry E1
$        DTAIL    T2
$        code for entry E2
$        DTAIL    T1
$E3      code for entry E3
$        MEND
```

Variable Declaration

Local variables must be declared by using a variable declaration which has the following form:

VARIABLE type-specifier/tailed-label$_1$, . . . , tailed-label$_n$/ $n \geq 1$

where type-specifier defines the type of the variable and tailed-label names the variable.

Data statements may also be used to declare variables.

Undeclared variables are presumed to be of the type REAL.

Examples:

```
VARIABLE   ROLL(PROP(BUG))/R1,R2$TAIL/
VARIABLE   INTEGER/I1, I2$, I3/
```

Tail Declaration

The STRAP pseudo-operations, TAIL and UNTAIL, are not permitted in DAMSEL decks. Single-level tailing is provided via the tail declaration which has the following forms:

DTAIL label

DTAIL

The tail declaration defines a current tail symbol (CTS). DTAIL label defines the CTS to be label. DTAIL defines the CTS to be all blanks. Initially the CTS is all blanks.

Local variables that are defined subsequent to a tail declaration are automatically suffixed by the CTS. References to variables must be made by tailed labels which have one of the following forms:

label

label$

label$tail

where tail is a label.

Label refers to label suffixed by the CTS. Label$ refers to label suffixed by blanks. Label$ tail refers to label suffixed by tail.

Begin Globals Declaration

The begin globals declaration has the form:

BEGIN GLOBALS

Only one begin globals declaration may occur in a deck and it must be followed by an end globals declaration. All location field labels that are bracketed by these declarations become global symbols unless column 1

of the card eontains the eharaeter plus (+) or minus (-). The "eseape eharacter" plus (+) is used for DAMSEL statements and minus (-) is used for SMAC and STRAP statements.

The global symbols defined in a routine are permanently associated with the routine and may be refereneed as system tailed symbols.

End Globals Declaration

The end globals deelaration has the form:

END GLOBALS

and is used in combination with the begin globals deelaration to define global symbols.

End Declaration

The last statement in a symbolie deek must be an end deelaration of the form:

MEND

DATA MANIPULATION STATEMENTS

The statements deseribed in this seetion are used to handle ADAM files and areas. Many of the parameters involved in data statements are deseribed by data statement variables whieh have the following forms:

S(loeation)	S(parameter)
I (location)	I(parameter)

loeation speeifies a full-word address in eore storage; parameter is a parameter refereneе.

S(location) and S(parameter) specify that core storage contains a variable of type STRING. I(location) and I(parameter) specify a variable of type INTEGER.

Examples:

S(ABC+1.)

S(P.P1)

S(X(J))

I((I))

I(ABC($7))

File Processing Statements

Files, file objects, and repeating groups are designated by file designators, object designators, and group designators respectively, which have the following forms:

(a) file designator
   (1) file name
   (2) data statement variable
(b) object designator
   (1) object name
   (2) data statement variable
(c) group designator
   (1) group name
   (2) data statement variable

In file name, object name, and group name the ADAM structure is identified by a system name which must be defined at compile time. In data statement variables, the ADAM structure is defined by a variable whose value

can not be known until execute time. The variable may be one of two types: STRING, giving the name of the structure; or INTEGER, giving the PV of the structure.

Open File Statements

ADAM files are opened by the open file statement, which has the form:

$label   OPEN(file-qualifier)   file-designator

where the file is specified by file-designator, and file-qualifier has the forms:

status, error exit
error exit

Error exit specifies the next statement to be executed if an error condition is obtained in opening the file. Status describes the status of the file and can be any of the following codes:

RW	Read and Write	Overwrite	Temporary
RWN	Read and Write	No Overwrite	Temporary
RWNP	Read and Write	No Overwrite	Permanent
RWP	Read and Write	Overwrite	Permanent
W	Write	Overwrite	Temporary
WN	Write	No Overwrite	Temporary
WNP	Write	No Overwrite	Permanent
WP	Write	Overwrite	Permanent
N	Read	No Overwrite	Temporary
NP	Read	No Overwrite	Permanent
P	Read	Overwrite	Permanent
blanks	Read	Overwrite	Temporary

The location field of the open file statement must contain a label which is used by other statements to refer to the file.

Examples:

$F1	OPEN (W, XYZ)	BUG
$F1	OPEN(, XYZ)	S(STBUG)
$F1	OPEN(, XYZ)	I(P. P1)

Open Entry Statements

The open entry statement opens entries in ADAM files and has the form:

$label OPEN (file qualifier) file-designator(entry designator)

The mode in which the file is opened is specified by file qualifier. The file is specified by file designator and the entry is specified by entry designator.

The location field of the open entry statement must contain a label which is used by other statements to refer to the entry.

Examples:

$E1	OPEN(RW, XYZ)	BUG(FLY)	
$E1	OPEN(, XYZ)	S(STBUG)	(I(PVFLY))
$E1	OPEN(, XYZ)	S(P. P1)	(S(P. P2))

Open List Statements

The open list statement opens lists of entries in ADAM files and has the form:

$label   OPEN(file qualifier)   file designator(object list)

where object list has the form:

object designator$_1$, ..., object designator$_n$ $n \geq 2$

The file is specified by file designator and the list is specified by object list. The mode in which the file is opened is specified by file qualifier.

The location field of the open list statement must contain a label which is used by other statements to refer to entry list.

Examples:

$L1	OPEN(WN, XYZ)	BUG(FLY, CENTIPEDE)
$L1	OPEN(, XYZ)	BUG(S(STFLY) , S(STCENT))

Open Miscellaneous Statements

The open miscellaneous statement opens miscellaneous sections of ADAM files and has the following form:

$label   OPEN MISC (file qualifier)   file designator

The file is specified by file designator, and the mode in which the file is opened is specified by file qualifier.

The location field of the open miscellaneous statement must contain a label which will be used by other statements to refer to the miscellaneous section.

Example:

$M1	OPEN MISC(RW, XYZ)	BUG
$M1	OPEN MISC(, XYZ)	S(P. P1)

For File Statements

The for file statement executes the corresponding BASAL option and has the following form:

FOR(error exit)   tailed label

where tailed label specifies the location of an open file statement.

For Group Statements

The for group statement executes the corresponding BASAL option and has the following form:

$label   FOR(error exit)   group designator(tailed label, non exit)

where group designator specifies a repeating group file property; tailed label specifies an open file statement or a for group statement; error exit specifies the next statement to be executed if an error condition is obtained; and none exit specifies the next statement to be executed if the group does not have any repetitions.

The location field of the for group statement must contain a label which is used by other statements to refer to the group.

Examples:

```
$G1   FOR(XYZ)   WINGS(F1, NONE)
$G2   FOR(XYZ)   COLOR(G1, NONE)
$G1   FOR(XYZ)   S(P. P1))F1, NONE)
```

Step Statements

The step statement is used to step ADAM files, lists, or repeating groups and has the form:

STEP(error exit)   tailed label, more exit

where tailed label specifies an open statement or a for group statement, error exit specifies the next statement to be executed if an error condition is obtained, and more exit specifies the next statement to be executed if another entry or repetition is found. A "drop through" occurs if no new entry or repetition is found.

Close Statements

The close statement is used to close a miscellaneous section of a file or a file entry and has the form:

CLOSE(error exit)   tailed label

where tailed label specifies an open statement (miscellaneous or entry), and error exit specifies the next statement to be executed if an error condition is obtained.

Release Statements

The release statement is used to release a file, entry, miscellaneous section, or entry list and has the form:

RELEASE(error-exit)   tailed-label

where tailed label specifies an open statement, and error exit specifies the next statement to be executed if an error condition is obtained.

Area Statements

Some of the area statements have parameters which are data statement integers and can have the forms:

integer

I(location)

I(parameter)

An integer can be any string of characters such that

VF, integer

is a legal STRAP instruction. I(location) and I(parameter) are special cases of the data statement variable and specify locations that contain integers.

Open Area Statements

The open area statement opens an ADAM area and has the form:

OPEN AREA(area qualifier)   label(page count)

where label specifies the name of the area and can be used as a variable in assignment statements and data transfer statements. The mode in which the area is opened is specified by the area qualifier which has the following forms:

status, error-exit

, error-exit

where error exit specifies the next statement to be executed if an error condition occurs in opening the area, and status describes the area status and may be any of the following codes:

NP	No Overwrite	Permanent
N	No Overwrite	Temporary
P	Overwrite	Permanent
blanks	Overwrite	Temporary

The page size desired for the area is specified by page-count which is a data statement integer. It may be omitted if a single page area is desired.

Examples:

```
OPEN    AREA(,XYZ)      AR1(2)
OPEN    AREA(P,XYZ)     AR1
OPEN    AREA(N,XYZ)     AR1(I(INT))
```

<u>Insert Area Statements</u>

The insert area statement inserts pages into an existing ADAM area and has the form:

INSERT(error exit)   tailed label(insert location, page count)

where tailed label specifies the name of the area, and error exit specifies the next statement to be executed if an error occurs; insert location is a data statement integer and specifies the relative location (0, 1, 2...) of the first inserted word; page count is a data statement integer and specifies the number of pages to be inserted. If a single page insertion is desired, page count may be omitted.

Examples:

```
INSERT(XYZ)    AR1(Ø,1)
INSERT(XYZ)    AR1$T1(I(LOC),I(CT))
INSERT(XYZ)    AR1$(I(LOC))
```

Release Area Statements

The release area statement releases a previously opened ADAM area and has the form:

RELEASE (error exit) tailed label

where tailed label specifies the name of the area, and error exit specifies the next statement to be executed if an error occurs.

File Generation Statements

The add repetition statement initiates the addition of a repetition to an instance of a repeating group property; it has the form:

ADD REP(error exit) tailed label

where tailed label specifies a four group statement and error exit specifies the next statement to be executed if an error occurs.

DATA TRANSMISSION STATEMENTS

Two statements, FETCH and STORE, are used to transmit data between ADAM files and core storage blocks. The core storage block involved is specified by a block designator and may be either part of a routine or part of an ADAM area. The file data involved is specified by a property designator which defines a nonrepeating-group property in the file. The property designator can have the following forms:

property name
data statement variable

The property name is specified by a system name which must be defined at compile time. In data statement variable form, the property is defined by a variable whose value will not be known until execute time. The variable may

be of type STRING, giving the name of the property, or of type INTEGER, giving the PV of the property.

Block Designators

A block designator defines a block of core storage registers. It may be either part of a routine (core block designator) or part of a previously opened ADAM area (area block designator).

A core block designator has the form:

CORE(tailed-label, relative-location, length)

where relative location and length are data statement integers.

The initial location of the block of storage registers is defined to be

tailed label + relative location

and length is the length of the storage block.

An area block designator has the following form:

AREA(tailed-label, relative-location, length)

where relative location and length are data statement integers, and tailed label specifies an ADAM area. The initial location of the block is defined as a relative location in the area by relative location and length specifies the length of the storage block.

Right-to-left drop-out is allowed in block designators. If length is omitted, it is assumed to be 1. If relative-location is omitted, it is assumed to be 0.

Examples:

CORE(C1$T1,1∅,5∅)

CORE(C1,I(LOC$T1))

AREA(A1$T1)

AREA(A1,I(LOC),I(LGT))

Fetch Statements

A fetch statement moves data from an ADAM file into core storage and has the form:

FETCH(error exit) property designator(tailed label) ,
block designator

where property designator specifies a nonrepeating group property, tailed label specifies an open file statement or a for group statement, and block designator specifies a block of storage registers.

If the type of the file property is not RAW, then the value of the property will be placed in both the accumulator and the first word of the specified storage block.

If the type of the file property is RAW, then the value of the property is placed in the raw data stream, an SHP* stream pointer is placed in the accumulator, and as much data as will fit is moved into the storage block from the raw data stream.

A block-designator is optional. If it is omitted, the data (or a pointer to it) appears in the accumulator.

---

*Stream handling pointer routine.

Examples:

FETCH(XYZ$T1)	COLOR(BUG1$T1)
FETCH(XYZ)	RAWPROP(F1), CORE(LOC$T1, 1ØØ, 3ØØ)

Store Statements

A store statement moves data from core storage into an ADAM file and has the following form:

STORE(error exit) property designator(tailed label),
block designator

where property designator specifies a nonrepeating group property, tailed label specifies an open file statement or a for group statement, and block designator specifies a block of storage registers.

If the type of the property is not RAW, the value of the property is presumed to be in the first word of the storage block specified by block designator. If block designator is missing, the value is presumed to be in the accumulator.

If the type of the property is RAW, the value of the property is defined by block designator. If block designator is missing, an SHP stream pointer to the value of the property is presumed to be in the accumulator.

ASSIGNMENT STATEMENT

The assignment statement defines a numerical computation. The simplest form for the assignment is a statement which sets the value of a variable equal to the value of an expression; i.e.,

variable = expression

Expressions are constructed from variables, constants, and operators.

Variables

A variable has a value and can appear on either side of an "equals" operator in an assignment statement. The following kinds of variables are defined:

local
file
index
parameter

A variable must have a type associated with it and may also have a unit name or a roll name associated with it.

Local Variables

A local variable must be defined by the program in which it occurs. Local variable references have the following forms:

tailed label (subscript)
tailed label

where a subscript can be an unsigned integer, an index variable, or a parameter variable. In the latter case the parameter variable must be of type LOCATION.

Tailed label specifies a storage location and subscript specifies a full-word address relative to the storage location. Tailed label is equivalent to tailed-label (Ø).

A local variable is defined if any of the following statements is true:

(1) It occurs in a variable declaration.

(2) It occurs in the location field of a data statement.

(3) It occurs immediately to the left of an "equals" operator in an assignment statement.

(4) It occurs in an open area statement.

Consistent multiple definitions are permitted. A type and possibly a unit name or roll name will be associated with the variable in situations (1) and (2). If no type is associated with a local variable, it is assumed to be of type REAL.

Examples:

X$T1 X$T1(1) X$(I) X(P.P1)

File Variables

A file variable is defined by a property roll. File variable references have the following form:

V.property-name (tailed-label)

where property-name is a system name and tailed label specifies an open file statement or a for group statement.

File properties of type LOG have variable type ROLL and file properties of type FP, CFP and INT have variable type REAL. File properties of other types are illegal. The property roll involved may associate a roll name with a file variable of type LOG and a unit name with file variables of type FP, CFP, or INT.

Examples:

V.COLOR (BUG1)

V.'OBJECT NAME' (F1$T1)

Index Variables

An index variable reference must be one of the following single letters:

I, J, K, L, M, or N

Index variables have type LOCATION. The value of an index variable is the contents of an index register as specified in system defined variables.

Parameter Variables

A parameter variable is defined by a parameter declaration. Parameter variable references have the following form:

P.parameter-name

A parameter variable has a type and may have an associated unit name or roll name.

Constants

A constant has a value but can only appear to the right of an "equals" operator in an assignment statement. The following kinds of constants are defined:

- real
- integer
- roll
- location
- string
- function references

Real Constants

A real constant has the same storage structure as a variable of type REAL. It may have an associated unit name and be converted to standard units. A real constant can have the following forms:

real

C.R(real)

real (assignment-unit-specifier)

C.R(real,assignment-unit-specifier)

where real is any string of characters such that

DD(N) ,real

is a legal STRAP instruction.

Real and C.R(real) are called incomplete, since they do not contain a unit specifier; they are completed by the context in which they occur. Real is ambiguous and is resolved by its context.

An assignment unit specifier associates a unit with the real constant. It may be either a unit specifier or have the following forms:

(a) VAR (tailed label)

(b) PAR (label)

(c) FILE (property name (tailed label))

(d) *

In VAR(tailed label) and PAR(label), the unit defined is the unit associated with the variable. In FILE(property name (tailed label)), the unit defined is the unit associated with the file property. The * specifies a scalar.

Examples:

1 C.R.(1)	-.1(*)
1 (MILLIMETERS)	C.R(1,BUG,LENGTH)
1E1Ø(VAR(V1$))	1E1Ø(PAR(P1))
1.(FILE(LENGTH(F1$T1)))	

<u>Integer Constants</u>

An integer constant has the same storage structure as a variable of type INTEGER. An integer constant can have the following forms:

integer

C.I (integer)

where integer is any string of characters such that

(F1Ø)DD(U) ,integer

is a legal STRAP instruction.

Integer is ambiguous and is resolved by context.

Examples:

1 -1E1Ø

<u>Roll Constants</u>

A roll constant has the same storage structure as a variable of type ROLL, and it must have an associated roll name. A roll constant can have the following forms:

element

C.N(element)

C.N(element, assignment-roll-specifier)

where element is an element name to be looked up in a roll (specified by assignment roll specifier in the final form).

Element and C. N(element) are called incomplete, since they do not contain a roll specifier. They are completed by the context in which they occur. Element is ambiguous and is resolved by context.

The assignment roll specifier defines a roll. It may be either a roll specifier or have the following forms:

VAR (tailed label)

PAR (label)

FILE (property name (tailed label))

In VAR (tailed label) and PAR (label), the roll defined is the roll associated with the variable. In FILE (property name (tailed label)), the roll defined is the roll associated with the file property.

Examples:

BLACK C. N(RED)

C. N(LOGAN, OBJ(AIRFIELD))

C. N(BLACK, BUG, COLOR)

C. N(BLACK, FILE(COLOR(F1)))

Location Constants

A location constant has the same storage structure as a variable of type LOCATION, and can have the following forms:

loc

C. L(loc)

where loc is any string of characters such that

XW, loc, Ø, Ø, Ø

is a legal STRAP instruction. If loc does not contain a radix point (.), it is suffixed by 1. Thus C. L(1) and C. L(1.) are equivalent.

Loc is ambiguous and is resolved by context.

Examples:

1 C. L(1.32)

String Constants

A string constant has the same storage structure as a variable of type STRING. It has the form

C. S(string)

where string is either a label or any string of characters, not containing a prime, enclosed in primes.

Examples:

C. S(JOE)

C. S('NOW IS THE TIME')

Function References

The function reference constant is described in the subsection entitled Routines and Functions and has the following form:

F. entry-name (function-argument-list)

Entry-name specifies an entry option defined by the routine file and may be either an entry name or a routine name having no associated entry names. Function argument list is a list of expressions, separated by commas, that are evaluated to provide values for the parameters of the entry option. A particular output parameter belonging to the entry option is selected to be the value of the function reference.

Simple Assignment Statements

Two kinds of simple assignment statements (SAS) are defined:

simple arithmetic assignment statements (SAAS)

simple string assignment statements (SSAS)

The SAAS deals with variables of type REAL, INTEGER, ROLL, and LOCATION. The SSAS deals with variables of type STRING.

Simple Arithmetic Assignment Statements (SAAS)

A SAAS has the form

variable = expression

where expression is any well-formed arithmetic expression involving:

terms (variables and constants)

operators

left and right parentheses.

When executed, the SAAS sets the value of variable on the left-hand side (LHS) of the equals to the value of the right-hand side (RHS) expression.

The RHS expression is defined to be the same type as its left-most term, unless this term has no type (e.g., the ambiguous constant). in which case, the RHS expression is defined to be the same type as the LHS variable.

RHS expressions of type REAL, INTEGER, and ROLL may contain terms and operators of type REAL, INTEGER, and ROLL intermixed. RHS expressions of type LOCATION may contain only terms and operators of type LOCATION. Mixed expressions are illegal. The LHS variable may, however, be of type REAL, INTEGER, ROLL, or LOCATION (i.e., not STRING).

RHS expressions of type REAL, INTEGER, or ROLL are evaluated using normalized floating-point arithmetic. Expressions of type LOCATION are evaluated using (B,25,1) fixed-point arithmetic.

Arithmetic Operators

The following table lists the operators of type REAL, INTEGER, or ROLL:

Operator	Definition	Precedence Class
+x	+x	1
-x	-x	1
x*y	x.y	2
x*-y	x.(-y)	2
x*/y	x.|y|	2
x*-/y	x.(-|y|)	2
x/y	x/y	2
x/-y	x/(-y)	2
x//y	x/|y|	2
x/-/y	x/(-|y|)	2
x+y	x+y	3
x+/y	x+|y|	3
x-y	x-y	3
x-/y	x-|y|	3

The following table lists the operators of type LOCATION:

Operator	Definition	Precedence Class
+x	+x	1
-x	-x	1
x*y	x.y	2
x*-y	x.(-y)	2
x/y	x/y	2
x/-y	x/(-y)	2
x+y	x+y	3
x-y	x-y	3

The precedence class of an operator is used to determine the order in which operations are performed in an expression.

Order of Computation

The order of computation in an expression can be completely specified by using parentheses. A completely parenthesized (CP) expression has one of the following forms:

(LEX $\otimes$ REX)

(LEX $\otimes$ RTERM)

(LTERM $\otimes$ REX)

(LTERM $\otimes$ RTERM)

where LEX and REX are CP expressions, LTERM and RTERM are terms, and $\otimes$ is an operation.

The order of computation for the above forms of a CP expression is the following:

(1) α = REX

β = LEX

$\beta \otimes \alpha$

(2) α = LEX

$\alpha \otimes$ RTERM

(3) α = REX

LTERM $\otimes \alpha$

Expressions that are not CP can be transformed into CP expressions by repeated applications of the following rules:

(1) Operations having a smaller precedence class number are executed before operations having a larger precedence class number. Thus

-A+B*C becomes ((-A)+(B*C))

-A*B+C becomes (((-A)*B)+C)

(2) Within a sequence of operations having equal precedence class operations are executed from left to right. Thus

A*B/C becomes ((A*B)/C)

Incomplete Constants

Forms real and C.R (real) of the real constant are called incomplete, since they do not specify a unit name. Forms element and C.N (element) of the roll constant are called incomplete, since they do not specify a roll name. Incomplete constants that occur in the RHS expression of a SAAS are completed by using the unit name or roll name associated with the LHS variable.

Ambiguous Constants

The forms real, integer, and loc of the real, integer, and location constants are ambiguous. The ambiguity is resolved by using the type of the RHS expression.

The form element, of the roll constant, is ambiguous, since it can be of the form

label

and thus denote a local variable. If label has been defined as a local variable, it is so interpreted, otherwise it is presumed to be a roll constant.

Simple String Assignment Statements (SSAS)

The SSAS has the form

variable = term

where term is a variable or a constant. Both the LHS variable and the RHS term must be of type STRING.

Assignment Vectors

An assignment vector is a generalization of a simple assignment statement.

Arithmetic Assignment Vectors

An arithmetic assignment vector has the form

$variable_1 = \ldots = variable_n = expression \qquad n \geq 1$

and is equivalent to the following sequence of simple arithmetic assignment statements:

$variable_n = expression$

...

$variable_1 = variable_2$

An arithmetic assignment vector has a type and a value which are defined to be the type and value of $variable_1$.

Examples:

A=B=C

A=B=(C+D)+1

String Assignment Vectors

A string assignment vector has the following form:

$variable_1 = \ldots = variable_n = term \qquad n \geq 1$

and is equivalent to the following sequence of simple string assignment statements:

$variable_n = term$

....

$variable_1 = variable_2$

Compound Assignment Statements

A compound assignment statement has the following form:

$vector_1, \ldots, vector_n \qquad n \geq 1$

and is equivalent to the following sequence of assignment vectors.

$vector_1$

...

$vector_n$

A compound assignment statement has a type and a value which are defined to be the type and value of $vector_1$.

Examples:

A=B=C, I=I+1

Generalized Assignment Statements

A generalized assignment statement is a generalization of a compound assignment statement. Generality is obtained by replacing the re-defining term as follows:

term: variable

constant

(compound assignment statement)

Examples:

A=B*(C=D=1, B=2)

BRANCH STATEMENTS

Statements are normally executed in the sequence in which they appear in the symbolic routine. Branch statements interrupt the normal flow of control, and then can be conditional or unconditional.

Unconditional Branch Statements

The unconditional branch statement always transfers control to a specified statement and has the following form:

GO TO statement

where statement specifies the next statement to be executed and can have the following forms:

tailed-label

tailed-label (subscript)

Tailed-label defines the next statement to be executed. In tailed-label (subscript), tailed-label defines a full-word branch table and subscript defines the full-word branch instruction to be executed. Subscript can be an integer, an index variable, or a parameter variable of type LOCATION.

Conditional Branch Statement

The conditional branch statement transfers control to be a specified statement if a specified condition is satisfied. It has the following forms:

IF condition THEN statement

IF condition THEN statement_1 ELSE statement_2

where statement defines a statement and condition has the form

relation (left-term, right-term)

The condition is satisfied if left term has the specified relation to right term. There are two kinds of relations: arithmetic relations; and string relations.

The following arithmetic relations are defined:

EQ	equal to
NQ	not equal to
GR	greater than
GQ	greater than or equal to
LS	less than
LQ	less than or equal to

If arithmetic relations are used, left term and right term can be arithmetic expressions.

Examples:

```
IF EQ (I, 1) THEN A1
IF NQ (I+1, J-1)  THEN A1
IF EQ (COLOR (I) , RED(COLOR(I)))) THEN A1
```

The following string relations are defined:

ID   identical to

NID   not identical to

If string relations are used, left term and right term must be either variables or constants of type STRING.

Examples:

ID (S1, C.S. (XYZ))   THEN A1

Entry Branch Statements

The entry branch statement is used to branch to the various entry options of a routine and has the following form:

ENTRY BRANCH

DAMSEL and entry declarations illustrate the use of the entry branch statement.

Enter Statement

The first statement executed in a routine should normally be an enter statement which has the form:

ENTER

and which preserves pertinent machine registers.

Exit Statement

The last statement executed in a routine should be an exit statement which can have the forms:

EXIT

EXIT (ERROR)   integer

EXIT specifies a normal exit. EXIT (ERROR) integer specifies an error exit, and integer .Ø is placed in VF ($13). The exit statement restores the machine registers saved by the enter statement and returns to the calling routine.

ROUTINES AND FUNCTIONS

Routines that have been added to the ADAM routine file may be called for execution by other routines. This can be done in a DAMSEL routine by using CALL or DO statements or by using function references in assignment statements.

Each routine in the routine file has a name and may have an associated set of entry names (entry declaration). Each entry name has an associated parameter configuration (parameter declaration).

A routine is called for execution by specifying an entry name (or a routine name if it has no associated entry options) and a set of suitable values for the parameters involved. Expressions may be used to supply values for input parameters and are evaluated before the entry option is executed. Variables may be used to supply values for output parameters and specify locations for results of the entry option.

Call Statement

The call statement is used to execute an entry name option and has the following form:

CALL (error exit) entry name (argument$_1$, . . . , argument$_n$) $n \geq 0$

where entry name is a label specifying the entry option, argument defines a parameter value, and error exit specifies the next statement to be executed, if executing entry name yields an error return.

The parameters associated with the entry name were defined and ordered by the entry declaration. By convention, argument$_1$ specifies a value for the first parameter, etc. The argument count of the call statement must agree with the parameter count of the entry declaration.

Input parameter values may be specified by expressions that are evaluated before the entry option is executed. Output parameter values may be specified by variables and specify locations for the results computed by the entry option.

Example:

P. P1 = expression_1, P. P3 = expression_3

. . . . .

CALL entry-name (P. P1, P. P2, P. P3, P. P4, . . .)

variable_2 =P. P2, variable_4 =P4

. . . . .

Ambiguous constants (incomplete constants) occurring in expressions defining values for input parameters may be resolved (completed) by the type of the parameter.

The examples that follow are based on the routine shown below:

```
ROUTINE    R1
INPAR      P1, ROLL(BUG, COLOR)
INPAR      P2, STRING
OUTPAR     P3,ROLL
OUTPAR     P4, STRING
ENTPAR     E1(P1, P3)
ENTPAR     E2(P2, P4)
```

Example:

```
X1    OPEN BUG (FLY)
      CALL E1 (BLACK, COLOR (X1) )
```

Example:

```
VARIABLE     STRING/V2/
CALL(ERR)    E2(C. S. (BUG), V2 (I) )
```

Function Reference

A function reference is a constant that can be used to form expressions in assignment statements, call statements, and conditional branch statements. A function reference has the following form:

F. entry name (argument$_1$, . . . , argument$_n$)

where entry name is a label specifying an entry option and argument defines a parameter value. The description of the function refrence is identical to that of the call statement except that a function reference has a value. Any output parameter associated with the entry can be selected as the value of the function reference by writing the single character $ as its argument rather than specifying a variable. All other output parameters must have variables (specifying locations for results) as their values.

The examples that follow are based on the routine shown below:

```
ROUTINE     SINCOS
INPAR       P1, REAL
OUTPAR      P2, REAL
ENTPAR      SIN(P1, P2)
ENTPAR      COS(P1, P2)
...
MEND
```

Examples:

```
X1=F. SIN(1, 32, $)+1
X2=F. SIN(F. COS(1. 32, $), $+1
```

If the output parameter selected for the function value is the right-most parameter of the entry, then the $ value designator may be omitted. The above examples then become:

```
X1=F. SIN(1, 32)+1
X2=F. SIN(F. COS(1. 32) )+1
```

DO Statement

The standard ADAM routine linkage is as follows:

$13 = location of input parameters

$14 = location of output parameters

$15 = entry point number

SIC, RETURN $A

B, routine

B, error exit

B, normal exit

The DO statement is used to execute an entry option using standard ADAM linkage. It has the following form:

DO(error exit) entry name(input location, output location)

where entry name is a label specifying the entry option, error exit specifies the next statement to be executed if executing entry name yields an error return, and input location and output location specify the location of the input parameters ($13) and the output parameters ($14) respectively. They may be any string of characters that are acceptable to the MPOINT macro.

MPOINT, $13, input location

MPOINT, $14, output location

DATA STATEMENTS

Data statements are used to specify initial values for variables. Every kind of type specifier yields a data statement having the form:

$label type specifier $datum_1, \ldots, datum_n$ $n \geq 1$

where label is optional.

The data statement is a declaration for the variable label and as such must be consistent with any other declarations. The initial value for the subscripted local variable

label (Ø)

is $datum_1$.

Real Data Statement

A real data statement defines constants of type REAL and has the following forms:

REAL $real_1, \ldots, real_n$

REAL(unit specifier) $real_1, \ldots, real_n$ $n \geq 1$

where real is any string of characters such that

DD(N), real

is a legal STRAP instruction, and unit specifier specifies a unit name.

Each real constant is converted into standard units from the units specified by unit specifier.

Examples:

REAL +1.Ø, -2

REAL(MILES) 5.7E1Ø

Integer Data Statement

An integer data statement defines constants of type INTEGER and has the form:

INTEGER $integer_1, \ldots, integer_n$ $n \geq 1$

where integer is any string of characters such that

(F1Ø)DD(U), integer

is a legal STRAP instruction.

Example:

INTEGER 1, 3, 7, 127

Roll Data Statement

A roll data statement defines constants of type ROLL and has the following forms:

ROLL element$_1$, . . . element$_n$

ROLL(roll specifier) element$_1$, . . . , element$_n$ $n \geq 1$

where element is an element name.

In ROLL (roll specifier) element$_1$, . . . , element$_n$ $n \geq 1$, element is looked up in the roll specified by roll specifier, and the value of the constant is its PV. If ROLL element$_1$, . . . , element$_n$ is used, the statement must have a label in its location field and the variable label must have an associated roll name.

Examples:

```
ROLL(BUG,COLOR)    BLACK,BROWN
XYZ  ROLL    BLACK,BROWN
        VARIABLE  ROLL(BUG,COLOR)/XYZ/
```

Location Data Statement

A location data statement defines constants of type LOCATION and has the form:

LOCATION loc$_1$, . . . loc$_n$ $n \geq 1$

where loc is any string of characters such that

XW, loc, Ø, Ø, Ø

is a legal STRAP instruction. If loc does not contain a radix point (.), it is suffixed by one.

Example:

```
LOCATION   1,-1,3,-3.
```

String Data Statement

A string data statement defines constants of type STRING and has the form:

STRING $string_1, \ldots, string_n$ $n \geq 1$

where string is either a label or any string of characters, not containing a prime, which is enclosed in primes.

The value of the string constant is a pointer having the form:

XW, location, bit-count, Ø, Ø

which points to string.

Example:

STRING ABC, 'NEW YORK', DEF

Switch Data

A switch data statement defines a branch table consisting of full-word branch instructions of the form:

BE, address;BE, Ø

It has the form:

SWITCH $address_1, \ldots, address_n$ $n \geq 1$

where address is a tailed label. Switch vectors are used by branch statements.

Example:

SWITCH A1$TAIL, A2, A3(I), A4($6)

SECTION V

STRING SUBSTITUTION

The string substitution capability in ADAM allows a user to extend the language of the input messages. By defining an association between a single new word and a string of characters, that word is added to his input language. Upon each subsequent use of the word in an input message, the word is replaced by the string of characters before the message is translated.

For example, the definitions

LET BOSTON MEAN (CITY BOSTON). ¦*

and LET BOS MEAN (CITY BOSTON). ¦

define the two keywords BOS and BOSTON to be associated with the string CITY BOSTON. Subsequent use of either BOSTON or BOS in an input message will result in the string CITY BOSTON being substituted before translation. Thus, both

FOR BOSTON...

and

FOR BOS...

would be transformed into

FOR CITY BOSTON...

before translation.

String substitution definitions define either explicitly or implicitly the input devices on which they are to be effective. Thus the keyword BOSTON might be defined differently for different input devices. The definition of a

---

* ¦ means end of message.

string substitution allows the specification that parameters from the input message be inserted into the defined string. A keyword in a definition may not contain internal blanks or any punctuation characters.

DEVICE DEPENDENCY

The definition

LET BOSTON MEAN (CITY BOSTON). ⌶

defines BOSTON as a keyword only for the input station (typewriter, printer, display) at which the definition was made. A keyword may be made effective for other input stations by a definition of the form

LET BOSTON MEAN (CITY BOSTON) FOR T1 T2.⌶

which defines BOSTON for the stations associated with devices T1 and T2. If _any_ device is named explicitly, no implied definition is made for the device at which the definition is entered; in other words, the above message entered at T3 would not define BOSTON for T3, only for T1 and T2.

The special case ALL defines a keyword for all input devices, as in

LET BOSTON MEAN (CITY BOSTON) FOR ALL. ⌶

OPERATION

The substitution operation proceeds as follows: Each word defined in a message of the form

LET ··· MEAN (...) .⌶

is called a keyword. When it is defined, the keyword and its substitution are placed in a file of substitutions, organized by device for which the keyword is effective. The file has one entry for each device and an entry for ALL devices.

When a subsequent input message is processed it is first scanned to separate words and punctuation (see paragraph entitled Punctuation and Separation, below). After separation, the substitution scan begins with the first word of the message.

Each message word is compared against two lists:

(1) The list of keywords defined for ALL devices

(2) The list of keywords defined for the device at which the message was entered

As soon as a keyword is found, scanning stops and the string associated with the keyword is inserted into the message in place of the keyword.

Scan Option

Unless otherwise specified in the original definition, scanning for keywords then continues at the word in the message following the keyword for which a substitution was just made. For example, if the definition

LET BOSTON MEAN (CITY BOSTON).

were made, and the message

FOR BOSTON . PRINT POPU.

were entered, the scan would proceed to identify BOSTON as a keyword, substitute the string to form

FOR CITY BOSTON . PRINT POPU.

and continue scanning at the period following the phrase CITY BOSTON.

Resumption of scanning at the word following the keyword for which a substitution is made is called the SCAN option, in contrast to the RESCAN option described below. The two definitions

LET BOSTON MEAN (CITY BOSTON). ⧘

and

LET BOSTON MEAN (CITY BOSTON) USING SCAN. ⧘

are equivalent.

Rescan Option

The RESCAN option specifies that after substitution for a keyword is made, scanning is to continue from the beginning instead of the end of the inserted string. For example:

LET BOSTON MEAN (CITY BOSTON) USING RESCAN. ⧘

is applied to the message.

FOR BOSTON.PRINT POPU. ⧘

would cause the first substitution

FOR CITY BOSTON.PRINT POPU.

with scanning resumed at the word CITY. The word BOSTON would again be identified as a keyword, with the result that the process would be repeated infinitely, producing

FOR CITY CITY CITY . . .

finally terminated by an error message (see Appendix V).

Suppose, however, that definitions were made as follows:

LET BOSTON MEAN (CITY BOSTON). ⧘

LET BOS MEAN (BOSTON) USING RESCAN. ⧘

Then the message

FOR BOS.PRINT POPU.

would first become

FOR BOSTON. PRINT POPU.

and then scanning would resume at the word BOSTON, producing

FOR CITY BOSTON. PRINT POPU.

PARAMETERS

To insert parameters, a user selects in the definition the INSERT or REINSERT option and indicates where parameters are to be inserted in the string. A parameter is a single word, a single punctuation character, or a sequence of words and punctuation enclosed in a pair of parentheses. For example, the definition

LET WHAT MEAN (FOR CITY /4/ . PRINT /2/) USING INSERT.

defines the keyword WHAT and indicates two insertions: The fourth parameter follows WHAT in an input message, and the second parameter follows WHAT in an input message. The notation /n/ in a definition indicates that the nth parameter following the keyword is to be inserted in place of the /n/ in the string. Thus, given the above definition of WHAT, the message

WHAT IS POPU OF CHICAGO.

would be transformed to

FOR CITY CHICAGO. PRINT POPU.

before translation and the message

WHAT IS LATITUDE OF WASHINGTON

would be transformed to

FOR CITY WASHINGTON. PRINT LATITUDE.

before translation.

When the INSERT or REINSERT options are specified in a definition after parameters are inserted and before scanning resumes, all parameters up to and including the highest numbered parameter specified in the definition are removed from the message. Thus "filler" words may be introduced. For example, with the above definition of WHAT, the fourth parameter is the highest number specified in the definition. Thus, the message

WHAT IS POPU OF CHICAGO.

first becomes

FOR CITY /4/ . PRINT /2/ IS POPU OF CHICAGO.

in which

IS	is parameter 1
POPU	is parameter 2
OF	is parameter 3
CHICAGO	is parameter 4
	is parameter 5.

After parameter insertions are made, four parameters are removed and parameters 1 and 3, although not used, are removed from the message. Parameter 5, the period in this example, remains, as would any higher numbered parameters.

Note that a parameter is a single word, a single punctuation character, or a sequence of words and punctuation enclosed in parentheses. In the example thus far, the message

WHAT IS POPU OF NEW YORK.

would not work, but would produce the message

FOR CITY NEW. PRINT POPU YORK.

because NEW and YORK are separate words and therefore separate parameters.

If a sequence of characters is enclosed in parentheses and used as a parameter, the parentheses are removed and the sequence of characters is inserted as a single parameter. Thus the message

WHAT IS POPU OF (NEW YORK).

is properly transformed to

FOR CITY NEW YORK. PRINT POPU.

Insert Option

The INSERT option, expressed as

LET ... MEAN (....) USING INSERT.

indicates that parameter specifications of the form /n/ are to be interpreted and also that scanning is to resume at the first word after the highest numbered parameter specified in the definition. For example, if the definition were

LET WHAT MEAN (FOR CITY /4/ . PRINT /2/ .) USING INSERT.

the message

WHAT IS POPU OF (BOSTON, CHICAGO) SORT ASCENDING.

would first be transformed to

FOR CITY BOSTON, CHICAGO. PRINT POPU. SORT ASCENDING.

after which, substitution scanning would resume at the word SORT.

Reinsert Option

The REINSERT option, expressed as

LET....MEAN (....) USING REINSERT.

operates like INSERT operates, except for the point at which scanning resumes. With REINSERT, scanning resumes at the first word introduced into the message by the substitution.

CAUTIONS

Messages Not Subject to Substitution

Only messages which pass through the SUBSCAN operation in ADAM are subject to substitution. At present, all messages sent to the translator (i.e., messages in FABLE and IFGL) are subject to substitution. Other messages, such as utility language messages ($TELL, $TIME, etc.) are not subject to substitution. In particular, string substitution definition messages (those which begin with the words LET or SCRUB) are not themselves subject to substitution.

Punctuation and Separation

When a message is processed, the first scan involves separating words from punctuation and punctuation from adjacent punctuation by a single space. Spaces are introduced where necessary, and eliminated where redundant. For example, a blank or space is represented as ▿ in the following message as originally typed:

FOR▿▿CITY.▿PRINT▿▿TITLE▿'▿▿▿CITY▿POPULATION▿▿▿'▿POPU.

After it is scanned, it looks like this:

FOR▿CITY▿.▿PRINT▿TITLE▿'▿▿▿CITY▿POPULATION▿▿▿'▿POPU.

Some spaces have been eliminated and some added. Punctuation and spaces are undisturbed between primes (').

Although string substitution definitions are not themselves subject to substitution, they are scanned by the separation scan.

Because of the separation scan and the handling of parameters, the following statements hold:

(1) Characters may not be concatenated into single words as a result of separate substitutions. For example, no substitutions for DAY and MONTH can cause the single word 23MARCH to be produced in a message.

(2) Numbers (or words) with internal punctuation are treated as separate parameters unless enclosed in parentheses. For example,

23.45

is separated by the separation scan into

23 . 45

and is treated as three parameters for the INSERT or REINSERT options. To be treated as a single number, it must be enclosed in parentheses.

(3) If the notation /n/ for parameter insertion appears between primes, it must be written as

▽/▽n▽/▽

where ▽ signifies exactly one space. The separation scan, which normally provides such spacing, does not operate on characters between primes, but the substitution scan expects the above spacing. For example, a string containing the phrase:

...TITLE 'SUMMARY OF /4/ /5/' ...

must be written:

...TITLE 'SUMMARY OF ▽/▽4▽/▽/▽5▽/▽' ...

DEFINITION AND SYNTAX

String Substitution Definition

The form of a string substitution message is as follows:

$$\text{LET } \underline{\text{AN}} \text{ MEAN (string) } \left[\text{FOR} \begin{Bmatrix}\text{ALL} \\ \underline{\text{DN}}\end{Bmatrix} [\underline{\text{DN}}\ldots] \right] \left[\text{USING} \begin{Bmatrix}\text{SCAN} \\ \text{RESCAN} \\ \text{INSERT} \\ \text{REINSERT} \\ \underline{\text{RT}}\end{Bmatrix}\right] [\cdot]$$ ⌶

Here, words in capital letters and not underlined are required, { } indicate alternate choices, [ ] enclose optional notation, and underlined abbreviations and lower case words are defined as follows:

AN — Alphanumeric string. The string may not be the same as any other keyword already defined for the same device or for ALL. It should not be LET if the keyword will ever be the first word of a message (see below).

String — Any sequence of characters. The string is terminated by the occurrence of one of the following sequences:

) FOR
) USING
) .
) ⌶

DN — Device name.

RT Routine name or entry-point name. The user may define his own routine to perform any kind of string insertion he wishes.

If no device is specified, it is assumed to be the device on which the message was entered. If no routine or option is specified, it is assumed to be SCAN.

String definition messages are not translated by the translator, but first undergo the scan described in the paragraph Punctuation and Separation, above. Whenever a message begins with the word LET, it is handled separately and treated as a string substitution definition. For this reason, each string substitution definition must be a separate message and may not be combined with FABLE statements or other string substitution definitions in one message.

Removing String Substitutions

Messages to remove string substitutions take the following form:

SCRUB AN [FROM {ALL | DN [DN...]}] [.]

SCRUB messages must also be separate messages. For purposes of string definition and removal, ALL is considered a separate device. This means that a string substitution defined FOR ALL may only be scrubbed by FROM ALL. String substitutions for specific devices must be scrubbed from those devices specifically. If no device is specified, the string substitution is deleted from the list of substitutions for the device which sent the message.

SUBTABLE FILE

String substitutions are stored in the SUBTABLE file and the corresponding keywords are in the KEYWORD roll. Hence, a user may save this file and

roll, along with his data base, using DABS or USAVE after he has defined those strings he wishes to be a part of his operating system.

SECTION VI

OUTPUT FORMATTING

Output formatting in ADAM is the process of rearranging an ADAM file in an order appropriate for output, translating names from the internal encoded form to alphabetic form, and sending the resulting messages to an output scheduling program for actual output. The entire process is directed by a format specification or a format selected from among the formats in the ADAM format file. This section describes a method for writing formats, i.e., for creating a format specification suitable for inclusion in the format file.

A format specification is a procedure for transforming data in a file into a form for output. Formatting consists of executing the format operators in the format specification. Data on which the format operates may be values from a file; names of properties and the name of the file itself; literal strings, such as headings, in the format itself; and the input message, time, date, and a title. A user may specify routines to be executed to convert data in any way, provided the property values are numeric. Graphs and vectors may be generated by specifying routines.

In order to be referenced and executed, a format must be in the ADAM format file. The user must compile his format specification and add it in binary form to the format file through a system routine called FORUP. Exact procedures are described in the documentation on ADAM system operating procedures.

A format suitable for inclusion in the ADAM format file is prepared as a deck of binary cards. Compilation by the SMAC compiler of a set of macro statements will produce such a binary deck; the macro for this purpose is called MOF. A format to be compiled consists of a series of MOF statements, with STRAP coding intermixed if desired:

MOF, op, op,......

MOF, op, op,......

Any number of ops can appear in a statement with the same effect as if they had appeared in separate MOF statements, with a few exceptions described in this document, thus:

MOF, op_1 op_2,....

has the same effect as

MOF, op_1

MOF, op_2........

Each format must begin with a MOF, BEGIN statement and end with a MOF, END statement followed by a MEND statement (required by SMAC). Further details on the use of MOF are explained under the various ops.

BASIC PRINCIPLES

An Example

The statement

MOF, BEGIN (F5), TIME, S, *Q, Q(1Ø), N(1Ø), V(5), N(1Ø), V(1Ø), /, S(9), Q*, END

is a simple but complete format which says:

Begin a format named F5	BEGIN (F5)
Print the time	TIME
Space over one character	S
Begin a loop through all object in the file	*Q
For each object,	
Print the object name in ten columns	Q(1Ø)
Print a property name in ten columns	N(1Ø)

Print the value of a property in five columns	V(5)
Print the next property name in ten columns.	N(1Ø)
Print the next property value in ten columns	V(1Ø)
Move to the left margin of the next line	/
Skip nine character spaces	S(9)
Get the next object in the file, if any, and repeat everything after the *Q	Q*
When all objects are finished,	
End formatting and send out the formatted output	END

For a sample city file with two objects, BOSTON and CHICAGO, and two properties, the formatted output would appear as:

```
12:34:56 BOSTON   COUNTRY  USA  POPULATION   731ØØØ
         CHICAGO  COUNTRY  USA  POPULATION  3621ØØØ
```

Which File

Various input messages may cause output to be formatted, frequently as a result of querying a file. The result of a query is usually an output file, and it is this output file that is formatted, not the file being queried.

Output Devices

The formatting program automatically handles differences between output devices to which the output is directed, including margins, pagination, and so forth. Some operations, such as *-- (draw a vector) are restricted to display-type devices. The entire format is re-done for each different device type specified. A special format operation BT (branch on device) allows different segments of a format to be used for output going to different devices (typewriter, printer, CRT, SPR).

Printed vs Display Output

Character output may be sent to any device, and is always referred to as printed output, although it may actually be displayed, typed, or sent to some other device. The formatting program keeps account of the location of the next field into which printed output is to be placed. The location for display output (points and vectors) on the other hand, is explicitly specified in the format itself and is always described relative to the lower left corner of the display.

Next Field

Formatting of printed output begins at the upper left corner of the first page. Each field to be output begins at the character immediately to the right of the preceding field, unless a formatting operator (such as /, go to the next line) changes the location of the next field.

Next Object

Iteration through all objects in a file is controlled by the pair of operators *Q and Q*. Objects are treated in order and formatting of one object does not begin until formatting of the last object is complete. Individual objects may not be selected for formatting.

Next Property

Property names and values are treated in order. Thus, in the example, N(1∅) means print the next property name, and V(1∅) means print the next property value. Property names or values may be skipped over by specifying "print in a zero-width field," e.g., N(1∅). A special operator, RC, resets the format to begin at the first property again, but skipping backward over properties or values is not provided. Operators *P and P* define an iteration over all properties in a file.

CATEGORIES OF OPERATORS

Print Operators

Print operators specify that character fields be output. Each operator specifies explicitly or implicitly a field length (number of characters) in which the material will be printed and implies the positioning of the next field. Examples:

N - print property name

V - print property value

LIT - print characters literally specified

TL - print title supplied.

Spacing Operators

Spacing operators specify a redefinition of the location of the next field for printed output. The operator S (skip) means skip a specified number of spaces and is thus similar to a print operator. The other spacing operators define the various positions of the next field with respect to the page margins. Examples:

/ - next line or carriage-return-line-feed

NXP - next page

Display Operators

Display operators are effective only when the output device is a display. They define vector or point outputs or specify material to be recorded for later use with light-pen inputs. Display operators do not change the definition of the next field location.

Examples:

*-- display vectors

*.. display points

--RM vector to right margin

Iteration Control Operators

Iteration control operators specify the beginning and end of a set of other operators to be used repeatedly. Any operator except BEGIN or END may be used within the scope of an iteration.

Examples:

...,*Q,Q,Q*,... for each object, print the object name

...,*L(3),/,TIME,L*... print the time on three successive lines

Mode-Setting Operators

Mode setting operators specify a mode of formatting to remain effective until unset. Any operator except BEGIN and END may appear within the scope.

Examples:

...,*TRU,V(5),V(1Ø),TRU*,... truncate the output for the two property values specified if it exceeds five or ten columns respectively.

Margin-Definition Operators

Margin definition operators define the right, left, and bottom margins of the output page, or reset the margins to the next previous definition. Margin settings are in effect until an unset operator occurs, at which time they revert to their previous setting. Margin settings may be changed at any point in a format.

Examples:

*RM(1∅) - set the right margin ten spaces to the right of the present next field position.

RM* - set the right margin to the setting it had before the last *RM operator still effective.

Special Routines

Special routine operators are available to cause the loading and execution of user-defined subroutines (to perform special conversions, for example) during formatting.

Examples:

VC - execute a routine and deliver to it the next file property value.

DO - execute a routine.

Miscellaneous Operators

Examples:

ROW, COL, RAW } define format type

...,*SH,...,SH*,... the operators between these two are special header material to be put on every page of output.

BT branch or device type (printer, typewriter, display or SPR.)

Macro-Control Operators

Macro control operators do not appear in the eventual output, but affect the operation of the macro MOF.

Examples:

....ST(XW, 1.)...	insert the STRAP statement XW, 1. at this point in the compilation of this format

FORMAT TYPES - COL, ROW, AND RAW

Format type specification affects the handling of right-margin overflow and of the positioning of the names, values, and units associated with repeating properties. The three format types COL, ROW, and RAW produce exactly the same output if

(1) the material being formatted never overflows the right margin

(2) no repeating-group property is part of the eventual output

The format is treated as RAW if no format type is specified. The type specified for a format may be changed in the middle of the format specification string, if desired.

The details of repeating group handling and of pagination and margins are discussed in their respective sections below. Briefly, the format types differ as follows:

COL — When the file being formatted contains material which, when output, would overflow the right margin, an automatic "/" (next line) operation is generated by OUTFOP.

Successive repetitions of repeating groups are aligned below one another on successive lines. For this reason, only the first of equal repeating group names or repetition names which are formatted below one another, is printed. Each time a new repeating group is encountered, the left margin is moved back after all the material in the group has been formatted.

ROW — Material that would overflow the right margin is formatted as if it were on a page immediately to the right. A collection of pages

in the vertical direction is called a section. All material in the first section is output before any in succeeding sections, and sections are output in order from left to right.

Repeating groups are treated the same as in COL formats except that repeating group material is not indented.

RAW — Right margin overflow is treated in the same manner as in COL formats.

Repeating group instances are neither aligned with one another nor indented.

FIELD DEFINITION AND FIELD OVERFLOW

During its operation, the formatting program maintains a set of coordinates which describe the character location at which the next field will begin. These coordinates are in the form

section/page/line/column, abbreviated as s/p/ℓ/c

At a BEGIN statement, coordinates are set to

Ø/Ø/Ø/Ø.

As print operators or spacing operators are encountered, the coordinates are changed to reflect the new position of the next field. For example,

N(1Ø) — in addition to printing a property name, changes

s/p/ℓ/c

to

s/p/ℓ/(c+1Ø)

presuming that the name to be printed does not exceed ten characters and there is no right-margin overflow.

The print operators specify a field width in which the material is to be printed, either as a number of characters or as "variable," which means "as much as is required." In either case, it is possible that the field is too small to fit the output or the field crosses a right margin, in which cases the formatting program automatically adjusts the output as described below.

Deleted Names and Values

If a print operator specifies a name or value from a file and that name or value is deleted or non-existent, a field of hyphens, the size of the defined field is printed. If the field is defined as variable, five hyphens are printed. The operator NPDV will suppress the printing of hyphens and leave spaces instead.

Variable Width

A variable width field is specified by omitting the field width, thus:

N(1Ø) specifies print a property name in a ten character field.

N specifies print a property name in a variable field, (which may differ for each property name).

Note that N(Ø) specifies print a property name in a zero-width field (i.e., skip over it) and thus differs from N.

For names and for logical and raw valued properties, variable width is exactly the number of characters required to accommodate the name, logical value, or raw value.

For integer property values, variable width is the number of characters required to accommodate the current value plus one character for the sign (+ or -).

For floating point property values, variable width is thirteen characters.

Floating Point Property Values

Floating point property values are printed either as integers or in floating point form, depending on the field width specified in the format and the defined maximum and minimum* for that property.

Floating point form is

$$\pm d.ddddd(\pm ee)$$

in which d.ddddd is a decimal number and ee is the power of ten to which that number is to be raised to give the value. For example, -1.ØØØØØ(+Ø2) is a floating point form for -1ØØ.

In a specified field, enough room is allowed to print the maximum or minimum as an integer. The value actually printed is aligned in that field. Floating point property values are converted to decimal fractions if their defined maximum or minimum (whichever is larger in absolute value) will fit within the defined field when converted to a decimal fraction, with its sign, and truncated to an integer. For example, suppose a floating property has the value 123.4. Then the following table shows how its formatting depends upon its defined maximum and minimum and on the field width specified:

maximum	+9999.	+9999999999999999.
minimum	Ø	Ø
number of characters in maximum (with sign)	5	17
V(4)	+1.2 34ØØ (+Ø2)	+1.2 34ØØ (+Ø2)

---

* The maximum and minimum values for a property are defined in the property roll.

V(5)	␣␣123	+1.23 4ØØ(+ Ø2)
V(13)	␣␣123.4ØØØØØØ	+1.234ØØ(+Ø2)
V(17)	␣␣123.4ØØØØØØØØØØ	␣␣␣␣␣␣␣␣␣␣␣␣␣␣123
V	␣123.4	+1.234ØØ(+Ø2)

Field Underflow

If a set of characters to be output is shorter than the field in which it is to be formatted it will be

(a) right adjusted if a numeric property value or repetition number, and

(b) left adjusted otherwise.

This handling may be overridden by the operators *RA,...,RA* which cause all fields to be right adjusted.

Field Overflow

If a set of characters to be output is longer than the field in which it is to be formatted, it is continued in a field immediately below and the same length. Thus LIT(5), ABCDEFGHIJK produces

```
ABCDE
FGHIJ
K
```

At the conclusion of formatting this field, the "next field" is set to the character and line immediately to the right of the originally defined field; i.e., immediately to the right of the E in the example. However, subsequent / operations, explicit or implicit, will move to the first unused line. For example, if the file had property values 7, 5, and 3 digits long respectively, the format segment

V(5), S, V(5), /, V(3)

would produce

```
'
'
'
'          12345 12345
'          67
' 123
'
'
```

This handling may be overridden by the operators *TRU, . . . , TRU* which cause all data that would overflow a field to be truncated.

Right-Margin Overflow

A field is never formatted across a right margin. If a field specification would cause a field to cross a right margin, a dummy blank field is inserted to bring the coordinates exactly to the right margin. If the format is ROW (or *F is in effect) the next field appears in the next section. If the format is COL or RAW, a dummy / (next line) operator is performed. The operators *TRU, . . . , TRU* cause all data that would overflow the right margin to be truncated.

MARGINS, PAGINATION, AND HEADINGS

Margins

The operators *RM, *LM, and *BM set the right, left, and bottom margins of a page to a position relative to the current setting of the "next field" coordinates. The operators *H and *SH define heading material and thereby define by implication the top margin.
Thus

*RM(59), *LM(40)

defines a page 20 columns wide which begins 40 columns to the right of the current "next field".

In other words, given coordinates

s/p/ℓ/c

the margins would be set to c+40 and c+59. The columns defined as margins are included in the page; thus in the example c+40 is the first column of the page and c+59 the last.

The entire format is reinterpreted for each different device type. Each time, margins are set (at the BEGIN operator) to the maximum available for that device. When a margin setting operator is encountered, the right and bottom margin settings used are the smaller of the margin settings and the maximum margins allowed for that device (see Appendix VII).

Each time a margin setting operation changes the margin setting, the old setting is saved in a push down list. The unset operations then move the margins to the next previous setting; e.g.,

MOF, BEGIN, *RM(4Ø), *RM(2Ø), RM*

leaves the right margin set at 4Ø.

Since the margins will frequently differ, depending on the destination of the output, the same format could produce output which looked significantly different on different devices.

Pagination

Devices are of two types, pageable and nonpageable. An SC3Ø7Ø printer, for example, is nonpageable, with output continuous in the vertical direction. The SPR (off-line system printer), on the other hand, is pageable and the formatting program automatically moves to the next page (and prints the page number if specified) when output overflows the bottom margin. Page-turning on the displays, which are also pageable devices, is under operator control.

For COL and RAW formats, pages normally follow one another vertically in order. For ROW format and for the special operators *F,...,F* with the other formats, page handling is modified to allow horizontal pages, or sections. The principle is that a ROW format may describe output which is wider than a single page and the formatting program will take care of assigning the output to physical pages. A format declared ROW may be written so that pages continue as far right as desired. During formatting, such material is assigned to vertical pages and horizontal sections.

	Section ∅	Section 1	Section 2	. . .
logical page ∅	physical page 1	physical page 3	physical page 5	. . .
logical page 1	physical page 2	physical page 4	physical page 6	. . .

Page numbers are assigned, starting with page 1, to all the pages of Section ∅ in order, then all the pages of Section 1, and so forth. Material is output in order of physical page number. Since physical page width differs from one device to another, the same material will be sectioned differently on different devices.

Headings

Material may be defined with the *H,...,H* operators to appear at the top of each logical page, or with the *SH,...,SH* operators to appear at the top of each physical page. For outputs consisting solely of vertical pages (as with COL type formats), *H and *SH are equivalent. The operators *MD,...,MD* similarly define material to be repeated at the left margin of each section.

The *H and *SH operators implicitly define a top-margin setting which can be overridden only by the operator IO.

The *MD operation defines a left margin only for the duration of the current line.

Page Numbers and Classification — Implied Top Margin

If a file being output has a classification other than UNCLASSIFIED, classification will be automatically printed at the top and bottom center of every page and the page size reduced by four lines. If page numbering is requested with the PP operator, page numbers will be printed at the top right of every physical page and the page length reduced by two lines.

FILE DATA

Objects and Properties

The operators *Q and Q*, which may be read "For each object", in reality cause a file to be opened (i.e., brought into memory) and are more properly described as "open file" and "step file" respectively. Without these operators, no file data is available; therefore, the operator V, even in the form V(∅), may appear only within the scope of *Q...,Q*.

Standard Properties

Every ADAM file contains a number of standard properties (e.g., DEAD SPACE BIT COUNT, ALTERNATE CLASSIFICATION). Of these, only Standard Classification and Alternate Classification are available to be formatted. If not desired, they must be skipped with N(∅), V(∅) operators.

Repeating Groups

Repeating group names, values, and units are handled specially; therefore the interpretation of an N, V, or U operator depends upon whether the property being formatted is within a repeating group or not, or is itself a repeating group or not. Repeating group formatting also depends on format type, but the following rules pertain, independent of format type.

(a) When successive N, V and/or U operators have exhausted the names, values and/or units of a repetition, subsequent N, V and/or U operators will format the properties of the repeating group again for subsequent repetitions until all repetitions have been formatted. After this, succeeding properties not within the group will be formatted.

(b) The value of a repeating group property is the repetition name; i.e., the operators N, V when applied to a repeating group property will format the name of the group and the name of the current repetition. If the group is not named, the formatting program will supply a repetition number instead of a name.

Alignment of Repetitions

For format types ROW and COL, the repetitions of a repeating group are aligned in columns. The number of N, V, or U operators necessary to print all the values in the file depends upon how many repetitions are currently defined for the group. For example, suppose a file is structured as follows:

ITEM	(unnamed repeating group)
SIZE	(of item - sq. ft.)
PART	(of item - named repeating group)

NAME	(of part - repetition name)
LENGTH	(of part - feet)
WIDTH	(of part - inches)

with the following data:

ITEM SIZE	PART NAME	LENGTH	WIDTH
1000 SQ. FT.	ABLE	10 FT	20 INCHES
	BAKER	30 FT	40 INCHES
2000 SQ. FT.	CHARLIE	50 FT	60 INCHES
	DOG	70 FT	80 INCHES
	EASY	90 FT	100 INCHES

The set of format operators...

...RAW,*P,V,S,P*,...

would produce as output the following values, in order:

1 1000 ABLE 10 20 BAKER 30 40 2 2000 CHARLIE 50 60 DOG 70 80 EASY 90 100

The values 1 and 2 are repetition numbers for the unnamed repeating group item. They are automatically generated for repetitions of any unnamed group.

The set of format operators

...ROW,*P,V,S,P*,...

would produce

1	1000	ABLE	10	20
		BAKER	30	40
2	2000	CHARLIE	50	60
		DOG	70	80
		EASY	90	100

If the format-type were COL, the output would be similar to that for ROW except that two extra columns would be placed between the values for size and the names ABLE, BAKER, CHARLIE, DOG, EASY, as described below.

The principle illustrated is defined more precisely by the description of the way the formatting program operates at the time it exhausts the properties within a single repetition of a group. When all properties in a group have been exhausted for one repetition, the group is stepped; i. e., processing recommences at the beginning of the next repetition (if there is one).

The action of the formatting program at the time a repeating group is stepped depends, in addition to format-type, on the presence or absence of line-changing operators (NXP, /, RCOL) in the format. The following rules govern the handling of groups.

(a) No Line Change

If no line changing operator is executed between the N operator - which prints the group name - and the V operator - which prints the repetition name, then, the following rules affect the indicated format-types.

	RULE	AFFECTED FORMAT-TYPE		
1.	For each repetition, the line coordinate is increased by one.	ROW	COL	
2.	For each repetition, formatting backs up to the N operator which printed the group name and proceeds from there.	ROW	COL	RAW
3.	The column in which the group name is printed is always the same.	ROW	COL	

	RULE	AFFECTED FORMAT-TYPE		
4.	When all repetitions have been formatted, the coordinates are reset to the line at which the first repetition was printed and to one plus the rightmost column used in formatting the first repetition.	ROW	COL	
5.	For the first repetition of any repeating group, the left margin is moved right two columns after the group name and repetition name have been printed and restored to its original setting when all repetitions of the group have been exhausted.		COL	

(b) Line Change

If any line changing operator is executed between the N operator - which prints the group name - and V operator which prints the repetition name - the action is the same except rule 2 becomes:

	RULE	AFFECTED FORMAT-TYPE		
2.'	For each repetition, formatting backs up to the V operator which printed the repetition name and proceeds from there.	ROW	COL	RAW

Repeating Group Stepping

Repeating group stepping occurs when the last property value in any repetition is formatted. For example, if a repeating group is being formatted by

MOF, ROW, *P, N, V

MOF, LIT, ZZZ

MOF, P*

the literal ZZZ will be output only once for the entire group.

Repeating group stepping may be postponed through the use of the *V operator, which prevents a repeating group from being stepped until a subsequent V* operator. Thus,

MOF, ROW, *P, N, *V

MOF, LIT, ZZZ

MOF, V*, P*

will format one field of ZZZ for each repetition. Applied to a nongroup property, the operator *V is identical to V and the operator V* has no effect.

MACRO AIDS

Begin and End

Some operators have special effects during the SMAC compilation of a format as opposed to at the time formatting of output being performed. In particular, BEGIN produces a card which contains

```
T                FORMAT, name
```

and END produces two extra cards in the output. These cards are needed as input to the format file updating program. In addition, BEGIN compiles a format operator to accomplish page numbering (if specified) and END compiles a format operator to signify the end of a format. Only one BEGIN and one END should be used with each format.

Labeling or Tagging

The operators B and BT specify branching; i.e., that the next format operator to be interpreted not be the next in sequence. To specify which operators should be next, a label or tag is used. Within a compilation, these tags may appear in the label field of any MOF instruction; for example

```
A          MOF,LIT...
```

or

```
HERE       MOF,N,S,N,S ...
```

If it is helpful to put a tag within a MOF statement, the tag may be specified with the T operator, as in

```
MOF,BT(ONE,ONE,TWO,ONE),T(ONE),/,T(TWO),N
```

which is equivalent to

```
           MOF,BT(ONE,ONE,TWO,ONE)
ONE        MOF,/
TWO        MOF,N
```

Tags may be any label acceptable to SMAC. The operator T(name) does not produce any format operator in the binary deck.

Strap or Smac Code Intermixed

Any STRAP or SMAC code can be intermixed with MOF statements; for example,

```
MOF,N,V
XW,1.32,1'       the STRAP code for S(1)
MOF,N,V
```

is equivalent to

```
MOF,N,V,S(1),N,V
```

If it is desirable to intermix STRAP or SMAC code with format operators in the same statement, the operator ST may be used; as in

MOF, N, V, ST(XW, 1.32, 1), N, V

which is also equivalent to

MOF, N, V, S(1), N, V

A particularly useful application occurs with literals, as in

MOF, F, S, ST(MOF, LIT, FILE), /(2)

which is equivalent to

MOF, F, S,
MOF, LIT, FILE
MOF, /(2)

and avoids the requirement that a LIT operation appear on a card by itself.

Since SMAC eliminates blanks in statements,

MOF, ST(MOF, LIT, IS A FILE)

will compile as

MOF, LIT, ISAFILE

The use of LIT inside ST does not preserve the internal blanks.

SOME EXAMPLES

Example of the format coding and output for each of the three format types- RAW, COL, and ROW, - are included in this section.

<u>Example of Column Format--SF1</u>

MOF, BEGIN(SFI, PP), COL'	THIS FORMAT IS TO BE NAMED 'SF1', PAGES ARE TO BE NUMBERED, ITS TYPE= COL

	MOF, *LPI, TIME'	OUTPUT USING THIS FORMAT IS TO BE SENSITIVE TO LIGHT-PEN. PRINT THE TIME
	MOF, / (2), QY, /(2), TL, /(2)'	SKIP A LINE, PRINT THE INPUT MESSAGE, SKIP A LINE, PRINT THE TITLE SUPPLIED, SKIP A LINE.
LABELA	MOF, *Q, Q, /'	FOR EACH OBJECT, PRINT THE OBJECT NAME AND GO TO THE NEXT LINE.
	MOF, *L(2), N(O), V(O), L*'	OMIT THE TWO STANDARD PROPERTY NAMES AND VALUES.
LABELB	MOF, *P, S(2), N, S'	FOR EACH PROPERTY, INDENT 2 SPACES (WITH RESPECT TO LEFT SIDE OF PAGE WHICH IS WHERE OBJECT NAME BEGINS), PRINT THE PROPERTY NAME IN A VARIABLE FIELD, AND SKIP 1 SPACE.
	MOF, *LM, V, LM*'	SET THE LEFT MARGIN SO THAT A PROPERTY VALUE TOO LONG FOR THIS LINE WILL OVERFLOW HERE (UNDER THE BEGINNING OF THE VALUE) RATHER THAN AT THE LEFT SIDE OF THE PAGE. PRINT THE VALUE.
	MOF, /, P*, /, Q*'	GO TO THE BEGINNING OF THE NEXT LINE AND GO BACK (TO THE FORMAT ELEMENT 'LABELB' ABOVE) FOR THE NEXT PROPERTY. WHEN ALL PROPERTIES ARE FINISHED, GO TO THE NEXT LINE AND GO BACK (TO THE FORMAT ELEMENT 'LABELA' ABOVE FOR THE NEXT OBJECT.

```
        MOF,IO,/,TIME,END'          WHEN ALL OBJECTS ARE
                                    DONE, OUTPUT WHAT IS
                                    DONE SO FAR. THEN GO TO
                                    THE NEXT LINE AND PRINT
                                    THE TIME. THE IO OPERATOR
                                    INSURES THAT THE TIME WILL
                                    BE OBTAINED AFTER OUTPUT
                                    IS BEGUN.

        MEND
```

The FABLE statement

FOR DOC . PRINT FORMAT SF1 ALL.

is entered and the following output is obtained.

```
13.24.18

  FOR DOC . PRINT FORMAT SF1 ALL .

12FF
  AUTHOR MILLS
  TITLE REVISED CATALOGUE OF ADAM DOCUMENTS TO DATE
  DATE  1
     DAY  15
     MO JUN
     YR  66
  KEYWORDS  1
     KEYWORD DOCUMENTATION
     K1 INDEX
     K2 -----
  KEYWORDS  2
     KEYWORD INDEX
     K1 -----
     K2 -----
  CAT -----
  STATUS -----

27M
  AUTHOR CLAPP
  TITLE BASAL
  DATE  1
```

```
    DAY   10
    MO   JUN
    YR   66
KEYWORDS   1
    KEYWORD ALLOCATION
    K1   BASAL
    K2   MARASS
KEYWORDS   2
    KEYWORD ALLOT TABLE
    K1 TOT TABLE
    K2 -----
KEYWORDS   3
    KEYWORD BASAL
    K1 -----
    K2 -----
KEYWORDS   4
    KEYWORD MARASS
    K1 -----
    K2 -----
KEYWORDS   5
    KEYWORD TOT TABLE
    K1 -----
    K2 -----
CAT -----
STATUS -----
```

Example of Raw Format

```
MOF, BEGIN(KEYS), RAW
MOF, *Q, /, Q(10), V(Ø), V(Ø)        PRINTS OBJECT NAME
MOF, *LM(Ø), V(27), LM*, Q*
MOF, END
```

A FABLE statement

FOR DOC. PRINT FORMAT KEYS KEYWORD.

is entered and the resultant output is illustrated in the table on the next page.

```
12FF      DOCUMENTATION          INDEX
27M       ALLOCATION             ALLOT TABLE       BASAL              MARASS             TOT TABLE
29A       THRUPUT                ADAM GENERAL      DESIGN
36B       ROUTINES               ALLOCATION        LINE               CLOD               PAT
          CHECKOUT               FAVOR             CORE               TABLES
42C       FILE POINTER SET       OBJECT ROLL       DATA STRUCTURES    REPEATING GROUPS   FILES
          PROPERTY ROLL          D-SPEC 16
43B       DATA STRUCTURES        ALLOCATION        STREAMS            STRCON             SHP
45B       THRUPUT                OUTPUT            TOP
47J       THRUPUT                PROCESSOR         SINTAB             TRANSLATOR
60D       LOADING                CLOD              CHECKOUT           ASSEMBLY
62A       COMPILER               PRE-PROCESSOR     DAMSEL
63A       DATA                   MSAVE             26-RELATIVE        BASAL              MLX
          MUNSAVE                26-ABSOLUTE       GLOBALS            ROUTINES
64A       MSIM                   ROUTINES          MRIM               RETURN$A           CALLING SEQUEN
                                                                                         CES
          CONVENTIONS
65A       FAVOR                  ROUTINES          CLOD               AS$A               PAT
          AUTOSTACKED            CONVENTIONS       RETURN             TABLE OF EXITS     SIO$A
76H       FILE GENERATION
78J       ROLLS                  ROLL ROUTINE
13.25.03  NORMAL RETURN FROM   THE PROCESSOR.

     FOR DOC . PRINT FORMAT KEYS KEYWORD .

13.25.03  MESSAGE PROCESSING COMPLETED.

13.25.03       JOB ENDED BY EOJ.
```

Example of Row Format

```
MOF, BEGIN(DL), ROW              ROW TYPE FORMAT, NAME IS DL
MOF, *H                          A HEADING TO BE TYPED ON
MOF, LIT(10), DOC. NO            EACH PAGE
MOF, LIT(9), DATE                      .
MOF, S(5)                              .
MOF, LIT(20), AUTHOR                   .
MOF, LIT(5), TITLE                     .
MOF, H*, /                             .
MOF, *Q, V(Ø), V(Ø), Q(10)       PRINT OBJECT
MOF, V(Ø), V(3), S(1), V(3), V(3), S(5)
MOF, V(20), V, /, Q*
MOF, END
```

A FABLE message which uses this format might be:

FOR DOC. PRINT FORMAT DL OBJECT NAME, DATE, AUTHOR, TITLE.

The subsequent printing is shown in the table on the next page.

DOC NO.	DATE	AUTHOR	TITLE
12FF	15 JUN 66	MILLS	REVISED CATALOGUE OF ADAM DOCUMENTS TO DATE
27M	10 JUN 66	CLAPP	BASAL
29A	2 NOV 64	KILBANE	ON-LINE OUTPUT FORMATTING IN ADAM
36B	14 FEB 64	HELWIG	GROSS ORGANIZATION OF CORE MEMORY
42C	26 MAR 65	FILE GEN. GP.	FILES AND FILE FORMATS
43B	14 FEB 66	BENSLEY, CLAPP	STREAMS AND STREAM CONTROL (STRCON)
45B	8 JAN 65	KILBANE	INPUT DATA FORMATS FOR PROGRAM TOP
47J	20 MAY 66	TERRASI	A DISCRIPTION OF THE PROCESSOR AND SINTAB
60D	2 JUL 64	PEW + HELWIG	ADAM RECURSIVE LOADER CLOD
62A	20 MAY 64	CLAPP + HODGINS	THE DAMSEL PRIMER
63A	18 MAR 65	CONNORS	HOW ADAM PROGRAMS REFER TO DATA
64A	18 JUL 63	CONNORS	CONVENTIONS FOR WRITING ROUTINES IN ADAM
65A	18 JUL 63	CONNORS	USE OF ONE ADAM ROUTINE BY ANOTHER
76H	7 MAR 66	MILLER, ANDERSON	INITIAL FILE DESCRIPTION LANGUAGE
78J	5 JAN 66	FILE GEN GP.	USERS MAN. FOR ROLL HANDLING ROUTINES

SECTION VII

WRITING ROUTINES

An ADAM routine is the data structure used for defining procedures which can be executed by the system. All ADAM routines, both system and user defined, are kept in an ADAM file called the Routine File. In this way, the ADAM file handling capabilities may be used to modify and add routines, and to retrieve routines for execution. The name of a routine and its entry option names, if any, are in rolls associated with the Routine File. Descriptions of the input and output parameters of a routine may also be stored in a system roll. This information is available for the translation and checking of statements which reference routines in on-line languages, such as FABLE, and in the DAMSEL language. The Routine File also contains tables, usually consisting of data, which are not executed by the system.

ADAM ROUTINES

Executable routines, hereafter called routines, must be coded as subroutines. They may be written in the SMAC, DAMSEL, or FORTRAN languages and are compiled into standard binary decks, q.v., which consist of two relocatable binary subdecks; the roll data subdeck and the routine file subdeck. A standard binary deck may be entered into the Routine File by the routine RUE (see Routine File Updating). An integer called a principal value or PV, is associated with the routine at this time.

A routine in the Routine File may be loaded into core memory by the routine CLOD (see Routine Loading). Two kinds of loading are available: fixed routine loading and allocatable routine loading.

A routine in core memory may be executed (using system linkage and resource allocation conventions described in System Conventions for Routines) by branching to its first half-word instruction. A routine can perform multiple functions (called entry options) which are specified by integers called entry option numbers. An entry option may in turn have an associated parameter configuration which describes its input and output parameters. A routine may refer to data contained in other ADAM data structures by using system conventions; in particular, data that belongs to another routine may be referenced using a global symbol.

STANDARD BINARY DECKS

The roll data subdeck defines the various attributes of the routine. The routine file subdeck contains the instructions and constants which actually form the routine, and it always begins with a standard routine heading.

Roll Data Subdeck

The following routine attributes are defined by the roll data subdeck:

(a) The PV of the routine in the Routine Roll. This is an optional attribute, since a PV can be supplied at Routine File update time.

(b) The loading attribute: fixed or allocatable.

(c) The execution attribute: routine or table.

(d) An optional list of parameter descriptions in which each element contains the following data:

  (1) The type attribute: input or output parameter.

  (2) The relative location of the parameter in the input or output area associated with the routine at execution time.

  (3) The structure of the parameter (floating point number, integer, index word, etc.).

(e) An optional list of entry option descriptions in which each element contains the following data:

   (1) The name and the number of the entry option.

   (2) The parameter configuration (an ordered subset of parameter descriptions) associated with the entry option.

(f) An optional list of global symbol descriptions in which each element contains the name and value of a global symbol.

Routine File Subdeck

The routine file subdeck contains relocation information and the code in relocatable binary form.

ROUTINE FILE

The Routine File is an ADAM file which is maintained by the routine RUE.

The object roll of the Routine File is called the Routine Roll and contains a component called the Routine Pointer Set which is permanently in core memory. Entries in the Routine Pointer Set are used by CLOD in loading routines.

A second roll, called the Compiler Roll, is required to completely describe routines which contain entry options or global symbols. The Compiler Roll is used by the DAMSEL compiler and the ADAM Translator to evaluate routine calls which are expressed in mathematical function notation and to evaluate system symbol references.

Both the Routine Roll and the Compiler Roll are updated by RUE.

ROUTINE FILE UPDATING (RUE)

Routines may be deleted from the Routine File, added to the Routine File, or corrected by a routine called RUE.

Delete Options

An existing routine may be deleted from the Routine File and the Compiler Roll. The routine may be identified either by its name or by its PV in the Routine Roll.

Add Option

A new routine may be added to the Routine File and the Compiler Roll. The name of the routine is specified at this time and a standard binary deck must be provided. If the binary deck does not specify a PV in the Routine Roll, then, RUE will supply one. The deck will be rejected if its inclusion would cause ambiguity in either the Routine Roll or the Compiler Roll.

Correct Option

The code section of an existing routine may be modified. The name of the routine or its PV in the Routine Roll must be specified and a correction deck must be provided. The corrections may be in a relocatable octal or relocatable binary form.

ROUTINE LOADING (CLOD)

A routine in the Routine File may be loaded into core memory by a routine called CLOD. Two kinds of loading are possible: fixed routine loading and allocatable routine loading. In both cases the input to CLOD is the PV of the routine in the Routine Roll and the output from CLOD is a location in a table called the Program Allocation Table (PAT). The routine may or may not actually be in core when CLOD returns control, but in any case it can be executed by branching to its PAT location.

Fixed Routine Loading

Any routine may be loaded into a section of core called the fixed routine area and be described by an entry in the fixed PAT Table. Routines loaded into the fixed routine area must have fixed PAT locations associated with them at compile time. The size of the fixed routine area is established when the ADAM system is generated. In general, new routines may not be loaded into the fixed routine area but an existing routine may be overloaded by a smaller routine.

Certain routines in the ADAM system have been assigned permanent locations in the fixed PAT Table. Any number of routines may have the same fixed PAT location but only one such routine can be in the fixed routine area at any given moment.

Allocatable Routine Loading

Any routine may be loaded into a section of core called the allocatable routine area and may be described by an entry in the Allocatable PAT Table. The size of the allocatable routine area can vary at system execution time. An allocatable routine can be dismissed and subsequently deleted from core; space relinquished by deleted routines becomes available for loading other routines and data.

If a routine has been assigned an allocatable PAT location it retains this location (regardless of its core status) until a general release is given. An allocatable routine can always be executed by branching to its PAT location. If it has not been loaded it will be allocated and loaded automatically.

Allocatable Routine Dismissal

CLOD has an option which will dismiss a routine (whose PV in the Routine Roll is specified) by turning on the dismiss bit in the allocatable

PAT entry. Another bit in the PAT entry (called the in-use bit) can be used to override the dismiss bit. Routines are assigned locations from low to high numbered storage areas; when CLOD loads a routine that is not in core it will first delete from core all routines that satisfy the following conditions:

(a) The in-use bit is off and the dismiss bit is on.

(b) All routines that have higher core locations satisfy condition (a).

General Release

General release is a CLOD option which can only be executed by routines in the fixed routine area. This option deletes all allocatable routines from core and deletes all entries in the allocatable PAT table.

SYSTEM CONVENTIONS FOR ADAM ROUTINES

The following section describes system conventions which must be followed by routines. System macros have been defined that implement these conventions.

Conventions for Called Routine

Routines must be coded as subroutines, entered at the first location, and are expected to respond to a calling sequence which generates the following standard environment:

(a) A specific location (called RETURN) in a specific table (called FAVOR) contains the location of a table-of-exits in the calling routine.

(b) If a routine requires input data, index register 13 normally contains a data pointer (see Movable Data) which specifies the location of an input block containing the data.

(c) If a routine produces output data, index register 14 normally contains a data pointer which specifies the location of an output

block into which data will be stored. The structure of the input and output blocks is defined by the called routine. The location of the blocks is specified by the calling routine.

(d) If a routine has entry options, the value field of index register 15 contains the entry option number specified by the calling routine.

Routines must preserve (i. e., save and restore) the contents of the following machine resources:

The indicator register ($ IND)

The mask register ($MASK)

Location RETURN in the FAVOR table.

The system macro MSIM can be used to save these registers and to set $MASK to a specified value.

MSIM, BLOCK, mask-location

. . . .

BLOCK DRZ (U) , 3

The system macro MRIM can be used to restore them.

MRIM, BLOCK

Routines must also preserve the contents of any index registers which they use except for index registers 13, 14, and 15. The system macros MSAVE and MUNSAVE can be used for this purpose (see Movable Data). Routines that call other routines must also preserve index registers 13, 14, and 15 since they are used in the standard routine calling sequence.

To prevent being overwritten, a routine should turn on its in-use bit when it is entered and turn it off when it exits. A routine may turn on or turn off the in-use bit by using the system macro MROUT as shown below:

MROUT, UB, ON

MROUT, UB, OFF

When a routine has finished operating, it must return control to a table-of-exits whose location is specified by RETURN. The first half-word in the table-of-exits is always the error return. The meaning of other half-words in the table of exits can be defined by the routine. Before exiting, a routine should be sure that it has released all data allocations that it made.

Conventions for Calling a Routine

The following example illustrates a call to a routine which defines one error exit and one normal exit. The system macro MPOINT is used to specify the locations of the input block (INLOC) and the output block (OUTLOC). The system symbol RETURN$A is used to refer to the location RETURN in the FAVOR table. The routine is presumed to be loaded, or at least assigned a PAT location, and is entered by branching to its PAT location.

```
MPOINT, INLOC, $13
MPOINT, OUTLOC, $14
LVI, 15, .32
SIC, RETURN$A
B, pat-table-location
B, error-return
```

Routines can be loaded into core by using the system macros MAD and MADATA. A reference to the macro MAD generates a call to CLOD and has the following format:

```
MAD, CLOD, entry-option-name
         , input-block-location
         , output-block-location
         , error-exit
```

The following example (entry-option-name = PA, meaning PAT Assign only) obtains a location in the allocatable PAT table for a routine (whose PV in the routine roll is 1Ø) and stores this location in OUT:

```
         MAD, CLOD, PA, IN, OUT, ERR
         .....
IN       MADATA, CLOD, PA, 1Ø.
OUT      BE, Ø
ERR      error return
```

The routine may or may not be in core memory when CLOD returns control but it can be executed by branching to OUT.

The following example (entry-option-name = PALI, meaning PAT Assign and Load Immediately) is equivalent to the above example but guarantees that the routine is in core memory when CLOD returns control.

```
         MAD, CLOD, PALI, IN, OUT, ERR
```

The PAT table location specified by OUT contains the core memory location of the routine.

Specifying Roll Data for Standard Binary Decks

Standard binary decks are automatically punched when a routine written in either DAMSEL or FORTRAN language is compiled. If a routine is written in SMAC language, system macros must be used to produce the roll data sub-deck which describes entry names, global names, and parameters for a routine.

In the following example, the system macro MASSEMA is used to define an allocatable routine*(called SINCOS) which has two entry options (called SIN and COS) and defines a global symbol (called PI).

---

*The system macro MASSEMF is used with fixed routines.

```
MASSEMA, SINCOS          routine name
         , PV(1Ø)        routine roll PV(optional)
         , E(SIN, COS)   entry-option names
         , G(PI)         global symbol names
ROUTINE CODE             The symbols SINCOS and PI must appear
         '               in location fields here.  SINCOS is the
         '               start of the routine.
         '
MROUT, EC                Produces a branch card.
MEND
```

The entry option numbers associated with SIN and COS are Ø and 1 respectively.

Parameters and parameter configurations may be specified using MASSEMA. In the following example a routine (ROUT) has two entry options (ENT1 and ENT2) and four parameters. These are two input parameters which are floating point numbers (I. RE) and two output parameters which are integers (O. I). The first input parameter and the first output parameter are used by entry option ENT1.

The second input parameter and the second output parameter are used by entry option ENT2.

```
MASSEMA, ROUT
         , PV(1Ø)
         , P(I. RE,      parameter 1 (input, real)
              O. RE,     parameter 2 (output, real)
              I. I,      parameter 3 (input, integer)
              O. I)      parameter 4 (output, integer)
```

```
            ,E(ENT1(1,2),
               ENT2(3,4))
    . . . . . . . . . . . . . . . . . .
    MROUT,EC
    MEND
```

References to Data

Data may be contained in routines or in data structures. Data belonging to ADAM data structures is normally referenced by bringing it into ADAM areas which are movable blocks of core storage. Routine data and data in certain ADAM data structures can be referenced using system symbols.

Routine Data

Any reference to data contained in another routine must be made in terms of a global symbol whose value is normally a relative location in the routine being referenced. Relative-to-absolute conversion must be done by the routine containing the reference and can be accomplished either at load time or execute time.

Global symbols may be defined at compile time in either of the following ways:

(a) By including a reference to the system macro MADAM if the program is being compiled by SMAC.

(b) By the Compiler Roll if the program is being compiled by DAMSEL.

Global symbol references are system tailed symbols and can use the system tails G, A, and I. Symbols tailed by I may not be defined using MADAM.

The value of the system symbol

name $G

is a relative location in the routine which defines name. It can be converted

to an absolute location if the core origin of the routine is known. For example, if a routine whose PV is 1Ø defines a global called GLOB, the following program will place the absolute location of GLOB into index register 7:

MAD, CLOD, PALI, IN, OUT, ERR	load the routine
LV, $7, OUT	PAT table location
LV, $7, ($7)	core location of routine
V+I, $7, GLOB$G	

The system tail A is reserved for references to global symbols defined by the fixed PAT table and the FAVOR table. Certain fixed routines (e.g., CLOD, and BASAL) have been assigned permanent fixed PAT locations. The value of the system symbol

name$A

is the absolute location in the fixed PAT table associated with the routine name.

For example, the absolute location of the global HIST, which is defined by the routine CLOD, can be computed as follows:

LV, $7, CLOD$A
V+I, $7, HIST$G

The system tail I may not be evaluated using MADAM. The value of the system symbol

name$I

is

name $G plus the core origin of the routine which defines name. I tailed symbols are evaluated by CLOD which will automatically load the routine which defines the global. The previous example involving the global GLOB may also be written as

LVI, $7, GLOB$I

If a routine that defines global symbols is re-compiled and the values of the global symbols change, then all routines that refer to the globals must also be re-compiled.

System Data

ADAM routines may use system tailed symbols to refer to names defined by certain rolls in the ADAM system. The system tails involved are F, R, P, and E. The system symbols may be defined using MADAM (SMAC compilations) or the rolls of the ADAM system (DAMSEL compilations). MADAM does not contain a complete set of these system tailed symbols, however. The following table indicates the name, the System Tailed Symbol, and the System Symbol value.

NAME	SYSTEM TAILED SYMBOL	SYSTEM SYMBOL VALUE
File name	name$F	PV of file in $FILE
Roll name	name$R	PV of roll in $ROLL
ADAM Routine name	name $P	PV of routine in Routine Roll
Entry Option name	name$E	Entry Number of name

In the following example a routine called SINCOS is loaded and an entry option called SIN is executed.

```
        MAD,CLOD, PALI, IN, OUT, ERR
        MPOINT, INLOC, $13
        MPOINT, OUTLOC, $14
        LVI, 15, SIN$E
        SIC, RETURN$A
OUT     BE, Ø
        B, ERR
```

.....

IN	MADATA, CLOD, PALI, SINCOS$P
ERR	error return

Movable Data

ADAM data structures are referenced by bringing them (perhaps a part at a time) into ADAM areas. An area is a core block which is created, managed, and deleted by a routine called BASAL. Whenever BASAL is used to create an area, it provides the user with a half-word instruction (called an AID) having the form:

LVE,, allot-location

where allot-location is an absolute location in an ADAM table (called ALLOT) which describes the area just created. Areas can be moved in core at system execution time. Data in an area is referenced by a data pointer which has index word format. The value field of data pointer specifies a 24-bit location. There are two kinds of data pointers: fixed data pointers and movable data pointers.

Bit 26 of a fixed data pointer is always Ø. The location specified by the value field is an absolute core location. The count and refill fields are not used.

Bit 26 of a movable data pointer is always 1. The refill field contains the location of an entry in the ALLOT table which describes an ADAM area. The location specified by the value field is either an absolute location in the area (index register form) or a relative location in the area (core memory form). The count field is not used.

A movable pointer should normally be in core memory form when it resides in a core register and in index register form when it resides in an index

register. Any call on BASAL can cause data to move; when this occurs, BASAL will automatically update all movable pointers which reside in index registers and will assume that they are in index register form.

The macro MSAVE can be used to convert a data pointer in an index register from index register form to core memory form. For example:

```
        MSAVE, LOC, $7
        .....
LOC DRZ(U), 1
```

stores the transformed pointer in core memory location LOC.

The macro MUNSAVE can be used to convert a data pointer which resides in core memory from core memory form to index register form. For example:

```
MUNSAVE, LOC, $7
```

stores the transformed pointer in index register 7.

The use of data pointers to specify input and output block locations in routine calling sequences is discussed in CONVENTIONS FOR THE CALLED ROUTINE.

The macro MLX can be used to convert an AID which resides in core memory into a movable data pointer having index register form. The resulting pointer references relative location Ø in the area. For example:

```
        MLX, $7, LOC
        .....
LOC VF, Ø
```

stores the movable pointer in index register 7.

Considerations for Code Operated in Autostacked Mode

In ADAM, a portion of code may be operated in autostacked mode either by accepting an interrupt or by being called by another autostacked routine. In either case, certain special considerations that should be noted are discussed below.

If a program accepts an interrupt and subsequently executes an MRET macro without ever giving up control, there are, in general, no additional requirements levied on the code. Use of the MSAVE and MUNSAVE macros should be avoided since the index registers may contain invalid data (alternatively, the index registers could be cleared upon accepting the interrupt). If, however, the program accepts an interrupt and then calls another ADAM routine, certain rules must be followed:

(a) AS$A must be set to 1 before calling any such routine and reset to Ø just before the MRET

(b) RETURN$A must be saved before calling any such routine and restored just before the MRET.

(c) The routine being called must allow autostacked calls.

(d) It is the best policy to clear all index registers upon accepting the interrupt in case the called routine is to use MSAVE and MUNSAVE.

A routine that is called by an autostacked routine has slightly different problems. It should avoid the use of MSAVE and MUSAVE in case the original accepter of the interrupt did not clear the index registers. A routine that may be called either autostacked or mainstream should take precautions to avoid being unexpectedly re-entered. The easiest way to do this is to use an MSIO macro at the beginning of the routine. The indicator SIO$A may then be tested for Ø to see if the routine should execute a MRIO macro before exiting; SIO$A

should then be set to 1 to indicate the true state of the machine. If an MRIO is subsequently executed, SIO$A should then be set to Ø. It is also possible to code the routine in a re-entrant manner, it is not to be recommended unless efficiency considerations demand it.

An allocatable routine must provide that any interrupt designated for an IO table-of-exits, internal to itself, be accepted prior to the final exit of the routine. Otherwise, a different routine may be resident at that location when the interrupt occurs.

Autostacked time signal interrupts provide a problem all of their own. It is not possible for the restart procedure to throw them away without additional information. A general method for solving this problem is not available in ADAM; therefore, there is an element of risk in using autostacked time signals without making corresponding changes in the restart program (BS1). This risk is so minute in the case of very short intervals that this is still the recommended method of getting into autostacked mode when necessary. Larger intervals, however, entail more risk and should be avoided if possible.

SYSTEM CONVENTIONS FOR FORTRAN ROUTINES

The COMFORT (COMpatible FORTran) system is a method for running specially prepared programs coded in FORTRAN within the ADAM System.

Preparation

The preparation of COMFORT routines involve the following steps, in order:

(a) A routine is written in FORTRAN, with certain special subroutine calls included for compatibility with ADAM.

(b) The routine is compiled with the FORTRAN compiler to produce a binary deck.

(c) The binary deck is processed with the COMFORT post-processor program to produce a revised binary deck suitable for inclusion in the ADAM Routinc File.

(d) The revised binary deck is inserted into the ADAM Routine File through the ADAM Routine-File update program.

(e) One or more "COMFORT lists" are prepared and inserted into the ADAM Routine File. A COMFORT list is a table of names of FORTRAN routines, DAMSEL routines, and COMMONS (if any) to be loaded into core and operated together.

Execution

At operate time, all routines on a single COMFORT list are loaded together and operated under control of the COMFORT Monitor (an ADAM routine), as follows:

(a) An ADAM routine calls the COMFORT Monitor and specifies to it the name or location of a COMFORT list.

(b) The COMFORT Monitor loads all routines on the COMFORT list into core and branches to the first routine on the list.

(c) Whenever a FORTRAN-coded routine must perform some action in order to be compatible with ADAM, it calls an appropriate subroutine.

(d) When the operation of routines on the COMFORT list is finished, the first routine returns to the COMFORT Monitor, which dismisses all routines on the COMFORT list and returns to the routine that called it.

Comfort List

A COMFORT list is an ADAM table, separately prepared and inserted into the Routine File. It contains the names of a group of DAMSEL routines and tables, or FORTRAN routines or COMMONS (if any).

All the routines of a COMFORT list are loaded together within the COMFORT list; FORTRAN routines may call only other routines in the list. DAMSEL routines may call on routines either within or without the list. The first routine on a list is the routine to which the COMFORT Monitor gives control and, in that sense, is similar to a FORTRAN main program. Routines within a COMFORT list pass data to one another either through arguments of calls or through COMMON. Initially, no provision exists for FORTRAN routines to accomplish input or output, or to access ADAM files or rolls; they get their data from, deliver their results to, and perform I/O through DAMSEL routines on the same list. DAMSEL routines within a list may access any ADAM data and perform I/O.

Restrictions on Fortran Statements

FORTRAN routines for use with COMFORT may contain any statements allowed in FORTRAN with the following exceptions:

(a) Every subprogram must be either FUNCTION or SUBROUTINE; main programs are not allowed.

(b) I/O statements including READ, PUNCH, PRINT, WRITE, END FILE, BACKSPACE, and REWIND are not allowed.

(c) Programs may not be segmented; the NODE card is not allowed.

(d) A FORTRAN/COMFORT routine may refer, by CALL statements or function references, only to routines on the same COMFORT list; i.e., to routines which have been loaded with it.

(e) A FORTRAN/COMFORT routine may not have a name identical to the name of a COMFORT subroutine.

(f) COMMON blocks may not be labeled.

Required Heading

Every FORTRAN routine to be used within ADAM must begin with calls to three dummy subroutines named ZHEDØ, ZHED1, ZHED2; e.g.,

```
SUBROUTINE...
CALL     ZHEDØ
CALL     ZHED1
CALL     ZHED2
```

These calls are never executed; they only serve to reserve space in the transfer vector for ADAM compatibility information.

Fortran Calls to Damsel Routines

A FORTRAN routine may call DAMSEL routines only if they are on the loaded COMFORT list, and must call them in a special way in order to make the parameters of a call compatible with ADAM usage.

First, the calling routine must execute a CALL ZSAVE statement immediately after the CALL ZHED2 statement and a CALL ZUNSAV statement immediately before each RETURN statement; e.g.,

```
SUBROUTINE...
CALL     ZHEDØ
CALL     ZHED1
CALL     ZHED2
CALL     ZSAVE
    .
    .
    .
CALL     ZUNSAV
RETURN
```

```
        .
        .
        .
CALL    ZUNSAV
RETURN
        .
        .
        .
```

These statements call routines which preserve index registers in a manner compatible with ADAM usage.

Second, the calling routine must execute a CALL ZPLIST statement immediately before the call to the DAMSEL routine and a CALL ZFIXUP statement immediately after the call to the DAMSEL routine. These statements call routines which adjust the arguments of the call to be compatible with DAMSEL usage.

NB: A DAMSEL routine requires that its input parameters be located together and that its output parameters be located together. It accepts an entry option, e.g., the SINCOS routine may have a SYN entry = 1 and a COS entry = 2, and returns with either a "normal" return or an "error" return. ZPLIST arguments need not obey these restrictions.

The form of ZPLIST and ZFIXUP are:

$$\begin{array}{l}\vdots\\ \text{CALL ZPLIST (I, N, Input}_1\text{, Units}_1\text{, Input}_2\text{, Units}_2\text{, ..., M,}\\ \qquad\qquad\text{Output}_1\text{, Output}_2\text{...)}\\ \text{CALL routine name}\\ \text{CALL ZFIXUP(J)}\\ \vdots\end{array}$$

In which

I	- is the entry option number for the DAMSEL routine.
N	- is the number of input pairs following.
$Input_1$, $Units_1$	- are pairs of parameters, each of which consists of an input parameter value and the name (in A8) of the units (*) in which it is measured; e.g....6,Ø...
M	- is the number of output pairs following.
$Output_1$, $Units_1$ $Output_2$, $Units_2$	- are pairs of output parameters in the format of input pairs above.
J	- is a parameter set to Ø if the called routine returns normally; set to nonzero if the called routine intends an error return.

Damsel Calls to Fortran Routines

A DAMSEL routine may call a FORTRAN routine only if the FORTRAN routine is within a COMFORT list already in core and the FORTRAN routine has been specially prepared to accept DAMSEL calls. The preparation requires the use of CALL ZENTER and CALL ZEXIT statements. A routine so prepared may be called only with the DAMSEL linkage; i.e., either by a DAMSEL routine or by a FORTRAN call including ZPLIST and ZFIXUP statements.

The FORTRAN routine to be called via a DAMSEL linkage must contain a CALL ZENTER statement immediately after the CALL ZHED2 statement (before a CALL ZSAVE, if one is used) and a CALL ZEXIT statement immediately before any RETURN statements (after CALL ZUNSAVE, if one is used). These

*$Units_1$ and $Units_2$ must be Ø (i.e., no units).

statements call routines which adjust the formats of DAMSEL calls to be compatible with FORTRAN usage. Their form is:

```
SUBROUTINE
CALL ZHEDØ
CALL ZHED1
CALL ZHED2
CALL ZENTER (I)
(CALL ZSAVE, if used)
        .
        .
        .
(CALL ZUNSAV, if used)
CALL ZEXIT (J)
RETURN
        .
        .
        .
(CALL ZUNSAV, if used)
CALL ZEXIT(J)
RETURN
        .
        .
        .
```

Note that, initially at least, a DAMSEL routine that calls a FORTRAN routine cannot include input or output parameters in its call. The calling DAMSEL routine may store parameters into and retrieve parameters from COMMON, however.

In ZENTER, the single integer argument (I in the example) is set equal to the entry option number supplied by the calling routine. CALL ZENTER(I) must be used, even if the entry number is not desired.

In ZEXIT, the single integer argument (J in the example) is to be set by the called routine. When Ø, it indicates a "normal" return. Nonzero indicates an "error" return.

Library Routines

FORTRAN library routines (except open functions) may be used only if they have been processed by the COMFORT postprocessor and inserted into the ADAM Routine File. In order to be processed by the postprocessor, FORTRAN library routines must have the compatibility statements (CALL-ZHEDØ, etc.) or STRAP equivalents inserted into their symbolic decks before FORTRAN compilation and COMFORT postprocessing.

Open functions may be used without special preparations.

APPENDIX I

SYNTAX OF FABLE

GUIDE TO FABLE SYNTAX

Basic constituents are represented as follows: lower case letters indicate a subdiagrem; upper case letters and , . ( ) + = - * / stand for themselves; { } indicate that one of the items within the braces is required; [ ] indicete that one of the items within the brackets is optional; ...[ ] indicete eny number of the items within the breckets is permissible.

AN is an elphanumeric string which may contain blanks.
CN is a property value to be converted by a convert-in routine.
FN is a file name.
NV is a string of digits.
ON is an object name.
PN is a property name.
RGN is a name of a repeating group type of property.
RN is a name in a roll.
RP is a repetition name.
RT is a routine or entry-point name.
RV is any string of characters enclosed in 's and not containing the character '.

Message: [statement]...[statement]

TYPES OF STATEMENTS

statements:

```
             ⎧⎧TYPE        ⎫                                     ⎫
             ⎪⎪            ⎪    (1)                 (2)          ⎪
for[boolcl,]⎨⎨DISPLAY[NEW]⎬[RN···[RN]][FORMAT ON][TITLE RV]⎬{proplist}.
             ⎪⎪PRINT       ⎪                                     ⎪{ALL     }
             ⎪⎩OUTPUT[NEW] ⎭                                     ⎪
             ⎩ SAVE                                              ⎭
                              → [NAME AN.][SORT[sortrg]ON sortpl.]

for boolcl.

                                        ⎧⎧TYPE        ⎫                                  ⎫
                                        ⎪⎪DISPLAY[NEW]⎪    (1)                            ⎪
for TALLY FOR tl[AND tl].[boolcl,]⎨⎨PRINT       ⎬[RN···[RN]][FORMAT TALLY][TITLE RV]⎬.
                                        ⎪⎩OUTPUT[NEW] ⎭                                  ⎪
                                        ⎩ SAVE                                           ⎭
                                          → TALLY[ae].[NAME AN.]

DO routcall.

SORT FN[sortol sortrg]ON sortpl.[NAME AN.]
       [       sortrg]

                                 ⎧PROPERTY NAMES OF FN         ⎫
⎧TYPE        ⎫     (1)           ⎪OBJECT NAMES OF FN           ⎪
⎨DISPLAY[NEW]⎬[RN···[RN]]        ⎨ELEMENT NAMES OF RN(3)       ⎬.
⎪PRINT       ⎪                   ⎪PROPERTY ROLL CONTENTS OF FN ⎪
⎩OUTPUT[NEW] ⎭                   ⎩LOGICAL VALUES OF FN cpn     ⎭

{DELETE}FN[FILE].
{REMOVE}

                                       ⎧PROPERTY FN cpn            ⎫
⎧DELETE⎫ {SYNONYM }                    ⎪OBJECT NAME FN ON          ⎪
⎨REMOVE⎬ {SYNONYMS}AN···[,AN]FOR       ⎨LOGICAL VALUE AN OF FN cpn ⎬ .
⎩ADD   ⎭                               ⎩LOGICAL VALUE AN IN RN(3)  ⎭
```

---
(1) Device Roll
(2) Format File
(3) Rolls Roll

RENAME { FILE FN | PROPERTY FN cpn | OBJECT [NAME] FN ON | LOGICAL VALUE AN {IN | OF} FN cpn } {AS | TO} AN .

for ADD {VALUE | VALUES} AN ...[,AN] TO cpn .

PARTS OF STATEMENTS

sortol: ON···[,ON]

sortrg: RGN···[RGN]

sortpl: PN [ASCEN | ASCENDING | DESCN | DESCENDING] [,sortpl]

for: FOR FN [ON(1)···[,ON]] [ALTER {FN(scpn) | FN ON | FN}] .

proplist: {nonrg | rg}...[, {nonrg | rg}]

nonrg: {PN | RV [OF RGN] = {ae | scpn}}

rg: rgspec[proplist]

rgspec: RGN [RP ···[,RP]]

srgspec: rgspec...[rgspec]

---

(1) In preceding FN

tl: { ae, { EQ | LS | GR | GQ | LQ } { aeaeae (1)(2)(3) | ae...[,ae] (4) } | scpn (7) }

change: cpn TO { NULL | ae | scpn | RN (5) }

cpn: [srgspec] PN

scpn: { [FN | FN ON (6) | ON | FN (scpn)] cpn | routcall }

vl: value···[,value]

reps: [srgspec]RGN(vl···[,,vl])

value: { PN = { ae | scpn | AN } | reps }

---

(1) $Limit_1$

(2) An increment

(3) $Limit_2$

(4) Maximum of 25 range specifications.

(5) In roll of cpn

(6) In preceding FN

(7) Logical property only.

```
altfz:      { CHANGE change···[,change]
            {       {REPETITION }
            {       {REPETITIONS} reps
            { ADD
            {       {OBJECT }
            {       {OBJECTS} vl...[,,vl]
            {
            {           { {REPETITION }                       }
            {           { {REPETITIONS}                       }
            { {DELETE}  {                       {GROUP }      } srgspec...[,srgspec]
            { {REMOVE}  { [REPEATING]           {GROUPS}      }
            {           {
            {           { {OBJECT }
            {           { {OBJECTS}
            { DO routcall

boolcl:     bool [UNTIL NV]

bool:       booltrm···[OR booltrm]

                                            { {ANY}                                  }
booltrm:    [IF][FOR  forcl···[,forcl].]    { {ALL}  boolprm[altfz...[,altfz]]       }
                                            { altfz...[,altfz]                       }
                 ···[AND booltrm]

            { {FN     }             }
forcl:(1)   { {ON     } [srgspec]   }
            { {FN ON  }             }
            { {FN (scpn)}           }
            {  srgspec              }

                   {NULL(scpn(3))
                   {ELSE
boolprm:    [NOT]  {(bool)
                   {blop relation (2)brop [{AND} relation(2)brop]
                                           [{OR }                ]

blop:       blot [,blop]            brop:     brot [,brop]

blot:       blpp [ALSO blot]        brot:     brpp [ALSO brot]

            {(blop)}                          {(brop)}
blpp:       {scpn  }                brpp:     {ae    }
            {ae    }                          {scpn  }
                                              {RN    }
```

(1) Restrictions similar to those for scpn apply to forcl.

(2) Operands on each side of the relation must match; i.e., an ae must have an ae and an scpn must have an RN in the same roll or an scpn with the same roll.

(3) May not be raw type property.

relation: [NOT] {EQUAL | EQUALS | EQ | GREATER | GR | LESS | LS | GQ | LQ}

ae: [- | +] term...[{- | +}term]

term: {factor [*term] | factor / factor}

factor: {(ae) | CN | number | routcall | scpn(1)}

number: [+ | -] NV [.NV] [([+ | -]NV)]

routcall: RT({ae(2) | scpn | RV | RN}...[, {ae(2) | scpn | RV | RN}])

(1) Must be an arithmetic property.

(2) ae or scpn may not specify raw valued properties.

APPENDIX II

TWO SAMPLE FILES USED IN THE EXAMPLES

1. AIRFIELD File

OBJECT NAME	
ALT	Logical, uses object roll of AIRFIELD file
LAT	Numeric
LONG	Numeric
CITY	Uses object roll of CITY file
RUNWAY	Group
NAME	Logical
LENGTH	Numeric
WIDTH	Numeric
LIGHTS	Group
NAME	Logical
COLOR	Logical
NUMBER	Numeric

2. CITY File

OBJECT NAME	
LAT	Numeric
LONG	Numeric
POPU	Numeric
STATE	Logical

APPENDIX III

AN IFGL FILE DESCRIPTION EXAMPLE

1. FILE SPECIFICATION

```
GENERATE FILE, AIRPORT,
ESTIMATED LENGTH 5Ø PAGES,
SCR,
CLASSIFICATION COMPANY CONFIDENTIAL.

BEGIN OBJECT.
SET SEQUENCE COUNTER TO Ø4Ø.
SPACE 1 CARD. SPACE TO NON '*'.

LOGICAL, OBJECT NAME,
LENGTH IS 2Ø COLUMNS.
CONVERT USING CAA.
USE OBJECT ROLL.

SEQUENCE CHECK,
LENGTH IS 3 COLUMNS.
CHECK FOR SEQUENCE NUMBER GREATER PREVIOUS.

LOGICAL, CITY,
SPACE 1Ø COLUMNS.
LENGTH IS VARIABLE, SCAN UP TO '**'.
PROTECT.
USE NEW ROLL LOCATION.

INTEGER, NUMBER OF RUNWAYS,
SPACE 2 COLUMNS. SPACE TO NON ' '.
LENGTH IS 3 COLUMNS.
CONVERT USING CDB.
MAX 1ØØ, MIN 1, 3 DIGITS.

BEGIN GROUP, AIRLINES  (AL), TERMINATED BY '***'.
SPACE TO NEXT CARD. SPACE 25 COLUMNS.
BEGIN REPETITION.

LOGICAL, NAME,
SPACE BACKWARD TO ' '. SPACE 1 COLUMN.
LENGTH IS VARIABLE, SCAN UP TO ' '.
USE OBJECT ROLL OF COMPANY FILE.
```

```
DECIMAL, NUMBER OF FLIGHTS,
SPACE TO NON ' ' .
LENGTH IS 4 COLUMNS.
4 DIGITS.

INTEGER, NUMBER OF PLANES,
LENGTH IS 5 COLUMNS.
5 DIGITS.

RAW, CODE NAME,
SPACE TO '*'. SPACE TO 'A' OR 'B'.
LENGTH IS VARIABLE, SCAN UP TO 'Ø' OR '*'.
CONVERT USING CAA.
PROTECT.
PRINT STANDARD.

END REPETITION.
SPACE TO NEXT CARD. SPACE 25 COLUMNS.
END GROUP, AIRLINES.

END OBJECT.
SPACE TO NEXT CARD. SPACE TO NON '*'.
```

THE MITRE CORPORATION — 7030 CODING FORM — /; - 114

2. EXAMPLE OF INPUT DATA CARDS

AUTHOR	PROJECT	DATE	PAGE ____ OF ____

NAME | STATEMENT: INPUT DATA CARDS FOR IFGL FILE DESCRIPTION | IDENTIFICATION

```
         THIS CARD MAY BE BLANK
•••LOGAN                    041          BOSTON••
   019
                     EASTERN 0125    80              • AE25  •
                    AMERICAN 0130    60              • AA26  0
                      MOHAWK 0115    34              • AM35  •
                          •••
•••KENNEDY                  042          NEW YORK••
   028
                         TWA 0225   105              • BT85  •
                    AMERICAN 0208    93              • AA26  0
                          •••
B EOF
```

MCP 846-1

3. EXAMPLE OF FILE GENERATION PRINTOUT (ADAM STANDARD FORMAT)

```
                                        COMPANY CONFIDENTIAL

                                                                                    PAGE   1
18.11.26

 GENERATE FILE , AIRPORT , ESTIMATED LENGTH 50 PAGES , SCR , CLASSIFICATION COMPANY CONFIDENTIAL . BEGIN OBJECT . SET SEQUENCE COUNT
ER TO 040 . SPACE 1 CARD . SPACE TO NON '*' . LOGICAL , OBJECT NAME , LENGTH IS 20 COLUMNS . CONVERT USING CAA . USE OBJECT ROLL . S
EQUENCE CHECK , LENGTH IS 3 COLUMNS . CHECK FOR SEQUENCE NUMBER GREATER PREVIOUS . LOGICAL , CITY , SPACE 10 COLUMNS . LENGTH IS VAR
IABLE , SCAN UP TO '**' . PROTECT . USE NEW ROLL LOCATION . INTEGER , NUMBER OF RUNWAYS , SPACE 2 COLUMNS . SPACE TO NON ' ' . LENGT
H IS 3 COLUMNS . CONVERT USING COB . MAX 100 , MIN 1 , 3 DIGITS . BEGIN GROUP , AIRLINES ( AL ) , TERMINATED BY '****' . SPACE TO NEX
T CARD . SPACE 25 COLUMNS . BEGIN REPETITION . LOGICAL , NAME , SPACE BACKWARD TO ' ' . SPACE 1 COLUMN . LENGTH IS VARIABLE , SCAN U
P TO ' ' . USE OBJECT ROLL OF COMPANY FILE . DECIMAL , NUMBER OF FLIGHTS , SPACE TO NON ' ' . LENGTH IS 4 COLUMNS . 4 DIGITS . INTEG
ER , NUMBER OF PLANES , LENGTH IS 5 COLUMNS . 5 DIGITS . RAW , CODE NAME , SPACE TO '*' . SPACE TO 'A' OR 'B' . LENGTH IS VARIABLE ,
 SCAN UP TO '0' OR '*' . CONVERT USING CAA . PROTECT . PRINT STANDARD . END REPETITION . SPACE TO NEXT CARD . SPACE 25 COLUMNS . END
 GROUP , AIRLINES . END OBJECT . SPACE TO NEXT CARD . SPACE TO NON '*' .


LOGAN
  CITY BOSTON
  NUMBER OF RUNWAYS  19
  AIRLINES EASTERN
    NUMBER OF FLIGHTS  125
    NUMBER OF PLANES  80
    CODE NAME AE25
  AIRLINES AMERICAN
    NUMBER OF FLIGHTS  130
    NUMBER OF PLANES  60
    CODE NAME AA26
  AIRLINES MOHAWK
    NUMBER OF FLIGHTS  115
    NUMBER OF PLANES  34
    CODE NAME AM35

KENNEDY
  CITY NEW YORK
  NUMBER OF RUNWAYS  28
  AIRLINES TWA
    NUMBER OF FLIGHTS  225
    NUMBER OF PLANES  105
    CODE NAME BT85
  AIRLINES AMERICAN
    NUMBER OF FLIGHTS  268
    NUMBER OF PLANES  93
    CODE NAME AA26

18.11.27


                                        COMPANY CONFIDENTIAL
```

APPENDIX IV

SOME EXAMPLES

FILLER WORDS

A substitution may be defined as blank; for example,

LET THE MEAN ( ) . ∤

Filler words may be defined thus. Note the danger of defining words such as AND or OR this way: they will always be substituted for and you may need them in other messages with Boolean conditionals.

KEYWORDS USED BY OTHER KEYWORD DEFINITIONS

Since RESCAN allows keywords to be used in the strings defined for other keywords, entire messages may be stored as string substitutions with portions of the messages altered dynamically. For example, a user may have defined many keywords which specify messages that include a phrase requesting a printout with the date in the title, such as

-----PRINT TITLE 'SUMMARY FOR DATE' ---

or ---DISPLAY TITLE 'STATUS ON DATE'---

Here DATE could be a keyword to be redefined by the user daily with inputs of the form:

SCRUB DATE
LET DATE MEAN (1 JANUARY 1966)

All messages which use the keyword DATE would automatically be updated by substitution of the current definition of DATE.

STRING SUBSTITUTION FOR SPECIFIC DEVICES

Suppose that the user wishes to be able to both print and display the population of all cities on the associated printer and display when he types POPU on any typewriter. The following sequence will do it:

> LET POPU MEAN (FOR CITY. SHOW POPU.) FOR ALL USING RESCAN.
> LET SHOW MEAN (OUTPUT P1 D1) FOR T1.
> LET SHOW MEAN (OUTPUT P2 D2) FOR T2.

etc., for each typewriter.

A STRING SUBSTITUTION WHICH ACCEPTS A VARIABLE NUMBER OF PARAMETERS FOR INSERTION

Although the REINSERT operation requires that the parameters to be inserted be enumerated specifically, nested REINSERT's can be used where a variable number of parameters are to be inserted into a message with other words in between.

The following example of an application of string substitution has a variable number of parameters to be inserted into the message. The problem was to define a sequence of string substitutions to change the message phrase

ZERO (A, B, ... , X)

into a phrase of the form:

CHANGE A TO O, B TO O, ... X TO O

where there can be any number of parameters (A. . . X) as long as there is at least one. The parentheses are necessary when there is more than one parameter. This particular definition is a little different from the general case in that the expansion for the first parameter is not quite the same as for the

remaining parameters; i.e., A results in CHANGE A TO O while the rest produce, B TO O, etc.

The sequence of definitions follow:

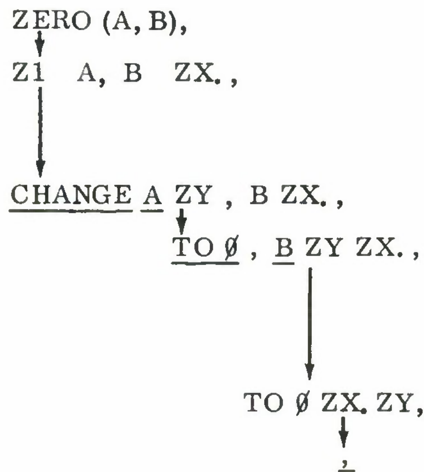
LET ZERO MEAN (Z1 /1/ ZX.) FOR ALL USING REINSERT.

LET Z1 MEAN (CHANGE /1/ ZY) USING REINSERT.

LET ZY MEAN (TO Ø /1/ /2/ ZY) USING REINSERT.

LET ZX MEAN (/3/) USING REINSERT.

A simple example illustrates the use of these definitions:



Input Message Phrase

Substitution for ZERO. ZX is a terminator for the variable number of parameters. Note that parentheses were stripped.

Substitution for Z1.

Substitution for ZY. This repeats until ZX appears before ZY. This condition occurs when /2/ is no longer a comma between two parameters in the list.

Substitution for ZY.

Substitution for ZX is necessary to skip over the . ZY without putting them into the message. Any string substitution with insertions discards all input words up to and including the highest numbered parameter after insertion. /3/ skips the period and ZY to the parameter following ZERO (---) and scanning proceeds from there.

The final result reads: CHANGE A TO Ø, B to Ø, ---

A STRING SUBSTITUTION FOR SPECIFYING PARAMETERS TO A ROUTINE

Consider a routine PRESORT which has the following input parameters in the specified order:

(a) Name of file to be sorted.

(b) 'name': name of property name on whose values the file would be sorted.

(c) Sort order: Ø = ascending, 1 = descending.

(d) More pairs of parameters of the form described in (2) and (3) if there are more sort keys; otherwise no more are necessary.

(e) 'name': name of sorted file if not the original file or dummy name if sorted data goes into original file.

(f) Output indication

  (1) Sorted file will be the original file

  (2) Sorted file is created as a new file with the name indicated in (c).

A message of the form:

SORT filename ON $prop_1$ $order_1$[, THEN $prop_2$ $order_2$ ---, THEN $prop_n$ $order_n$].

$\left\{ \begin{array}{l} \text{SAME} \\ \text{CALL IT new name} \end{array} \right\}$.

may be used, where $prop_i$ = property name and $order_i$ = ASCEN or DESCN, to produce a FABLE message of the form:

DO PRESORT (name, '$prop_1$', sort order-----).

The following string substitutions accomplish this:

(∇ represents precisely one required blank)

```
LET SORT MEAN (DO PRESORT (/1/, '▽/▽3▽/▽', /4/6/)) FOR ALL
     USING REINSERT.
LET THEN MEAN (,'▽/▽2▽/▽', /3/ /5/ /1/) FOR ALL USING REINSERT.
LET SAME MEAN (,'X', -1/1/) FOR ALL USING INSERT.
LET CALL MEAN (,'▽/▽3▽/▽', -2/1/) FOR ALL USING INSERT.
LET ASCEN MEAN (Ø) FOR ALL USING SCAN.
LET DESCN MEAN (1) FOR ALL USING SCAN.
```

The blanks must be supplied because separators are not introduced by ADAM system processing between pairs of primes (').

ANOTHER EXAMPLE OF THE USE OF STRING SUBSTITUTIONS

Another example of the use of string substitutions is given below. The user can verify the results by himself.

String substitutions are defined to transform a message of the form

TALLY filename numeric property name (range spec, range spec----
range spec).

where 'range spec' is a relational operator and a number; e.g.,

LS 1ØØ or GR Ø

into a sequence of FABLE statements that count the number of values of the named property in each range where the value is counted in the first range (from left to right) that it satisfies.

```
LET TALLY MEAN (FOR C 1.SAVE 'SUBT' = Ø.NAME T.RA /3/ TT (/1/) (/2/))
                    USING REINSERT.
```

The string for TALLY introduces RA, another keyword followed by the ranges, and TT ___ a keyword which marks the end of the ranges. The parenthesized insertions, (/1/) and (/2/) are used to carry along for insertion in subsequent

string substitutions the file and property names. They are in parentheses in case the names are more than one word so each name will be only one parameter.

```
LET RA MEAN (FOR T. ELSE, SAVE '▽/▽1▽/▽/▽2▽/▽' = 0
     /3/ RB (/1/ /2/)) USING REINSERT.
```

/3/ will be a comma if there is another range spec. Otherwise it will be TT. Ranges are being saved in (/1/ /2/) for insertion in subsequent keywords.

```
LET RB MEAN ('▽/▽2▽/▽/▽3▽/▽' = Ø /4/ RB
     (/1/ or /2/ /3/)) USING REINSERT.
```

RB is repeated for each additional range spec. Additional range specs are accumulated by (/1/ OR /2/ /3/) with OR between each. The substitution for RB is stopped when there are no more range specs because /4/ becomes TT.

```
LET TT MEAN (.NAME T1. FOR T1.ALL(FOR /3/.
TB (/3/ /4/ /2/ TC)) FOR ALL USING REINSERT.

LET TB MEAN (/1/ /2/ /3/ /4/ CHANGE /3/ /4/ TO T1 /3/ /4/ +
     1 /5/ TB (/1/) (/2/)) USING REINSERT.

LET TC MEAN (OR ELSE), DISPLAY FORMAT TALLY
     TITLE '▽/▽2▽/▽/▽3▽/▽' ALL. DELETE T1 FILE.
     DELETE T FILE) USING REINSERT.
```

APPENDIX V

MESSAGES TO USER FROM STRING SUBSTITUTION

NORMAL STRING SUBSTITUTION DEFINITION OR REMOVAL

STRING SUBSTITUTION MESSAGE ACCEPTED

SYNTAX ERROR IN STRING SUBSTITUTION DEFINITION OR REMOVAL

KEYWORD NOT FOLLOWED BY 'MEAN'. e.g., LET X BE
(PRINT --- NO LEFT PAREN AFTER 'MEAN'. e.g., LET
X MEAN Y NO RIGHT PAREN, e.g., LET X MEAN (Y+Z .
INVALID 'USING' PHRASE, e.g., LET X MEAN (Y) USING RESCAM
NO DEVICE LIST AFTER 'FOR'. e.g., LET X MEAN (Y) FOR .
ILLEGAL DEVICE NAME, e.g., SCRUB X FROM D7
WORD AFTER ALL NOT 'USING', e.g., LET X MEAN (Y)
FOR ALL USINT

ERROR IN KEYWORD BEING DEFINED

KEYWORD ALREADY DEFINED FOR DEVICE
KEYWORD ALREADY DEFINED FOR DEVICE, NOT AVAILABLE
FOR 'ALL'

ERROR IN KEYWORD BEING SCRUBBED

KEYWORD IS NOT DEFINED
KEYWORD IS NOT DEFINED FOR DEVICE

ERROR IN USE OF A KEYWORD

INSUFFICIENT NUMBER OF PARAMETERS. e.g.,
LET X MEAN (/1/*/4/).
X Y Z .] There are only three parameters before the end-of-message.

MESSAGE CONTAINS TOO MANY SUBSTITUTIONS, e.g.,

LET A MEAN (A) USING RESCAN.

A will loop indefinitely substituting A for A. To prevent this, a maximum number of substitutions for a message is defined; if it is exceeded, substitution stops on an error condition.

APPENDIX VI

FORMATTING OPERATORS

The notation s/p/ℓ/c stands for the

section/page/line/column

at the time a formatting operator is encountered.

The notation smax, pmax, ℓmax, cmax stands for the maximum section, page, line, or column thus far attained in the format.

Throughout,

f	Stands for field length which, if omitted, will be set to "variable".
n	Stands for a count (of columns, lines, etc.) which, if omitted, will be set to 1.
k	Stands for a count (of columns, etc.) which, if omitted, will be set to Ø.

f, n, and k must all be positive or zero.

ALPHABETICAL INDEX

OPERATOR	STRAP CODE	PAGE
B	†XW,3.Ø	52
BEGIN	----------	34
BT	†XW,15	52
COL	XW,13.Ø	35
DATE	XW,16.Ø,f	47
DO	†XW,26.32	51
END	XW,8.32,n	34
F	XW,8.Ø,f	41
F*	XW,27.32	35
H*	XW,11.32	50
IO	XW,12.Ø	55
LIT	†XW,2.32,f	46
LM*	XW,19.32	36
LPI*	XW,23.32	53
L*	XW,7.32	45
MD*	XW,21.32	50
N	XW,4.32,f	38
NPDV	XW,2Ø.Ø	38
NXP	XW,9.Ø,n	49
NXS	XW,9.32,n	49
OBJ	XW,24.32,f	41
PAUSE	XW,28.32	55
PDV	XW,2Ø.32	38
PP	XW,15.32	34
P*	XW,4.Ø	43
Q	XW,6.32,f	41
QY	XW,1Ø.32,f	47
Q*	XW,1.	44
RAW	XW,13.32	35
RA*	XW,26	48
RC	XW,18.32	38
RCOL	XW,28.Ø	40
RM*	XW,25.0	36
ROW	XW,12.32	35

OPERATOR	STRAP CODE	PAGE
S	XW,1.32,n	39
SH*	XW,14.32	50
ST	----------	56
T	----------	56
TIME	XW,16.32,f	47
TL	XW,1Ø.Ø,f	47
TRU*	XW,22.32	48
U	XW,5.32,f	38
V	XW,5.Ø,f	38
VC	†XW,5.Ø,f	51
VC*	XW,29.Ø	42
V*	XW,29.Ø	42
*BM	XW,24.Ø,k	36
*F	XW,27	35
*H	XW,11.Ø	50
*L	XW,7.Ø,n	45
*LM	XW,19.Ø,k	36
*LPI	XW,23.Ø	11
*MD	XW,21.Ø	50
*P	XW,3.32	43
*Q	XW,Ø.32	44
*RA	XW,25.32	48
*RM	XW,6.Ø,k	36
*SH	XW,14.Ø	50
*TRU	XW,22.Ø	48
*V	†XW,5.Ø,f	42
*VC	†XW,5.Ø,f	42
*--	†XW,17.32	54
*..	†XW,17.Ø	54
--RM	XW,18.Ø	54
--*	----------	54
		54
..*	----------	40
/	XW,2.Ø,n	

†Further information required.

STRAP CODE

NUMERICAL INDEX

OPERATOR	STRAP CODE	PAGE
*Q	XW,Ø.32	44
Q*	XW,1.	44
S	XW,1.32,n	39
/	XW,2.Ø,n	40
LIT	†XW,2.32,f	56
B	†XW,3.Ø	52
*P	XW,3.32	43
P*	XW,4.Ø	43
N	XW,4.32,f	38
*V	†XW,5.Ø,f	42
*VC	†XW,5.Ø,f	42
V	XW,5.Ø,f	38
VC	†XW,5.Ø,f	51
U	XW,5.32,f	32
*RM	XW,6.Ø,k	36
Q	XW,6.32,f	41
*L	XW,7.Ø,n	45
L*	XW,7.32	45
F	XW,8.Ø,f	41
END	XW,8.32,n	34
NXP	XW,9.Ø,n	49
NXS	XW,9.32,n	49
TL	XW,1Ø.Ø,f	47
QY	XW,1Ø.32,f	47
*H	XW,11.Ø	50
H*	XW,11.32	50
IO	XW,12.Ø	55
ROW	XW,12.32	35
COL	XW,13.Ø	35
RAW	XW,13.32	35
*SH	XW,14.Ø	50
SH*	XW,14.32	50
BT	†XW,15	52
PP	XW,15.32	34

OPERATOR	STRAP CODE	PAGE
DATE	XW,16.Ø,f	47
TIME	XW,16.32,f	47
*..	†XW,17.Ø	54
*--	†XW,17.32	54
--RM	XW,18.Ø	54
RC	XW,18.32	38
*LM	XW,19.Ø,k	36
LM*	XW,19.32	36
NPDV	XW,2Ø.Ø	38
PDV	XW,2Ø.32	38
*MD	XW,21.Ø	50
MD*	XW,21.32	50
*TRU	XW,22.Ø	48
TRU*	XW,22.32	48
*LPI	XW,23.Ø	53
LPI*	XW,23.32	53
*BM	XW,24.Ø,k	36
OBJ	XW,24.32,f	41
RM*	XW,25.Ø	36
*RA	XW,25.32	48
RA*	XW,26	48
DO	†XW,26.32	51
*F	XW,27	35
F*	XW,27.32	35
RCOL	XW,28.Ø	40
PAUSE	XW,28.32	55
VC*	XW,29.Ø	42
V*	XW,29.Ø	42
	XW,29.32	57
	XW,30.0	57
BEGIN	----------	34
ST	----------	56
T	----------	56
--*	----------	54
..*	----------	54

†Further information required

INDEX BY PAGE NUMBER

OPERATOR	STRAP CODE	PAGE
BEGIN	----------	34
END	XW,8.32,n	34
PP	XW,15.32	34
ROW	XW,12.32	35
COL	XW,13.Ø	35
RAW	XW,13.32	35
*F	XW,27	35
F*	XW,27.32	35
*RM	XW,6.Ø,k	36
*LM	XW,19.Ø,k	36
*BM	XW,24.Ø,k	36
RM*	XW,25.Ø	36
LM*	XW,19.32	36
N	XW,4.32,f	37
V	XW,5.Ø,f	37
U	XW,5.32,f	37
RC	XW,18.32	37
NPDV	XW,2Ø.Ø	37
PDV	XW,2Ø.32	37
S	XW,1.32,n	38
/	XW,2.Ø,n	39
RCOL	XW,28.Ø	39
F	XW,8.Ø,f	40
Q	XW,6.32,f	40
OBJ	XW,24.32,f	40
*V	†XW,5.Ø,f	41
V*	XW,29.Ø	41
*VC	†XW,5.Ø,f	41
VC*	XW,29.Ø	41
*P	XW,3.32	42
P*	XW,4.Ø	42
*Q	XW,Ø.32	43
Q*	XW,1.	43
*L	XW,7.Ø,n	44
L*	XW,7.32	44
LIT	†XW,2.32,f	46
DATE	XW,16.Ø,f	47
TIME	XW,16.32,f	47
TL	XW,1Ø.Ø,f	47
QY	XW,1Ø.32,f	47
*RA	XW,25.32	48
RA*	XW,26	48
*TRU	XW,22.Ø	48
TRU*	XW,22.32	48
NXP	XW,9.Ø,n	49
NXS	XW,9.32,n	49
*H	XW,11.Ø	50
H*	XW,11.32	50
*SH	XW,14.Ø	50
SH*	XW,14.32	50
*MD	XW,21.Ø	50
MD*	XW,21.32	50
DO	†XW,26.32	51
VC	†XW,5.Ø,f	51
B	†XW,3.Ø	52
BT	†XW,15	52
*LPI	XW,23.Ø	53
LPI*	XW,23.32	53
*..	†XW,17.Ø	54
..*	----------	54
*--	†XW,17.32	54
--*	----------	54
--RM	XW,18.Ø	54
IO	XW,12.Ø	55
PAUSE	XW,28.32	55
ST	----------	56
T	----------	56
	XW,29.32	57
	XW,30.0	57

†Further information required.

BEGIN
END
PP

BEGIN(name,PP)

or

BEGIN(name,)

or

BEGIN(name)

sets the initial s/p/ℓ/c coordinates to Ø/Ø/Ø/Ø and causes the MOF macro to produce a card with

T FORMAT,name

in which the T is in column 1 and the F in column 10. If PP is specified, the first operator in the format specifies page numbering and the top margin is set to line two. Page numbers appear as PAGEddd at the extreme right edge of every page, on the first line.

END(n)

defines the end of a format. The number n specifies the number of copies to be output. END also causes the MOF macro to produce two terminal cards for convenience in deck separation: the first has asterisks in columns 1-1Ø inclusive and the second has

T END OF FORMAT name

in which the T is in column 1 and name is the name specified on the BEGIN card.

Discussion

1. There must be only one BEGIN and one END operator in each format; several formats can be compiled in the same SMAC subtype however.
2. The entire format is repeated for each different device-type specified in the input.

ROW
COL
RAW
*F
F*

ROW	Set the format type
COL	as the operator
RAW	indicates.
*F	Begin forced sectioning.
F*	End forced sectioning.

If the current format-type is ROW, *F and F* are meaningless.

If the format-type is COL or RAW, *F defines right-margin overflow to appear in the next section and F* returns to the usual COL and RAW handling of right-margin overflow.

Discussion

1. Format-type may be changed at any point in a format.
2. Format-type affects: right margin overflow, repeating group indentation.

*RM
*LM
*BM
RM*
LM*

*RM(k)	Set the right margin to the c + k.
*LM(k)	Set the left margin to the c + k.
*BM(k)	Set the bottom margin to the ℓ + k.
RM*	Set the corresponding margin to the value it
LM*	had before the last effective "set margin" operator.

Discussion

1. Note that top margin control is provided by *H and *SH operators.
2. Margins are originally set to zero (left and top) and to "device margin" (right and bottom).
3. Upon formatting output for a device, if a margin-setting operator specifies a margin outside that available on the device, the device margin is used. See Appendix B for device margins.
4. Since margin setups are always less than page width and length, set and unset operations, given when in other than Section zero, page zero, have the same effect as if given Section zero, page zero.
5. To move a margin left of the current coordinates, use /(Ø),*LM(k); higher than the current coordinates, use RCOL(Ø),*BM(k).

Discussion

1. Names, values, and units are counted separately: thus N,N,N,V,V,/,N,V,U will print on the second line the fourth property name, the third property value, and the units associated with the first property.

2. The operation V may not be performed unless the operation *Q has been previously performed.

3. Note that N(Ø), V(Ø), or U(Ø) will bypass the printing of the next name, value, or unit specified.

4. N, V, U are subject to repeating group rules.

5. The standard properties which appear at the beginning of each object are not included in the count of properties, except for the Classification and Alternate Classification (i.e., Object Name, Dead-Space-Bit count, etc., cannot be formatted as properties). Similarly, a format which does not want to print Classification and Alternate Classification should include *Q,...,*L(2),N(Ø),V(Ø),U(Ø),L*,...,Q* or its equivalent.

N
V
U
RC
NPDV
PDV

N(f)	Print a property name in an f column field.
V(f)	Print a property value in an f column field.
U(f)	Print the units associated with a property in an f column field.

as in:

N(1Ø),S,V(1Ø),S,U(1Ø)

Print the next property name, next property value, and next units; each in a ten column field separated by single spaces.

RC Reset counts

Define the "next" property name, "next" property value, and "next" units to be those associated with the first property in the file, as in:

Q,N(1Ø),V(1Ø),RC,N(Ø),V(1Ø),Q

"For each object print the first property name once, the first property value twice, and leave the counts set to print next the second name, second value, and units associated with the first property".

NPDV No-print deleted value

Cause all deleted names and property values to be printed as blanks until a PDV operator.

PDV Print deleted value

Cause all deleted names and property values to be printed as hyphens until an NPDV operator.

S

S(n) Skip n columns

Change the next field from s/p/ℓ/c to the smaller of s/p/ℓ/(c+n)
and s/p/ℓ/right margin

/
RCOL

/(n) Next line (or, carriage return line feed)

as in: ...,/(2)...

"Skip to second line from present position and go to the left margin."

Change the next field from s/p/ℓ/c

to Ø/p/ℓmax/left margin

and cancel the effect of any previous *MD (collect marginal data).

RCOL(n) Next column

as in: ...,RCOL(1),...

Go to the top of the next column.

Change the next field from s/p/ℓ/c

to s/Ø/top margin/(c+n)

Discussion

1. Notice the /(1) means "go to the next line" not "skip a line" and similarly for RCOL(1).

2. /(Ø) and RCOL(Ø) are quite legal and useful especially to set margins leftward or upward.

3. ℓmax will be greater than ℓ+n if:

   (a) Repetitions have been formatted below one another,

   (b) Field overflow has caused subsequent lines to be used.

F
Q
OBJ

F(f) Print file name in sn f column field

Q(f) Print object name in an f column field

as in:

F(1∅),S,*Q,Q(1∅),S,Q*

Print the file name followed by the names of all the objects in the file, each in a ten character field and separarted by s single space.

OBJ(f) Print the nsme OBJECT NAME or designated synonym for it in an f column field.

Diacussion

1. For the operator Q, objecta are treated one at a time with no backup.
2. For the operator OBJ, the characters OBJECT NAME will be printed unless a print synonym appears in the object roll of the file.

*V
V*
*VC
VC*

*V(f) Value in repeating group

Same as V(f) and in addition prevent repeating group stepping until the occurrence of V* or VC*.

V* Step repeating group

Cause repeating group to be stepped. If the property being formatted is not within a repeating group, do nothing.

---

*VC(f) Value with conversion in repeating group

Same as VC(f) and, in addition, prevent repeating group stepping until the occurrence of V* or VC*. Property value should be either: VFL, FP, or CFP; otherwise *VC(f) is treated as *V(f).

VC* Step repeating group

Identical to V*.

Discussion

1. The subject of repeating group stepping is covered in Section VI.

*P	Start a loop through all properties
P*	Step to the next property

as in: *P,N,V,U,P*

For each property, print its name, value and units.

<u>Discussion</u>

1. The order of properties is as described under N, V, and U.

*Q
Q*

*Q	Open a file (i.e., start a loop on all objects)
Q*	Step the file (i.e. end the loop on all objects)

as in: *Q,Q,Q*

"For each object, print its name."

Discussion

1. *Q must be followed eventually by Q*.
2. Note that a V operation may be performed only within a *Q,...,Q* loop.
3. *Q *P may be nested as in

   ...,*Q,*P,N,V,P*,Q*

   "For each object in the file step through all the properties, for each property print its name and value."
4. *P *Q may be nested as in

   ...,*P,*Q,N,V,Q*,...

   "For each property, step through all the objects in the file; for each object print the property name and property value of the current property."

   Note however, that:

   Object loops within property loops take much more time;
   No property may be a repeating group in a property loop which contains an object loop.

*L
L*

*L(n)	Define beginning and end of literal
L*	loop to be passed through n times

as in:

L(3),S,L

Make three spaces.

<u>Discussion</u>

1. Any operators except BEGIN and END may appear in a literal loop.

LIT

LIT(f), character-string	Print a literal string in an f character field

as in:

MOF,LIT(15),TEST OUTPUT

Print the characters TEST OUTPUT in a 15-character field.

character-string includes all the characters and spaces after the comma following LIT up to and including the last nonblank character.

Discussion

1. LIT must be the first and only operation following MOF, but may overflow onto SMAC continuation cards.
2. Note that LIT(Ø), is not allowed.

DATE
TIME
TL
QY

DATE(f)	Print the date in an f column field. The form of the date is DD/MM/YY
TIME(f)	Print the time in an f column field. The form of the time is HH.MM.SS.
TL(f)	Print the title specified for this output in an f column field.
QY(f)	Print the input message in an f column field.

Discussion

1. The title and input message are specified in the call to the formatting program.

*RA
RA*

*TRU
TRU*

*RA	Begin right adjusting
RA*	End right adjusting

For all print operators after *RA and before RA*, if the number of characters to be output is smaller than the field size specified, place the output in the right-most characters of the field.

---

*TRU	Begin truncating
TRU*	End truncating

For all print operators after *TRU and before TRU* if the number of characters to be output is larger than the field size specified, ignore excess characters on the right. In addition, if any field would overflow the right margin, ignore excess characters on the right.

Note that numeric property values are right adjusted regardless of the RA mode.

NXP(n) Next page

as in: NXP(2)

"Skip to the beginning of the second page from the present position."

Change s/p/ℓ/c

to s/p+n/top-margin/c

NXS Next section

Change s/p/ℓ/c

to smax+1/p/ℓ/left-margin

See Section VI for description of pagination and sectioning.

*H
H*

*SH
SH*

*MD
MD*

*H	Begin header
H*	End header

Each time the page number coordinate changes upward, perform all format operators between the last *H and H* and, when completed, set the top margin to the current line plus one.

---

*SH	Begin super header
SH*	End super header

Each time either the page numbers coordinate or section numbers coordinate changes upward, perform all format operators between the last *SH and SH* and, when completed, set the top margin to the current line plus one.

---

*MD	Begin marginal data
MD*	End marginal data

Each time the section number coordinate changes upwsrd perform all format operators between the last *MD and MD* or / and, when completed, set the left margin to the current column plus one.

Discussion

1. A header may be as wide as desired, but a super header should not exceed page width.
2. Note that marginal data may occupy only one line. Therefore / should not appear between *MD and MD*.

DO(f)   (entry-point-number, parameter-count, routine-PV),...
Load the routine with the given PV and execute it with entry-point-number given. If the routine returns any output, print each set of such output in an _f_ character field. In each case, the routine specification must be followed by a number of full-word parameters equal to the parameter count. The meaning of the parameters is established by the routine; the formatting program does not use them.

VC(f)   (entry-point-number, parameter-count, routine-PV),...
Value with conversion, an operator identical to DO except that the formatting program delivers the next property value to the routine executed. Property value should be either: VFL, FP, or CFP; otherwise VC(f) is treated as V(f).

The subject of special routines is covered in Appendix VIII.

B
BT

BT(t,p,d,s)   Branch on device type

Take as the next format operator that with the label:

t	if output is to	a typewriter
p		a printer
d		a display
s		SPR, the off-line printer.

B(name)   Branch

Take as the next format operator that with the label name
as in:

```
                MOF,BT(A,A,B,A)
A               MOF,LIT,----------
                MOF,B(C)
B               MOF,*--,(Ø,Ø),(Ø,1ØØ),--*
C               MOF,...
```

"For displays, draw a vector from Ø,Ø to Ø,1ØØ; for other devices print 1Ø dashes."

Note that T(A) also can be used to define labels.

*LPI
LPI*

*LPI	Begin making light pencil input table.
LPI*	End making light pencil input table.

For all print and display operators between *LPI and LPI* sent to a display device, make the appropriate entries in the light-pencil-input table associated with that device.

—

The light-pencil-input table is discussed in Appendix IX.

*..
..*
*--
--*
--RM

*..	Start points display
..*	End points display

as in:

..,(Ø,Ø),(Ø,1Ø24),(1Ø24,Ø),(512,512),(1Ø24,1Ø24),..

"Display a point at each of the left-bottom, left-top, right-bottom, center, and right-top of the current page and section."

---

*--	Start vector display
--*	End vector display

as in:

--,(Ø,Ø),(Ø,1Ø24),(1Ø24,Ø),(1Ø24,1Ø24),--

"Display a vector which begins at the left-bottom of the display screen, proceeds to the left-top, thence to the right-bottom, thence to the right-top of the current page and section." The resulting output is a large N.

--RM Vector to right margin

Display a horizontal vector from the present print position to the current right-margin.

Discussion

1. Note that there are 64 characters to a line on a display, but 1Ø24 points; each character corresponds to the 16x16 point square in which the character appears in the center.
2. Vector and point output does not change the line/column coordinate settings for printed output.
3. Vector and point output appears only on the current page, unless included in a header or super header.

IO

Immediate output. Send material thus formatted to the output program (as a partial message) then continue formatting. In addition, cancel the effect of any previous *H and *SH operators.

PAUSE

Terminate the processing of this format without issuing any output and save the results to be output as part of, and merged with, the output of the next message.

Discussion

1. The IO operator is intended to allow material which would otherwise appear in Section Ø, last page of the output, to appear in last section, last page and thus follow all other output. When given in any page, other than the last page of the last section thus far formatted, it has the effect of outputting all material thus far formatted and restarting the format at Ø/Ø/ℓ/c. When given in the last page of the last section, it has the effect of defining that page and section as 'page Ø, section Ø' for all further formatting.

2. Note that IO given in a single-section output does not change the appearance of the output.

3. The PAUSE operator is intended to allow the outputs from two or more messages to be overlaid.. The messages must be sequential and should therefore be part of the same input.

4. Output formatted by a format which includes a PAUSE operator and output of subsequent overlaid messages must have devices of only one type specified[1]. The output is actually sent only to the devices specified in the last overlayed output.

---

[1] At present, display-type devices are prohibited.

ST
T

ST( )   Straight code

as in:

ST(XW,1.32)

or   ST(MOF,LIT,ITEMS)

"Send the material within the parentheses, as a card image beginning in column 1Ø, directly to the SMAC compiler."

T(name)   Tag

as in:

MOF,N,T(ABC),V...

which is equivalent to

```
    MOF,N
ABC MOF,V
```

Insert the name in the SMAC label field of a MOF call at the point at which the T appears.

The operator T does not produce a binary operator in the compiled format. It is for use in defining destinations for B and BT operators.

XW,29.32
XW,3Ø.Ø

XW,29.32	**Suppress coordinate sorting**
XW,3Ø.Ø	**Restore coordinate sorting**

Sorting of the coordinates (s/p/ℓ/c) assigned to the file data may be suppressed by the format operator, XW,29.32, and restored by the format operator, XW,3Ø.Ø There are no MOF symbols for these operators.

<u>Discussion</u>

1. Normally the coordinates must be sorted, since OUTFOP assigns them in logical order rather than output order and TOP requires sorted coordinates for hard-copy output.
2. The sort is suppressed in the standard format, since there the coordinates are already in sort.

APPENDIX VII

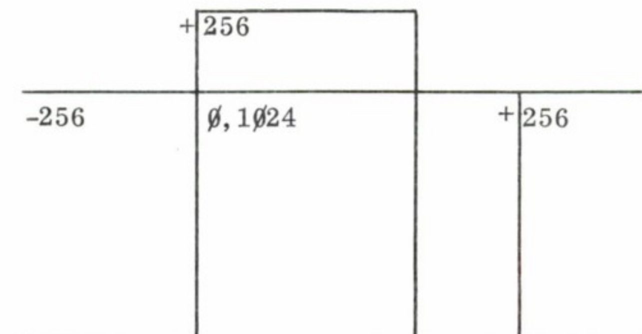
PAGE SIZES FOR VARIOUS DEVICES

The list below contains the maximum page size, in characters, for each output device, and the number of points for display devices. The formatting program processes a format for each device separately and sequentially, and therefore keeps section, page, line, and column coordinates and margin settings individually for each device type. The initial margin settings are Ø, Ø except for the display which is set Ø, 1Ø24; the formatting starts at the upper left corner. The display may have negative values (maximum of -256) and may extend the right, left, and top margin ±256 points beyond the initial margins. In addition, page size is subject to the following considerations for printed output. (For display output the entire current page is always available.)

(1) Vertical page size is reduced by two lines if page numbering is requested, one for the page number and one blank line.

(2) Vertical page size is reduced by four lines if material from a classified file is output, a line at the top for the classification, a blank line, a blank line at the bottom, a line at the bottom for classification.

(3) Although page size in characters is defined for display output as a rectangle and printed output specified by format operators falls within this rectangle, special routines may specify coordinates outside the rectangle but still on the display face. For display output, special routines may change margin settings as illustrated in the following sketch.



SKETCH OF MARGINS FOR DISPLAY

(4) On nonpageable devices, the vertical page size is not used; a single page is as long as the data on it.

TYPE*	DEVICE	PAGE WIDTH	COMMENT
t	typewriter	120	nonpageable
p	SC3070 printer	72	nonpageable
s	SPR (off-line system printer)	132	page length 58 lines
s	FDSPR (nonpageable off-line system printer)	132	nonpageable, otherwise identical to SPR
d	display	64	for points and vectors page is 1∅24 by 1∅24; for characters page length = 64 lines
p	teletype	72	nonpageable
p	flexowriter	72	nonpageable
p	QWISP printer	72	nonpageable

*For use with BT operator

APPENDIX VIII

OUTPUT FORMATTING BY SPECIAL ROUTINES - DO AND VC OPERATORS

GENERAL

An output format in the format file can contain specifications to cause the loading and execution of a routing from the routine file during the formatting process under control of the DO, VC, *VC, and VC* operators.

At the time output is being formatted, when OUTFOP encounters a DO, VC, or *VC specification it delivers to the routine executed, the

(a) current section number,

(b) current X coordinate,

(c) current page number,

(d) current Y coordinate

(e) current margin settings: top, left, right, and bottom;

(f) any explicit parameters declared in the format specification. (The meaning of these parameters is established by the routine; OUTFOP does not use them.)

In addition, if the specification was VC or *VC, OUTFOP delivers:

(g) the numerical value of the next property to be output.

When the called routine returns to OUTFOP, it may specify any output acceptable to TOP and, in addition, may change any of the values (a) through (f) above. When the called routine operates, formatting is in process but no output has yet been sent to TOP[1]; the output it returns to OUTFOP will be included in the formatted output and sent to TOP at the appropriate time.

---

[1] Unless an IO formal element has been encountered.

Each time a routine returns to OUTFOP it must specify whether it is finished, in which case formatting continues, or not finished, in which case OUTFOP accepts the TOP output and returns to the called routine.

The special routine specification may include a column count -- if it does, character output returned by the routine will be output in that number of columns. Thus,

VC(1Ø)...

means operate a special routine and, if it returns with output, use ten columns for the output. Each time the routine returns with output, ten more columns will be used.

CALLING SEQUENCE TO SPECIAL ROUTINE

```
SIC,RETURN$A
B,routine
error-exit-half-word
normal exit.
```

$13VF = Location of input area - absolute full-word address.

$4VF = Location of word Ø of the VC or DO expansion with the parameter area starting in relative word 3.Ø--msaveable full-word address.

$14VF = To be filled by routine with location of its output - full-word address. Vector and point output must be immediately preceded by one full word for use by OUTFOP, but $14VF points to actual data.

$15VF = Entry number as specified in format.

$1 = Points to device roll subval describing currently used output device.

$5 = Pointer to the subval of the property to be converted (No meaning if DO) --munsaveable word.

$10VF = Absolute location of the working tables.

$9 = Munsaveable pointer to location of two pages of temporary working storage. This area is not inviolate and may be used between calls to special routines. (Note: OUTFOP will not accept output in excess of 511 computer words.)

Input Area

Word Ø	bits	Ø - 17	Pointer to Data Location.
		18 - 63	Not used.
Word 1	bits	Ø - 19	Current X coordinate.
		2Ø - 39	Current Y coordinate. First 8-bits of coordinates give page number, last 12 give location within that page.
		4Ø - 51	Current top margin[1].
		52 - 63	Current left margin[1].
Word 2	bits	Ø - 11	Current right margin[1].
		12 - 19	Filled-in (by routine operated) with TOP command character (optional).
		2Ø	Filled-in (by routine operated) with return option: Ø = routine is finished, expects no return, 1 = routine expects to be operated again immediately (required).
		21 - 32	Current bottom margin[1].

Data Location

This full word contains a floating point value of a numerical property; if property is not numerical, contents are meaningless. If routine is executed by DO instead of VC, contents are meaningless.

VC or DO Expansion Area

Word Ø	XW, sl, c, s2	s1	Together specify the OUTFOP code
		s2	for the format operation; e.g., sl = 26.32, s2 = Ø specifies DO
		c	column count.

---

[1] The margin settings are signed numbers (B, 12, 1) that define the columns and rows currently delimiting a page; hard copy devices may not have negative settings.

Word 1	XW,		This word is reserved for OUTFOP.
Word 2	XW, epn, pc, pv	epn	Entry-point-number in routine to be operated
		pc	Parameter count--number of full words (rounded up, i.e., three and a half = four) of parameters to follow.
		pv	Pv of routine to be operated.
Word 3 (3+pc)			Any arbitrary parameters specified in format.

Routine Output

Input area word 2 bit 2Ø and word 2 bits 12 - 19 are filled in by the routine with the following information:

(a) Routine must fill in the return option in the input area (word 2, bit 2Ø): OUTFOP does not reset this bit when the routine sets the bit to 1; OUTFOP does set the bit and the TOP command character to Ø when encountering a DO or VC operator, and in addition, sets the TOP op code to Ø.

(b) If the routine supplies a value to be output, routine must fill in TOP command character (cited below) in word 2, bits 12 - 19 of the input area, unless standard A8 code is Ø. Value will be output in the number of spaces specified in the format.

(c) On exit, the routine sets up:

$14VF = Location of output

$14CF = Ø means no output generated; otherwise $14CF is an output size whose meaning depends on the TOP command character set into the input area as follows:

TOP COMMAND	TOP COMMAND CHARACTER	$14CF
Output until termination character	Ø or 2	Number of bits of output
Output repeated	1 or 3	Number of times to output 8 bits
Display points or vectors	4 or 5 or 15	Number of points or vectors
Display depending on switches	7 or 14	Number of switches
-- All Others --	-----------	Meaningless.

<u>Device Roll Information</u>

Information in the device subval may be used for:

(a) Spacing - by using the incremental information in the device subval, column spacing may be accomplished with the following coding.

```
L(B,12,1) ,1.Ø($1) ,68'          X Increment
*(U) ,3.Ø($4)'                   Parameter
M+(B,12,1) ,1.Ø8($13),68'        Current Column
```

where 3.Ø($4) contents are DD(U),NXØ; N being the number of columns to space; N may be negative.

(b) Decisions - by using the device roll information regarding device type, graphs can be drawn on the display - with vectors -- and on the printer - with an appropriate A8 character -- with the same format. Further levels of decision may be accomplished by using the subval device number as a branch table entry.

Property Subvals

Information in the property subval allows the formatter to construct:

(a) Graphing - generalized graphing with routines is possible since the maximum and minimum of a property are available.

(b) Special outputs - for example, special conversion routine to print the names and maximum and minimum value of the numeric properties of a file. This information is useful for analysis and as an aid in formatting.

Format of the property subval buffer:

Word		Bits	Contents
Word Ø	bits	Ø - 23	Relative position in object or RG
		24	L
		25 - 3Ø	Ø
		31	C
		32 - 46	Ø
		47 - 5Ø	TYPE
		51 - 55	Ø
		56	G
		57 - 63	Ø
Word 1	bits	Ø - 11	Exponent 38 (optional) with Ø in bits Ø and 11
		12 - 33	Ø
		34	Ø
		35 - 49	Group PV
		5Ø - 63	Ø
Word 2	bits	Ø - 11	Exponent 38 (optional) with Ø in bits Ø and 11
		12 - 33	Ø
		34	Ø
		35 - 49	Group PV
		5Ø - 63	Ø
Word 3	bits	Ø - 11	Ø
		12 - 17	LENGTH
		18 - 2Ø	BYTE
		21 - 27	OFFSET

		28 - 34	Ø
		35 - 49	Length of fixed RG
		5Ø - 55	Ø
		56	P
		57 - 63	Ø
Word 4	bits	Ø - 11	Exponent 38 (optional) with Ø in bits Ø and 11
		12 - 33	Ø
		34	Ø
		35 - 49	Name roll PV
		5Ø - 63	Ø
Word 5	bits	Ø - 11	Exponent 38 (optional) with Ø in bits Ø and 11
		12 - 33	Ø
		34	U
		35 - 49	Units PV
		5Ø - 63	Ø
Word 6	bits	Ø - 11	Exponent 38 (optional) with Ø in bits Ø and 11
		12 - 33	Ø
		34	Ø
		35 - 49	Print PV
		5Ø - 63	Ø
Word 7	bits	Ø - 11	Exponent 38 (optional) with Ø in bits Ø and 11
		12 - 33	Ø
		34	Ø
		35 - 49	PV of last subproperty
		5Ø - 63	Ø
Word 8			RANGE if TYPE is numeric, format is either FP, CFP, or VFL
Word 9			(Same as for Word 8)
Word 10	bits	Ø - 2	Ø
		3 - 17	Branch - to convert-out-routine (if any)
		18 - 34	Ø
		35 - 5Ø	Entry-point-number
		51 - 63	Ø
Word 11	bits	Ø - 2	Ø
		3 - 17	PV of output conversion routine
		18 - 34	Ø
		35 - 5Ø	Entry-point-number
		51 - 63	Ø

Word 12	bits	∅ - 2	∅
		3 - 17	Reserved
		18 - 34	∅
		35 - 5∅	Reserved
		51 - 63	∅
Word 13	bits	∅ - 11	Exponent 38
		12 - 34	∅
		35 - 49	PV of property
		5∅ - 59	∅
		6∅	Sign
		61 - 63	∅
Word 14	bits	∅ - 17	Relative location in format working tables of property name
		18 - 31	∅
		32 - 49	Bit count of the name
		5∅ - 63	∅
Word 15			Temporary storage available for user
			{ Used by format program if TYPE is NRG or URG
Word 16			Temporary storage available for user

MACRO STATEMENTS

The macro statements are in the general format prescribed for MOF.

MOF, DO(n), (entry-point-number, parameter-count, routine-pv).
MOF, VC(n), (entry-point-number, parameter-count, routine-pv).
MOF, *VC(n), (entry-point-number, parameter-count, routine-pv).

MOF, VC*

In each case, the macro must be followed by a number of full words of parameters equal to the parameters count. The parameters may be generated by SMAC or STRAP code following the MOF statement or by the ST ( ) operator of MOF. Use of the DO or VC operators causes the routine specified to be operated with the entry-point-number specified in $15VF.

The macro operations may appear on the same card with other MOF operations.

The operation *VC operates as *V; i.e., it prevents a repeating group from being stepped until the occurrence of a VC*. The operation VC* does not cause a routine to be called.

EXPANSION

In the examples below, the routine with PV $17_{1\emptyset}$ is loaded and executed in each case. When executed, $13VF points to the input area described under "Calling Sequence" above and $4VF points to the word indicated by the label (A, B, or C in the example).

DO

The expansion of

```
A     MOF, DO(8), (1.32, 2, 17)
      XW, 1.   '                    } These two words are sample
      XW, 2.   '                    } parameters.
```

is

```
A     XW, 26.32, 8, Ø'              26.32 is the code for DO, 8
                                    is the column count
      XW, Ø    , Ø, Ø'              Blank word used by OUTFOP
      XW, 1.32, 2, 17'
      XW, 1.
      XW, 2
```

Any output returned by the routine will be printed in an 8-character field.

VC

The expansion of

```
B     MOF,VC(12) , (Ø, Ø, 17)
```

is

B	XW, 5.Ø, 12, 1'	VF = 5. and RF = 1 are the code for VC, 12 is column count
	XW, Ø, Ø, Ø '	Blank word is used by OUTFOP
	XW, Ø, Ø, 17'	Note entry = Ø and no parameters.

<u>*VC</u>

The expansion of

C MOF, *VC(12), (Ø, Ø, 17)

is

C	XW, 5.Ø, 12, 1 (.27) 1'	Everything except the 12 is the code for *VC, 12 is the column count.
	XW, Ø, Ø, 0 '	
	XW, Ø, Ø, 17'	

<u>VC*</u>

The expansion of

D MOF, VC*

is

D XW, 29.Ø

Note that VC* does not cause a routine to be called. It is identical to the operator V*.

APPENDIX IX

LIGHT PENCIL INPUT STREAM

The format operator *LPI causes the format program to make entries in a stream which describes the output being formatted so that inputs from light-pencil actions against the material as displayed may be interpreted. The operator LPI* prevents further entries from being made. Thus, a display may be activated completely, in part, or not at all by light pencil action.

In any event, each display output has associated with it a "light-pencil index number", which is an address in a light-pencil input stream (LPI stream). A separate LPI stream exists for each display device.

When a (nonraster mode) light-pencil action is taken, the input delivered to the COP program for recognition includes the light-pencil index number instead of the rightmost 18 bits of the 24-bit inquiry word. A zero index corresponds to stream address Ø in the LPI stream, which always contains 64 bits of Ø. Any other index is the stream address of the beginning of an entry in the following format:

(a) Bits Ø - 7 (the descriptor) of an entry describe what is displayed, using the following code:

1 = file name

2 = object name

3 = property name

4 = property value

5 = literal string

6 = point

7 = vector

(b) The contents of the remainder of the entry depends upon the value of the descriptor.

	∅	7	22	63
	1	FILE PV		

	∅	7	22	37	63
OBJECT	2	FILE PV	OBJECT PV		

	∅	7	22	37	63
PROPERTY NAME	3	FILE PV	PROP PV		

	∅	7	22	37	52	63
PROPERTY VALUE	4	FILE PV	OBJECT PV	PROP PV	N	

If the property value displayed is prime level, no other information is saved. If, however, the property value is a RG member or a RG property itself, enough 16-bit bytes containing the name PV or repetition number of its antecedents are provided to enable a trace back to the prime level. These bytes are placed, four to a word, in succeeding words, as follows:

NAME PV OR REP NO HIGHEST PARENT	ETC.	NAME PV OR REP NO GRANDPARENT	NAME PV OR REP NO PARENT

A repeating group property is considered its own parent. The number of name PV or repetition number bytes following is contained in the field marked N in the identifier word.

Following this any associated literal string is stored thus:

Ø	23	28	45	63
RELATIVE BIT ADDRESS OF LITERAL IN NEXT WORD		NO. OF BITS IN LITERAL		

1.Ø		1.63
	LITERAL	STRING

2.Ø n.63

	Ø	7	23	63
LITERALS	5	CHARACTER COUNT	LITERAL STRING	

1.Ø n.63

Here the literal string is assumed to have originated in the format file.

There is other literal information which may be output as a result of the formatting process, such as the query, a title, page numbers, etc. These do not result in entries in the LPI table.

	Ø	7	19	31	63
POINT	6	X(B, 12, 1)	Y(B, 12, 1)		

In this identifier the X-Y coordinates of the point are provided.

	Ø	7	19	31	43	55	63
VECTOR	7	X_o(B, 12, 1)	Y_o(B, 12, 1)	X_1(B, 12, 1)	Y_1(B, 12, 1)		

Here the end points of the vector are provided.

APPENDIX X

DESIGN FEATURES NOT IMPLEMENTED

This appendix contains brief descriptions of some features of the ADAM system design which were not implemented.

PERMANENT ROUTINE DATA

The routine file subdeck (which is part of the standard binary deck) contains two sections: the code section and the permanent data section. The code section, which contains instructions and data, has been described. It is placed into the Routine File and loaded into core whenever a fresh copy of the routine is required. The permanent data section contains initial values for permanent data.

When a routine is added to the Routine File, its permanent data is stored in a disk region called the Routine Data Region or RDR. When a routine is deleted from core, the current values for its permanent data are stored into the RDR. When a routine is deleted from the Routine File its permanent data is deleted from the RDR.

THE RUE CHANGE OPTION

The Routine Pointer Set is a component of the Routine Roll which is permanently in core memory. Entries in the Routine Pointer Set are associated with routines and, by using the PV of the routine, table look-up can be used to find an entry. The ADAM system also contains two other pointer sets which describe files and rolls. If an object is deleted from a pointer set, its PV becomes available for use by new objects. System contamination will occur if other objects refer (by PV) to the deleted object. This difficulty is overcome by associating an integer called a Critical-Mod-Number or

CMN with a pointer set PV. The CMN associated with a PV is changed whenever an object associated with the PV is deleted.

The RUE change option has the following functions:

(a) To completely replace an existing routine with a new version (similar to DELETE followed by ADD). The PV of the routine does not change.

(b) To change the CMN associated with the PV if the routine change is critical.

Some examples of changes to a routine that are critical changes include:

(1) An entry option name is deleted.

(2) The value of a global symbol has changed.

ROUTINE RENOVATION

Routines that are compiled using DAMSEL may incorporate ADAM System data into the compiled binary deck. Such data are systematically tabulated and are described by a renovation directory whose location is placed in the standard routine heading. Two examples of system data that may be incorporated into the binary deck are:

(a) Roll, file, and routine PV's and CMN's

(b) File Property Descriptions.

CLOD will examine the data described by the renovation directory at load time to see if system contamination has occurred. If contamination has occurred, CLOD will attempt to fetch new data from the current data base and incorporate it into the routine.

Unclassified

Security Classification

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) The MITRE Corporation Bedford, Massachusetts	2a. REPORT SECURITY CLASSIFICATION Unclassified 2b. GROUP N/A

3. REPORT TITLE

A USER'S GUIDE TO THE ADAM SYSTEM

4. DESCRIPTIVE NOTES (Type of report and inclusive dates)

N/A

5. AUTHOR(S) (First name, middle initial, last name)

ADAM Project Staff

6. REPORT DATE	7a. TOTAL NO. OF PAGES	7b. NO. OF REFS
December 1967	250	0

8a. CONTRACT OR GRANT NO. AF19(628)-5165 b. PROJECT NO. 502F	9a. ORIGINATOR'S REPORT NUMBER(S) ESD-TR-66-644
c. d.	9b. OTHER REPORT NO(S) (Any other numbers that may be assigned this report) MTR-268

10. DISTRIBUTION STATEMENT

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

11. SUPPLEMENTARY NOTES	12. SPONSORING MILITARY ACTIVITY
N/A	Deputy for Command Systems, Computer and Display Division, Electronic Systems Division, L. G. Hanscom Field, Bedford, Massachusetts.

13. ABSTRACT

This report describes the kinds of capabilities available in the ADAM system and the way in which they are used. The processes for creating and maintaining a data base, specifying formats, modifying the form of the input, and specifying procedures are described. The FABLE, IFGL, and DAMSEL languages are also described.

DD FORM 1 NOV 65 1473

Unclassified

Security Classification

Unclassified

Security Classification

14. KEY WORDS	LINK A		LINK B		LINK C	
	ROLE	WT	ROLE	WT	ROLE	WT
Programming (Computers)						
Generalized Data Management						
File Manipulation						
Information Retrieval						
Command and Management Systems						
Information Processing						

Unclassified

Security Classification