# MUMPS Development Committee

MUMPS
DOCUMENTATION
MANUAL

MUMPS DEVELOPMENT COMMITTEE

Revised
31 August 1976

# MUMPS DOCUMENTATION MANUAL

MUMPS Documentation Manual
ISBN 0-918118-10-7

Distributed by the MUMPS Users' Group, 700 South Euclid Avenue,
St. Louis, MO  63110, USA.

MDC/35
Revised
11/10/76

# MUMPS DOCUMENTATION MANUAL

A Type A Release of the MUMPS Development Committee
Produced by MDC Subcommittee #3, Documentation

[illegible]

Jack Bowie, Chairman
MUMPS Development Committee

## FOREWORD

This document presents MUMPS documentation conventions that were adopted and approved for publication as a Type A release of the MUMPS Development Committee on September 17, 1975. It supersedes the earlier documents with the same title, dated May 16, 1975 and September 17, 1975.

## ACKNOWLEDGEMENTS

This document is the result of a cooperative effort on the part of the MUMPS Development Committee, its Subcommittee on Documentation, and that Subcommittee's Task Group on MUMPS Documentation Standards. The document was prepared by Mr. Larry J. Peck under Purchase Order No. 512577 from the National Bureau of Standards and is based on earlier drafts of material written by Mr. Peck and by Robert A. Greenes, M.D. under Purchase Orders No. 409777 and 409778, respectively, from the National Bureau of Standards. The work was authorized under the terms of an inter-agency agreement between the National Center for Health Services Research, Health Resources Administration, U.S. Department of Health, Education, and Welfare and the Institute for Computer Sciences and Technology, National Bureau of Standards, U.S. Department of Commerce.

**MDC Participants**

| Institution | Participant(s) |
|---|---|
| Advanced Medical Systems Corporation | Stanley M. Rose |
| Artronix, Inc | David A. Bridger |
| B-D Spear | Charles M. Smith |
| Baylor College of Medicine | David B. Brown, Rudolph F. Trost |
| Beth Israel Hospital | Robert F. Beckley III |
| Community EKG Interpretive Service | James Cruce |
| Dartmouth College | William Campbell |
| Department of Health, Education, and Welfare (HRA) | Donald R. Barnes, William V. Glenn, Jr. |
| Department of Health, Education, and Welfare (NIH) | David B. Swedlow |
| Department of Health, Education, and Welfare (NLM) | David C. Hartmann |
| Digital Equipment Corporation | Roger S. Gourd, Robert D. Shear, Elisabeth Sopka |
| Georgetown University | David L. Williams |
| Georgia Institute of Technology | Fred R. Sias |
| Health Care Management Systems, Inc | Leonard L. Hurst |
| Institute of Living | R. Peter Ericson |
| Interpretive Data Systems, Inc | Paul L. Egerman, Carl B. Lazarus, Phillip T. Ragon |
| Jefferson Medical College | Robert F. Curley |
| Massachusetts General Hospital | G. Octo Barnett, Jack Bowie, Norma Justice, Robert L. Rees, Craig J. Richardson |
| Meditech, Inc | A. Neil Pappalardo, Richard J. Pietravalle |
| MITRE Corporation | Richard E. Zapolin |
| National Bureau of Standards | Melvin E. Conway, Edward A. Gardner, Martin E. Johnson, Joseph T. O'Neill |
| Northeastern University | Wendy D. Mela |
| Regional Health Resource Center | Thomas T. Chen, Henry A. Warner |
| Stanford University Medical Center | Russell Briggs, Robert A. Greenes |
| University of California, Davis | Richard F. Walters, Jerome C. Wilcox |
| University of California, San Francisco | David D. Sherertz, Anthony I. Wasserman |
| University of Massachusetts | Jeffrey Rothmeier |
| University of Missouri | James L. Lehr |
| University of Pennsylvania | Martin Pring |
| University of Tennessee | Larry J. Peck |
| University of Washington (Seattle) | Arden W. Forrey |
| University of Wisconsin | Ellis A. Bauman, Gary S. Holmes |
| Washington University (St. Louis) | W. Edward Long, Joan Zimmerman |

# TABLE OF CONTENTS

# PREFACE

The MUMPS Documentation Manual is intended to aid the MUMPS programmer in documenting computer applications written in MUMPS. This manual identifies information required for complete documentation and also provides guidelines for its presentation. The forms, definitions, and recommendations contained herein reflect the work and accomplishments of the MUMPS Development Committee Documentation Subcommittee.

A major portion of this manual (Chapters 1 through 4) is devoted to MUMPS source code documentation. MUMPS source code is addressed at the following three levels:

(1) Package - A collection of programs which function together in an application.
(2) Program - A closed set of routines used to perform a specific task.
(3) Routine - A collection of MUMPS code filed as a single unit.

Chapter 1 introduces the MUMPS Abstract form, designed to be a one page stand-alone document supplying general data on the purpose and scope of a given MUMPS application along with specific details for data items affecting the transportability of the application. Instructions for completing the Abstract along with a sample Abstract are given in the chapter.

Chapter 2 describes and contrasts the two basic approaches to source code documentation: logical structure and physical structure. The concepts of variables at the package level, program level, and routine level are also explained in the chapter.

Chapter 3 describes Package and Program documentation utilizing the MUMPS Package/Program Face Sheet. The physical structure versus logical structure approach is discussed for both packages and programs, and the chapter is concluded with a sample of program documentation.

Chapter 4 discusses the documentation of routines, primarily through a discussion of the MUMPS Routine Documentation form. The physical structure versus logical structure approach is presented, along with a sample documented routine.

Chapter 5 describes the MUMPS File Summary form and gives instructions for its use. The form is used to provide summary information regarding a global file. An example completed form is given in the chapter.

Chapter 6 describes some of the basic concepts of globals and provides a file documentation procedure along with a sample of file documentation.

Chapter 7 discusses the broad spectrum of papers, documents, manuals, etc., which might be generated during the development of a given MUMPS application. The correspondence between the development phase of the application and the documents produced is also discussed.

It is <u>not</u> necessary to read this manual from cover to cover in order to understand the documentation procedures presented. The particulars of MUMPS documentation are contained in the first five chapters of this manual. The remaining two chapters may be read if the reader is also interested in global file concepts, documentation approaches, and an overview of the documentation process. Even when addressing the particulars of MUMPS documentation, it is not really necessary to read the first five chapters in great detail. Chapters 1, 3, 4, and 5 offer a detailed explanation of the use of the documentation forms. Since approximately eighty percent of the items on the forms are self-explanatory, the programmer need only refer to these chapters when a question arises concerning an item on one of the forms. However, since Chapter 2 explains the approaches to code documentation along with variable concepts assumed throughout the manual, it should definitely be read before attempting to utilize the forms for documentation.

# Chapter 1

# THE MUMPS ABSTRACT

## 1.1 Introduction

The MUMPS Abstract was developed by the MUMPS Development Committee Documentation Subcommittee with the intent of providing a standard medium for the exchange of abstracted information on MUMPS routines, programs, and packages. The Abstract was designed to supply general data on the purpose and scope of a given MUMPS application along with specific details for data items affecting the transportability of the application. Although the Abstract was primarily designed to be a stand-alone document, it may be included in the documentation of a program or package as a cover sheet providing summary information. Whenever this is the case, Abstract items (machine, dialect, partition size, etc.) which also appear on the Package/Program Face Sheet need to be completed only once. An example completed Abstract is illustrated on the following page and a blank Abstract is provided in Appendix II.

A library of submitted abstracts has been established and is currently being maintained by the MUMPS Users' Group (700 South Euclid Avenue, St. Louis, Mo. 63110). Interested users can determine the capabilities and potential portability of available applications by simply reviewing these one-page abstracts. Use of the MUMPS Abstract should significantly enhance the exchange of routines, programs, and packages among MUMPS users.

## 1.2 Instructions for Using the MUMPS Abstract

The MUMPS Abstract is divided into the following five parts:

(1) Identification
(2) Source
(3) Requirements
(4) Size
(5) Available Items

The application being abstracted should first be identified as either a routine, program, or package by checking the appropriate box at the top of the Abstract. Then, each of the five parts of the Abstract should be completed as thoroughly as possible. Instructions for each MUMPS Abstract item follow.

### Application Area

Give the application area in which the routine, program, or package would fall. For example: Business, Educational, Medical, etc.

### Short Title

Enter the title or name which the author associates with the routine, program, or package. A one-line description may also be included here if desired.

17 Sept. 1975

# MUMPS ABSTRACT

ROUTINE [ ]  PROGRAM [ ]  PACKAGE [X]

IDENTIFICATION

APPLICATION AREA: MEDICAL [X] OTHER ____

SHORT TITLE: ACF - Automated Census

DATE LAST MODIFIED: 18 December 76

PURPOSE (up to 150 words; underline key words; continue on back if necessary):
The ACF Package allows admission of patients to the hospital data base, bed availability lists, ward reports, patient transfers, and appointment scheduling. The package is designed to use up to three CRT terminals for admitting and two hard copy printers for reports.

SOURCE

AUTHOR (S): Mr. James Watkins

CONTACT

| | |
|---|---|
| NAME: | Mr. Bob Anderson |
| BUSINESS ADDRESS: | Automata Incorporated<br>300 South Main Street<br>Washington, D.C. 20256 |
| TELEPHONE NO.: | (202) 555-1212 |

REQUIREMENTS

DIALECT

STANDARD [ ] OTHER ____ SUPPLIER ABC Corporation

MACHINE:

PERIPHERALS

A/D [ ] CRT [X] DECTAPE [ ] DISC [ ] MAGTAPE [X] PRINTER [ ] OTHER Tape Cassette

PARTITION SIZE (characters): 3600 characters

GLOBAL SIZE (number of blocks and block size): 10 Blocks/patient (64 word blocks)

RESTRICTIONS (continue on back if necessary): Routines have been coded specifically for the Quadtronix CRT with tape cassette. A modification of approximately five routines will be required for users not having these devices.

SIZE

NUMBER OF CHARACTERS OF CODE: 30,000

NUMBER OF ROUTINES: 20

AVAILABLE ITEMS

| | |
|---|---|
| COMMERCIAL SERVICE [ ] | SOURCE LISTING [X] $10.00 |
| DOCUMENTATION [X] $50.00 | USERS' MANUAL [X] No charge |

Return to: MUMPS Users' Group, 700 South Euclid Avenue, St. Louis, MO 63110

Last Date Modified

Give the date of last modification of the routine, program, or package.

Purpose

State exactly what the routine, program, or package functionally accomplishes. Features and attributes should be identified in terms of the application's end purpose. Key word descriptions should be underlined.

Author(s)

State the name(s) of the author(s).

Contact

Enter the name, address, and telephone number of the person responsible for disseminating information about the routine, program, or package. Any interested party may communicate directly with the contact in order to obtain additional details on the MUMPS application.

Dialect

Check the box if the application is written in the MUMPS Standard; otherwise, write the name of the dialect beside "OTHER". In either case, give the supplier of the dialect.

Machine

Enter the machine used and the manufacturer.

Peripherals

The peripherals required specifically by the routine, program, or package should be checked. If "OTHER" is checked, write in the peripherals needed.

Partition Size

Give the partition size (in characters) that the MUMPS routine, program, or package currently runs in. If dynamic partition sizing is used, give the size of the largest and smallest partition required.

Global Size

If globals are used, give the global block size and the number of blocks. In some applications, global size varies directly with the number of patients, items, etc., in the data base. If this is the case, indicate global size as the number of blocks required for a given number of items, or as the blocks required per single item.

## Restrictions

Anything that might restrict the use of the routine, program, or package should be listed here. Some possible restrictions are:

(1) The routine, program, or package may have a data base size limitation. For example, response time in the system slows significantly when the data base exceeds a certain size.

(2) Parts of the MUMPS code may be terminal-dependent, if the code is written to take advantage of some of the more advanced or "intelligent" terminals.

(3) The MUMPS code may require the printing of a clinical disclaimer prior to execution.

(4) The MUMPS code may be for proprietary use only.

## Number of Characters of Code

If the exact number of characters of MUMPS code is unknown, give an estimate.

## Number of Routines

Give the number of MUMPS routines which comprise the program or package.

## Source Listing

Check the box if a source listing is available and indicate whether or not there is a charge for the listing.

## Commercial Service

Check the box if the MUMPS routine, program, or package is being offered as a commercial service. If possible, indicate charges for the service.

## Documentation

Check the box if documentation is available and indicate if there are any charges for the documentation.

## User's Manual

Check the box if a User's Manual is available and indicate if there is any charge for the manual.

# Chapter 2

# APPROACHING CODE DOCUMENTATION

Good documentation is essential if a MUMPS installation wishes to properly maintain the routines, programs, and packages which have been developed. MUMPS source code documentation should provide enough information for a competent programmer, other than the original programmer, to modify the MUMPS code. The documentation should impart a clear description of the code.

The initial decision a programmer should make when beginning the documentation task concerns the approach. There are basically two ways of approaching MUMPS documentation: the physical structure approach and the logical structure approach. A description of each follows.

## 2.1 Physical Structure Approach

This approach can be considered a "bottom up" approach in that the programmer begins at the routine level and relates physical routines to larger functions within the package. After each routine is documented, the documentation for closed sets of related routines is combined to form program documentation. Program documentation is then used to form package documentation.

## 2.2 Logical Structure Approach

This approach may be considered a "top down" approach in that the programmer attempts to relate logical functions within the package to routines. In either approach, a package is considered to be a collection of programs which function together in an application. However, for the logical structure approach the program concept must be altered slightly as follows:

Program - the collection of MUMPS code needed to perform a logical task or function.

With this definition, the logical task performed determines the physical collection of code, which may or may not be a closed set of routines.

## 2.3 Discussion

The approach chosen will probably depend upon a given programmer's style and the nature of the application. The important point to remember is that regardless of the documentation approach, a routine is considered to be a collection of code filed and invoked as a single unit. Also, documentation for a routine is considered to exist as a single unit. For example, if a routine was composed of five logical sections and each section was used in a different program, the documentation for all five sections would exist as a single unit (just as the code does), even though the sections may be unrelated as to logical function. However, the documentation of each section should reference the program in which the section is used. In summary, the three phases of documentation are:

(1) Routine documentation - Specifies the logic of a single filed unit of code.

(2) Program documentation - Specifies the interrelationships existing within either a closed set of routines or a collection of code performing a specific logical task.

(3) Package documentation - Specifies the interrelationships existing within a collection of programs.

The documentation techniques presented in the chapters which follow are guidelines and tools for MUMPS documentation. These guidelines should not be considered rigid documentation constraints. For example, error return logic and interface variables might be most clearly depicted in the Package Flow Chart instead of the Package Interface Specifications. While the documenter may use his discretion as to the form his documentation may take, it is essential that all required documentation items be included. These required items are a description of the code logic and entry points, local variables, interface specifications, and files. The MUMPS Abstract is also a required documentation item.

## 2.4 Variable Concepts

In order to be consistent with the three levels of documentation (routine, program, and package), this manual also addresses variables at these same three levels as follows:

(1) Package Variable - Variables which are common throughout an entire package or retain a single usage throughout the package.

(2) Program Variable - Variables which are not package variables and are common throughout a program or retain a single usage throughout a program.

(3) Routine Variable - Variables appearing in a routine which are not considered package or program variables.

By addressing variables, variable classes, and conventions at these three levels we eliminate repetitious documentation, since each variable need be documented only once. As an example of a convention, all variables with single letter names could be declared as scratch variables at the package level. As another example, the variable NAM representing the patient's name may appear in hundreds of routines within a package. Since NAM would be considered a package variable, it is only documented once at the package level.

## 2.5 Automated Documentation Aids

Many of the documentation items recommended in Chapters 3, 4, and 5 can be generated either entirely or in part by automated documentation aids. A variety of these documentation programs have been developed and are available from the developing institution. Some of the existing MUMPS automated documentation programs are available from the MUMPS Users' Group (700 South Euclid Avenue, St. Louis, Mo. 63110). The use of automated documentation aids is recommended, whenever possible.

# Chapter 3

# PACKAGE AND PROGRAM DOCUMENTATION

Package and program documentation can be regarded as the task of documenting relationships and interfaces existing between collections of MUMPS source code. As mentioned previously, program and package documentation may take either a logical or physical structure approach. Thus, the collection of source code making up a program may be viewed as either a closed set of routines or the collection of code necessary for performing a given logical task. The primary documentation aid for packages and programs is the Package/Program Face Sheet. The face sheet is oriented toward a physical structure approach; however, the sections which follow explain modifications needed for a logical structure approach to documentation.

## 3.1 MUMPS Package/Program Face Sheet

A completed MUMPS Package/Program Face Sheet is illustrated in the sample of program documentation given in this chapter. The face sheet is divided into three parts:

(1) Common Items - Those items of information which are common to both programs and packages.

(2) Program Items - Program-specific information.

(3) Package Items - Package-specific information.

When using the face sheet, the Common Items section is always completed, and then either the Program Items or Package Items section (whichever is applicable) is completed.

The Common Items section of the Package/Program Face Sheet repeat many of the items requested by the MUMPS Abstract. Therefore, the reader should refer to Chapter 1 for instructions on completing the following items:

(1) Contact's name, address, and telephone number
(2) Dialect
(3) Machine
(4) Peripherals
(5) Partition size

Also, since the above items are likely to remain fixed throughout a package, they need to be completed only once on the package face sheet and should not be repeated on the program face sheets for programs which are components of the package. Instructions for completing the remaining common items are as follows:

### Package/Program Name

Enter the name associated with the package or program and mark out either the word 'package' or the word 'program' to indicate which one the face sheet is addressing. Also, on the same line enter the date of the package or program.

Entry Routine(s)

Give names of routines which initiate execution of the package or program.

Purpose

Give a brief statement of the overall package's or program's end purpose and describe its function.

List of Globals Used

Indicate the globals utilized by the package or program.

Global Documentation Appended

Indicate whether or not global documentation is included.

Variable List Appended

Indicate whether or not this document is appended. For packages, the variable list would list the variable name and give a brief description for the set of variables that are common throughout the package. For programs, the list would address itself to variables common throughout a program. Package level variables should not be repeated on the program variable list. Variables need only be listed and described once throughout the entire set of documentation. It is desirable to have the variable list in alphabetical order.

Test Run Data Appended

Indicate if test run data has been included in the documentation.

Instructions for completing the Package Items section are as follows:

Total Number of Programs

Give the total number of programs comprising the package.

Package Flow Chart and Package Interface Specifications

Indicate whether or not these documents are appended.

Instructions for completing the Program Items section are as follows:

Total Number of Routines

Give the total number of routines comprising the program.

Program Flow Chart and Program Interface Specifications

Indicate whether or not these documents are appended.

A detailed description of Flow Charts and Interface Specifications will be given when discussing program documentation and package documentation. It is recommended that they be included when documenting a program or package.

## 3.2 Package Documentation

A package is defined as a collection of programs which function together in an application. Package documentation involves combining the documentation of each program component, and with the aid of the MUMPS Package/Program Face Sheet and appended documents, describing the logical structure and interfaces between the components. Thus, the primary objective of package documentation is to give a clear description of all details regarding the interfaces and relationships existing between the program components of the package. The physical versus logical structure approach to documentation does not affect package documentation to any great extent. The task is to document interfaces between programs, regardless of whether programs are considered to be closed sets of routines or collections of code which perform a given task. The primary documents needed for describing these interprogram relationships are the Package Flow Chart and the Package Interface Specifications. A description of these two documents follows.

### Package Flow Chart

The logical structure of the package is traced in terms of the program components. The flow chart should depict the functional relationship of each program to the overall purpose of the package and also the relationships existing between programs.

### Package Interface Specifications

The Package Interface Specifications should give a detailed description of the interprogram relationships which exist during execution of the package. This could be thought of as a narrative description of the Package Flow Chart. Items (listed in order of importance) which might be included in the description are:

(1) A statement of each program function as it relates to the overall function of the package.

(2) A cross-reference of variables for the entire package, giving for each package level variable the routine names in which the variable is used. Note that the variable cross-reference is included in addition to the package level variable list which functionally describes each package level variable. If desirable, the two may be combined into one document.

(3) For each program, a narrative description of its relationship to other programs within the package.

(4) Program execution sequence.

(5) Error conditions, error return logic, and restart conditions for each program.

(6) Program interface variables and their description. Those variables linking one or more programs.

(7) A cross-reference of routines for the entire package, giving for each routine the routines which it invokes and those which it is invoked from.

## 3.3 <u>Ordering Package Documentation</u>

The recommended order for the components of package documentation is:

(1) Package/Program Face Sheet
(2) Package Flow Chart
(3) Package Level Variable List
(4) Package Interface Specifications
(5) Test Run Data
(6) Global Documentation
(7) Program Documentation for each program within the package.

## 3.4 <u>Program Documentation - Physical Structure Approach</u>

When utilizing the physical structure approach to documentation, a program is defined as a closed set of routines used to perform a specific task. In order to document a program, each routine within the program should be documented utilizing the MUMPS Routine Documentation form as described in Chapter 4 and illustrated in Appendix II. The routine documentation is combined to form program documentation with the aid of the MUMPS Package/Program Face Sheet.

Since the program's components (routines) have been documented individually, the task of program documentation is to specify in detail the relationships existing between the routines and to clarify the role of each routine as it relates to the overall purpose of the program. The primary documents needed for describing these interprogram relationships are the Program Flow Chart and the Program Interface Specifications. A detailed description of each of these documents follows.

## Program Flow Chart

The Program Flow Chart should trace the logical structure of the program in terms of the routines comprising the program. The function of each routine should be depicted as it relates to the overall purpose of the program. Routine interrelationship should be clearly illustrated.

## Program Interface Specifications

The Program Interface Specifications should describe in detail those routine interactions which take place when executing the program. Items (listed in order of importance) which might be included in the description are:

(1) For each routine, give the routine name and a short statement of the routine function as it relates to overall purpose of the program. If a naming convention is being used to relate routines within a program, the convention should be explained.

(2) A variables cross-reference for the entire program, giving for each program level variable the routine names in which the variable is used. As in package documentation, the program variables cross-reference should be provided along with the program level variable list, or the two documents may be combined as one document.

(3) For each routine (when applicable), a narrative description of its relationship with any of the other routines within the program.

(4) Calling sequence of the routines.

(5) Error conditions and error return logic between routines.

(6) Interface variables and their meanings. Those variables which are passed between routines and are necessary for linking two or more routines.

(7) A cross-reference of routines for the entire program, giving for each routine the routines which it invokes and those from which it is invoked. This cross-reference may be excluded if a similar package cross-reference is included in the documentation.

The Interface Specifications may serve as a detailed description of the Program Flow Chart.

## 3.5 Program Documentation - Logical Structure Approach

For the logical structure approach, a program is considered to be the collection of MUMPS code necessary to perform a specific logical task or function. The documentation task is to specify the relationships and interfaces existing within this collection of code. The physical location of each subunit of code making up the program should be identified. For example, program XYZ is made up by a subunit of code from routine A, two subunits from routine B, and a single line from routine C. Note that an entire routine commonly represents a subunit of code within a program. Following this identification of subunits, interfaces within the program are documented with the Program Interface Specifications and the Program Flow Chart. It should be noted that the total number of routines may not be applicable for a logical structure approach and, if so, need not be completed. A detailed description of the interface specifications and flow chart follows.

### Program Flow Chart

The flow chart should provide a picture of the logical structure of the program. Each smaller task within the program should be depicted as it relates to the overall purpose of the program. If possible, the physical location of each subunit of code should be included on the flow chart.

### Program Interface Specifications

All interactions taking place during execution of the program should be explained. Items (listed in order of importance) which might be included in the description are:

(1) Any smaller subunit of code existing within the overall collection of code making up the program should be identified (physical location) and a short statement given concerning its relation to the overall task of the program.

(2) A list and cross-reference of all program level variables. See Program Interface Specifications for a physical structure approach for details.

(3) A description of the execution sequence of the source code making up the program.

(4) Error conditions and error return logic within the program.

(5) Interface variables and their meaning (those variables which are passed within the program and are necessary for linkage of the source code making up the program).

(6) A cross-reference listing for routines.

## 3.6 Ordering Program Documentation

The recommended order for the components of program documentation is:

(1) Program/Package Face Sheet
(2) Program Flow Chart
(3) Program Level Variable List
(4) Program Interface Specifications
(5) Test Run Data
(6) Global Documentation
(7) Routine Documentation for each routine within the program.

A sample of program documentation is provided in the remaining section of this chapter.

Example of Completed Package/Program Face Sheet

17 Sept. 1975

# MUMPS PACKAGE/PROGRAM FACE SHEET

**COMMON ITEMS**

~~PACKAGE~~/PROGRAM (delete one) NAME: Medication Order Program DATE: 18 December 76

ENTRY ROUTINE (S):

PURPOSE (continue on separate sheet if necessary): The medication order program is that part of the Pharmacy Package which reads in the data for each prescription, checks for interactions, files the data in the patient record, and prints the label. The program checks each prescription number to make sure it is unused. Also, the abbreviated instructions (SIG) are expanded to patient readable form for the label (e.g., TID converted to "Take 1 tablet by mouth 3 times a day"). Appropriate messages are displayed whenever a drug interaction is detected.

CONTACT'S NAME, BUSINESS ADDRESS AND TELEPHONE NUMBER:

John H. Doe
3457 State Street
Rockville, Maryland 20122
301 555-1212

DIALECT

STANDARD [X] OTHER ____________

MACHINE: PDP-15

PERIPHERALS

A/D [ ] CRT [X] DECTAPE [ ] DISC [ ] MAGTAPE [ ] PRINTER [X] OTHER ____________

PARTITION SIZE USED: 1200 words/3600 characters

LIST OF GLOBALS USED: ^PHA, ^PHB

GLOBAL DOCUMENTATION APPENDED [ ]

VARIABLE LIST APPENDED [X]

TEST RUN DATA APPENDED [X]

**PACKAGE ITEMS**

TOTAL NUMBER OF PROGRAMS:

PACKAGE FLOW CHART APPENDED [ ]

PACKAGE INTERFACE SPECIFICATIONS APPENDED [ ]

**PROGRAM ITEMS**

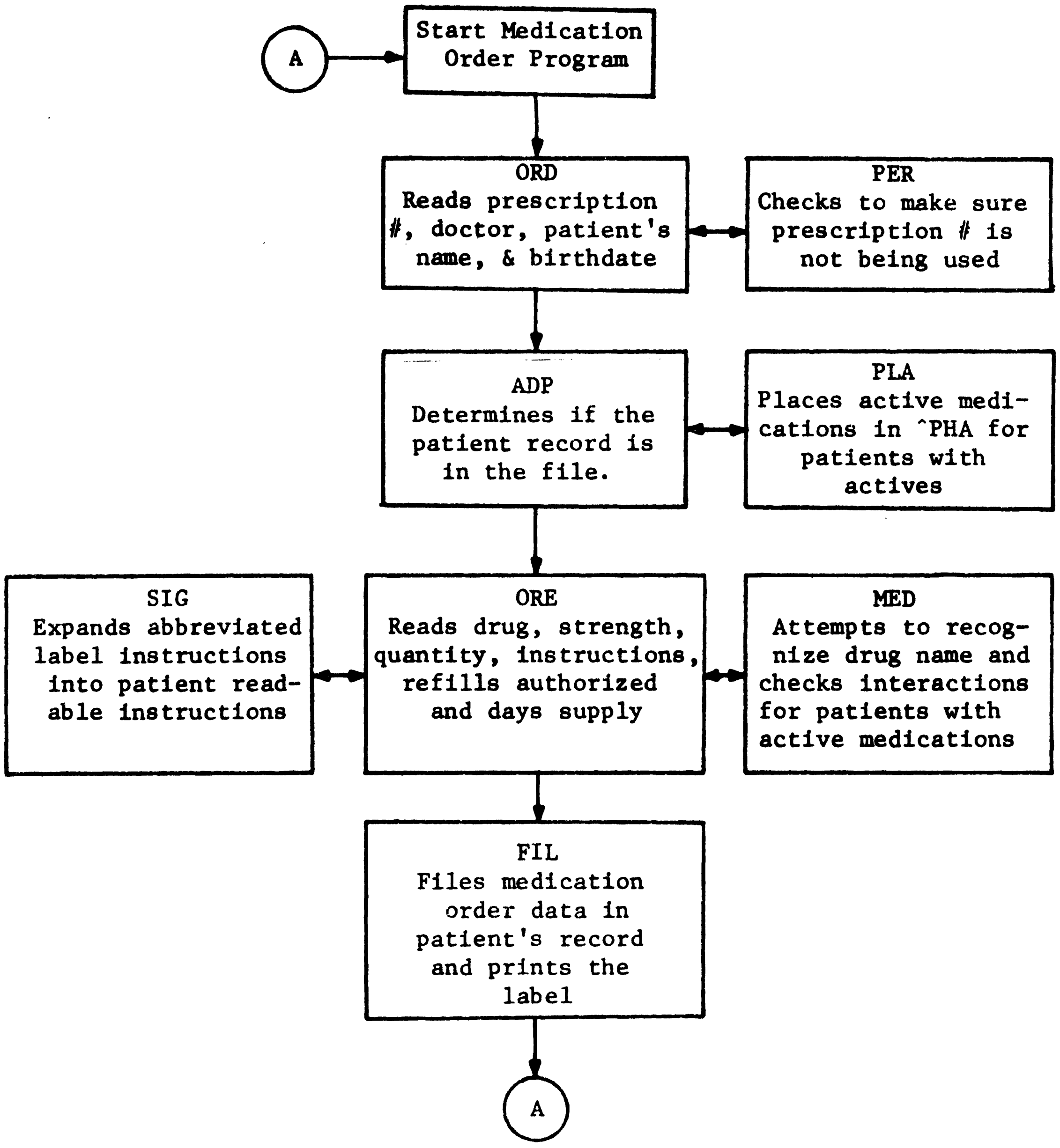
TOTAL NUMBER OF ROUTINES: 8

PROGRAM FLOW CHART APPENDED [X]

PROGRAM INTERFACE SPECIFICATIONS APPENDED [X]

Package: A collection of programs which function together in an application.
Program: A closed set of routines used to perform a specific task.

PROGRAM FLOW CHART


A
Start Medication Order Program
ORD
Reads prescription #, doctor, patient's name, & birthdate
PER
Checks to make sure prescription # is not being used
ADP
Determines if the patient record is in the file.
PLA
Places active medications in ^PHA for patients with actives
SIG
Expands abbreviated label instructions into patient readable instructions
ORE
Reads drug, strength, quantity, instructions, refills authorized and days supply
MED
Attempts to recognize drug name and checks interactions for patients with active medications
FIL
Files medication order data in patient's record and prints the label
A

## PROGRAM LEVEL VARIABLE LIST

PER - Prescription number (6 characters)  
DOC - Doctor name (10 characters)  
NNM - Patient name (25 characters)  
BD - Patient birthdate (6 characters)  
DRG - Drug name (30 characters)  
STR - Strength of drug (8 characters)  
SIG - Label instructions (30 characters)  
QTY - Quantity of drug dispensed (5 characters)  
REF - Number of refills authorized (4 characters)  
SUP - Number of days supply (4 characters)

## Program Interface Specifications

### Routine ORD

This routine initiates execution of the Medication Order Program. ORD reads the prescription number (PER), doctor (DOC), patient's name (NNM), and birthdate (BD) from the principal device and checks the syntax of each entry. The numeric variables R and M are computed from the name and birthdate. Interfaces with other routines are as follows:

(1) Routine PER - Routine ORD invokes PER to determine if the prescription number (PER) is unused. PER returns one of the following conditions:

   (a) Variable ERR=0 - Prescription is OK and variables PA and PB are returned. Variables PA and PB provide a mechanism for locating the prescription in the active prescription file ^PER(PA,PB).

   (b) Variable ERR=1 - Error condition has resulted; the prescription number is in use.

(2) Routine ADP - Upon completion of routine ORD, a GOTO is made to routine ADP. Variables NNM, BD, PER, DOC, R, M, PA, and PB are passed to ADP via local storage.

### Routine ADP

This routine determines if medications are being ordered for a new patient or for one who is already in the data base. ADP determines the value of the numeric variable X which gives the exact location for storing medication orders for this patient (^PHA(R,M,X)). Interfaces with other routines are as follows:

(1) Routine PLA - Routine ADP invokes PLA if the patient record resides in the data base. PLA places the patient's active medications in global ^PHD. No variables are returned to ADP by PLA.

(2) Routine ORE - Upon completion of ADP a GOTO is made to routine ORE. Variables NNM, BD, PER, DOC, R, M, X, PA, and PB are local variables passed to ORE.

### Routine ORE

This routine completes the data entry for the medication order. ORE reads the drug name or code (DRG), strength (STR), abbreviated label instructions (SIG), drug quantity (QTY), refills authorized (REF), and number of days supply (SUP) from the principal device. The syntax of each entry is checked. Interfaces with other routines are as follows:

(1) Routine MED - Routine ORE invokes MED to determine if the drug name or code (DRG) can be recognized. If the drug name is recognized, it is checked for interactions whenever the patient has other active medications. If an interaction occurs, appropriate messages are displayed on the principle device. MED returns one of the following conditions:

   (a) Variable ERR=0 - The drug name has been recognized and the Medicaid limit (ML), drug group (GRP), and drug name (DRG) are returned.

   (b) Variable ERR=1 - An error condition has resulted. The drug name or code could not be recognized.

(2) Routine SIG - Routine ORE invokes SIG in order to expand the abbreviated label instructions (SIG) into patient-readable instructions. SIG returns one of the following conditions:

   (a) Variable ERR=0 - Variable SIG has been successfully expanded and the instructions are returned in variable INS.

   (b) Variable ERR=1 - Variable SIG could not be interpreted.

(3) Routine FIL - Upon completion of ORE a GOTO is made to FIL. Local variables NNM, BD, PER, DOC, R, M, X, PA, PB, DRG, GRP, ML, STR, SIG, INS, QTY, REF, and SUP are passed to FIL.

## Routine FIL

This routine completes the medication order. All data thus far accumulated in local variables are stored in the patient record (^PHA). FIL then prints the label on a remote printer and kills all local variables. Interfaces with other routines are as follows:

(1) Routine ORD - Routine FIL transfers control to routine ORD in order to restart the medication order program.

## Routine PER

This routine is invoked by ORD to verify that the prescription number (PER) is unused. Input to the routine is variable PER. See the discussion of ORD for the variables returned by PER.

## Routine PLA

Invoked by ADP to place active medications in ^PHD. Input variables required are R, M, and X. See the discussion of routine ADP for additional details.

## Routine MED

Invoked by ORE to check the drug name or code. Input to this routine is the variable DRG. MED also checks for interactions if the patient has active medications. See the discussion of routine ORE for the variables returned by MED.

Test Run For Medication Order Program*

```
PHARMACY TRANSACTION: ORDER MEDICATIONS


          <<< MEDICATION ORDERS   03/24/74 >>>

PER DOC: 102030 LASH
NAM BDT: SMITT,JOHN,H  6/13/47
DRG STR: ASA  5GR   ASPIRIN
    SIG: Q4H PRN HEADACHE

QTY,REF,SUP:  100,3,30
NEW PATIENT   MED OR CCS: C

ORDER OK (Y OR N).. Y
--------------------------
| 102030  DR LASH        |
|SMITT,JOHN,H 06/13/47   |
|TAKE 1 TABLET BY MOUTH  |
|EVERY 4 HOURS AS NEEDED |
|FOR HEADACHE            |
|ASPIRIN 5GR             |
|03/25/74 #100           |
--------------------------
LABEL OK ?... Y

PER DOC: 102031  LASH
NAM BDT: SMITT,JOHN,H  06/13/47
DRG STR: DARVON   OK

  ** WARNING ** DRUGS ARE IN THE SAME GROUP - GROUP 28:8
DARVON AND ASPIRIN


     SIG: 2 CAP HS

QTY,REF,SUP: 50,2,10
1 ACTIVE FOR SMITT,JOHN,H

ORDER OK (Y OR N)..Y
--------------------------
| 102031  DR LASH        |
|SMITT,JOHN,H 06/13/47   |
|TAKE 2 CAPSULES BY      |
|MOUTH AT BEDTIME        |
|DARVON                  |
|03/25/74 #50            |
--------------------------
LABEL OK ?.. Y
```

*user input is underlined

## Routine Documentation for Medication Order Program

At this point in the documentation of the Medication Order Program the complete documentation for routines ORD, PER, ADP, PLA, ORE, MED, SIG, and FIL should be appended. The documentation is not included here since routine documentation is illustrated in Chapter 4.

# Chapter 4

## ROUTINE DOCUMENTATION

Routine documentation provides a detailed explanation of the MUMPS source code. Error conditions or source code modifications will necessitate a need for a detailed description of the routine logic, variable specifications, I/O, etc., all of which are provided by the routine documentation. It should be noted that routine documentation changes very little with the documentation approach. For the physical structure approach, the name of the program to which the routine belongs should be given. For the logical structure approach, each logical unit of the routine utilized by a program should be clearly identified and the program's name given.

To document a routine the MUMPS Routine Documentation form is utilized. A completed sample form is illustrated in the example of routine documentation given in this chapter and a blank form is provided in Appendix II.

The form is divided into two parts:

(1) Items which may be entered on the form,
(2) Items which may be checked to indicate that this information is provided on attached sheets.

Documentation for any given routine may not include all items on the form; however, a well documented routine would include most of them.

### 4.1 Instructions for the MUMPS Routine Documentation Form

Instructions follow for each section and the items within each section of the MUMPS Routine Documentation form.

#### Routine Name

Indicate the MUMPS name of the routine, that is the name used to invoke the routine. The name may be supplemented by a one-line description of the routine.

#### Date

Give the date of the current version of the routine.

## Purpose

The Purpose section should state what the routine functionally accomplishes. If the routine is part of a program or package, this section should be stated so as to relate it to the overall purpose of the program or package. The Purpose section normally requires considerably less than one page; it may be continued on the back side of the MUMPS Routine Documentation form if necessary.

## Size

Give the number of characters of code in the routine.

## Multiple Entry Points

The entry point feature of MUMPS allows a routine to be invoked and to begin execution at any line of code in the routine. This capability is similar to the entry feature of FORTRAN, and is frequently used to combine several similar procedures into one routine.

The first line of code in a routine is an assumed entry point; therefore, only those entry points other than the first line should be documented. The documentation for each entry point should include the line label, a brief statement as to the purpose of the entry point, and the requirements of the entry point (e.g., certain variables must be defined). When several entry points within a routine have identical requirements, the requirements should only be documented one time. Entry points should be documented in order of their appearance in the routine.

## Local Variables

The name and purpose of each routine level local variable may be given for local variable documentation. For arrays, indicate the name by following the character name of the array with "()". For example, array A(1),A(2)...A(100) may be represented as "A()". Routine level variables are those variables appearing in the routine which are not considered package or program level variables, and therefore have not been described elsewhere in the documentation. Whenever variable conventions are used (e.g., if all single character variable names are routine level scratch variables), then a statement of the convention should be given instead of repeatedly listing the set of variables for each routine. It should be noted that all variables in a routine may be package or program level variables, and therefore the routine may not contain any routine level variables. A cross-reference listing giving the source code position of the use of each variable may be included in local variable documentation.

### Globals

The names of all globals referenced by the routine may be given. For each global, the source code location of each global reference may also be included.

### Detailed Description

A detailed description of routine logic may be included. This description should be sufficiently detailed to allow another MUMPS programmer to understand and modify the routine easily. It should include a listing of the routine along with a brief narrative description of each logical block or section within the routine without going to the extreme of describing each line of code. The programmer should give special mention to potential problem areas or tricks in the code (e.g., indirection). An annotated flow chart of the routine logic may also be included.

## 4.2 Internal Documentation

Internal documentation refers to all explanatory notes and text which are stored along with the routine as nonexecuting lines of source code. This type of text is commonly referred to as a comment line and is indicated by a preceding semicolon. The quantity of comment text within any given routine varies with the nature of the application, the abundance of storage space, installation policies, etc. In general, extensive line-by-line comments within a routine are not recommended. However, some comment text is justified, especially in the case of routine header information and for explaining especially obscure pieces of code.

### Routine Header Information

It is recommended that the routine name, the name of the program or package, the programmer's initials, and the date of last modification be stored in the routine as comment text (preferably the first line of the routine). A short statement concerning the purpose of the routine is also helpful if space is available. The particular format of the header line may be determined by the programmer or the installation. However, it is important that the format of the text remain consistent for all routines within a program or package.

### Highlighting of Tricks and Gimmicks

Any special coding tricks or gimmicks which might be difficult for another programmer to understand should be commented upon. No attempt should be made to provide a complete explanation. All that is needed is a brief note indicating the presence of a "tricky" section of code and possibly a reference to the external routine documentation where a complete explanation of the code may be found. It is especially important for transportability to provide comment text for any special gimmicks which are implementation-dependent.

## 4.3 Ordering MUMPS Routine Documentation

The suggested order for the elements comprising MUMPS routine documentation is as follows:

(1) The MUMPS Routine Documentation form
(2) Detailed description of the routine
(3) Multiple entry points
(4) Local variables
(5) Globals

# MUMPS ROUTINE DOCUMENTATION

**ENTER THESE ITEMS ON THIS SHEET**

*ROUTINE NAME: GED

DATE: 15 July 1976

PURPOSE:
Allows general text editing functions on global data. A replacement type editor is used to modify data within a global node. A particular global node may be killed or replaced with entirely new data. Data may also be transferred from one global node to another.

SIZE (number of characters in routine): 829 Characters

**CHECK WHICH OF THESE ITEMS ARE PROVIDED ON ATTACHED SHEETS**

MULTIPLE ENTRY POINTS

- [ ] Line Label
- [ ] Purpose
- [ ] Requirements

LOCAL VARIABLES

- [X] Name
- [X] Purpose
- [X] Cross-Reference Listing

GLOBALS

- [X] Name
- [X] Location of Global References

DETAILED DESCRIPTION

- [X] Narrative Description
- [X] Listing
- [ ] Flow Chart

* A routine is defined as a collection of MUMPS statements filed, called, and/or overlayed as a single unit.

## LISTING

```
010 ; GED - GLOBAL EDIT AND TRANSFER
100 ;
    D 700 R !!,"GLOBAL NAME ^",NAM I NAM="" D 700 Q
    D ^GCK I ERR=1 G 100
120 R !!,"EDIT OR TRANSFER (E OR T)...",ANS I ANS="" G 100
    I ANS?1"T".E G 300
130 W !!,"GLOBAL NODE ^",NAM,"(" R NDE I NDE="" G 120
    D S8 I ERR=1 G 130
    D S1
    I AX=0 W "  UNDEFINED" G 180
    I AX=10 W "  POINTER - NO DATA" G 180
    D S2
    D 600
    I AX=""!(AY="") G 130
    D 155 G 130
155 D S3 S @NDE=AX
    Q
160 R !!,"KILL OLD LOCATION ?...",ANS I '(ANS?1"Y".E) Q
    S NDE=AX D S4 L
    Q
180 D S5 I AX="" G 130
    D 155 G 130
300 W !!,"OLD GLOBAL LOCATION: ^",NAM,"(" R NDE I NDE="" G 120
    D S8 I ERR=1 D S6 G 300
    D S1 I AX=0 W " UNDEFINED" G 300
    D S2 S AY=AX,AZ=NDE
325 W !,"NEW GLOBAL LOCATION: ^",NAM,"(" R NDE I NDE="" G 300
    D S8 I ERR=1 D S6 G 325
    D S1 I AX="0" W " UNDEFINED" G 325
    D 155,160 G 300
600 ;
    R !," R ",AY I AY="***" D S5 Q
    I AY="" Q
    I $F(AX,AY)=0 W " ? " G 600
    R " W ",AZ
    S AX=$E(AX,1,$F(AX,AY)-$L(AY)-1)_AZ_$E(AX,$F(AX,AY),255) Q
700 ;
    K ANS,AX,AY,AZ,ERR,NAM,NDE,A,I
    Q
800 ;
    S A=$P(NDE,",",I) D S7
    Q
 S1 S NDE="^"_NAM_"("_NDE_")",AX=$D(@NDE)
    Q
 S2 S AX=@NDE W !,AX
    Q
 S3 L @("^"_NAM)
    Q
 S4 D S3 K @NDE
    Q
 S5 R !,"NEW: ",AX I AX="KILL" D S4 S AX=""
    Q
 S6 W "  BAD FORMAT - HIT ALT" R ANS W *13,*30,*26,*26
    Q
 S7 I A?.N S ERR=0 Q
    S ERR=1
    Q
 S8 W ")" S ERR=0 F I=1:1 Q:$P(NDE,",",I)=""  D 800
    Q
```

## NARRATIVE DESCRIPTION

| Logical Section | Description |
|---|---|
| 100 to 120+1 | Reads in the global to be edited and select either edit or transfer. |
| 130 to 130+8 | Edit section. Reads global node, makes sure it is defined, and then edits. |
| 300 to 325+3 | Transfer section. Reads old global location, new global location, makes the transfer, and kills old location if desired. |
| 600 to 600+5 | Used by the edit section to replace or edit data within the string AX. |

## VARIABLE DESCRIPTION

| Variable Name | Purpose |
|---|---|
| NAM | The name of the global being edited. |
| ERR | Used as an error flag when routine GCK is invoked. If ERR=1 the global name is valid. Also used as an error return when checking syntax of variable NDE. |
| ANS | Scratch variable used as the response to a read. |
| NDE | The global node which is to be edited. NDE has the form ^NAM(NUM,NUM,...). |
| I | Scratch variable used as the increment in a FOR loop. |
| AX | Set equal to the data node to be edited or transferred. |
| AY | In transferring global data, takes the value of the data to be transferred (old location). Also used in the edit section as the data string to be replaced. |
| AZ | Takes on the value of the global node (old location) in the transfer section. Used as the data string to replace AY in the edit section. |
| A | Scratch variable used for checking syntax of variable NDE. |

Example of Routine Documentation

## VARIABLES CROSS-REFERENCE LISTING

| Variable Name | Line Label or Line Label + Offset | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| NAM | 100+1 | 130 | 300 | 325 | 700 | S1 | S3 | |
| ERR | 100+2 | 130+1 | 300+1 | 325+1 | 700 | S7 | S7+1 | S8 |
| ANS | 120 | 120+1 | 160 | 700 | S6 | | | |
| NDE | 130 | 155 | 160+1 | 300 | 300+3 | 325 | 700 | 800+1 |
| | S1 | S2 | S4 | S8 | | | | |
| I | 700 | 800+1 | S8 | | | | | |
| AX | 130+3 | 130+4 | 130+7 | 155 | 180 | 300+2 | 300+3 | 325+2 |
| | 600+3 | 600+5 | 700 | S1 | S2 | S5 | | |
| AY | 130+7 | 300+3 | 600+1 | 600+2 | 600+3 | 600+5 | 700 | |
| AZ | 160+1 | 300+3 | 600+4 | 600+5 | 700 | | | |
| A | 700 | 800+1 | S7 | | | | | |

## GLOBALS

Routine GED does not use any specific global, since the purpose of the routine is to edit and transfer data for any global. Global references are made at the following labels: S1, S2, S3, and S4.

# Chapter 5

## MUMPS FILE SUMMARY FORM

The MUMPS File Summary form, an example of which is reproduced on the next page, has been adopted by the MUMPS Development Committee. It is to be used to provide summary data regarding a global file and will typically accompany a global file documentation package. The following is a description of the items to be completed:

FILE NAME. Name of the file.

DATE. Date of documentation.

GLOBAL(S). Name of the global or globals used, indicating subscript range if not an entire global.

PURPOSE. Brief statement of use of the file.

SIZE. Characterization of the size of the file as maximum number of blocks and block size. If size is a function of a particular data item (e.g., patient) indicate the size relationship (number of blocks/patient).

NUMBER OF LEVELS. Maximum number of levels in the file.

LEVELS WITH DATA. List the levels which contain data.

PACKAGES AND PROGRAMS UTILIZING THIS FILE. Use of the file should be indicated in terms of individual routines, if its use is isolated; otherwise, by programs or packages, as a cross-reference to program documentation on the use of the file.

DOCUMENTATION PROVIDED. Indicate whether documentation of the logical structure of the file is provided, in narrative, tabular, and/or graphic form by checking the appropriate box. Also indicate whether documentation for the physical structure of the file is provided in narrative, tabular, and/or pictorial form.

ANNOTATED REPRESENTATIVE LISTING ATTACHED. Indicate whether an example printout, with annotations, of the global file's contents is provided.

RETENTION CHARACTERISTICS. Indicate frequency and nature of purges, dumps, etc.

COMMENTS. Describe any peculiarities or programming conventions with regard to the use of the file, e.g., interlock procedures for multi-user updating.

A sample blank MUMPS File Summary form is given in Appendix II.

17 Sept. 1975

# MUMPS FILE SUMMARY

FILE NAME: Patient File  DATE: 11 June 1976

GLOBAL(S): ^PAT

PURPOSE: Storage of data on hospital patients who have had laboratory procedures.

SIZE: 1250 blocks (256 16-bit words/block) for 500 inpatients and 200 outpatients.

NUMBER OF LEVELS: 4

LEVELS WITH DATA: Level 3 and level 4

PACKAGES AND PROGRAMS UTILIZING THIS FILE: Admissions and bed census (ADM), Clinical laboratory (LAB), and Patient billing (BIL)

DOCUMENTATION PROVIDED

| | Logical Structure | Physical Structure |
|---|---|---|
| Narrative | ☒ | ☒ |
| Tabular | ☒ | ☒ |
| Pictorial | ☒ | ☒ |

ANNOTATED REPRESENTATIVE LISTING ATTACHED ☐

RETENTION CHARACTERISTICS: Patient data are deleted by midnight by purging routine (PUR) of LAB program, after 10 days inactivity for inpatients, of 3 days after completion for outpatients.

COMMENTS: In-use flag IUF at third level used as software interlock by updating an entry for a specific patient.

The MUMPS File Summary form briefly describes a global file and indicates what further documentation on the file is available. A collection of such forms is necessary to fully describe a data base consisting of several files. A programmer faced with an unexplained error in a program referencing a specific global node, however, needs one futher documentation aid, the maintenance of which is recommended on an installation-wide basis. This additional documentation aid is a directory of the globals utilized in the instaliation, and the files contained in them. An example follows:

GLOBAL FILE DIRECTORY

| Global | Suoscripts | File |
|---|---|---|
| ^A | 0-99 | Patients |
| | 100-101 | Location Directory |
| | 200-226 | Alphabetic Patient Directory |
| ^B | 1 | Data Acquisition Control Buffer |
| | 2 | Data Acquisition Error Messages |
| | 4 | Stat/Critical Messages |
| ^C | 0-99 | Chemistry Laboratory Specimens |
| | 100 | Request mnemonic codes |
| | 101 | Test normal ranges |
| | 102 | Test long names |
| | 103 | Test worksheet assignments |
| | 104 | Test billing codes |
| | 105 | Test tally data |
| ^D | | Same as ^C for Hematology |
| ^E | | Same as ^C for Microbiology |
| ^K | | System parameters |
| ^M | | Physicians |
| `Z | | Scratch file |

# Chapter 6

# FILE DOCUMENTATION PROCEDURES

**File structure terms used in this chapter are defined in Appendix I.**

## 6.1 Logical File Structure Documentation

The process of describing a file in terms of its logical structure independent of implementation details should consist of the following elements:

(1) Identification of the hierarchical organization of repeating groups in the file.
(2) Specification of the access key(s) to each repeating group.
(3) Identification of the data elements in each repeating group.

This may be done through a combination of narrative, tabular description, and pictorial representation. Such documentation serves as a convenient logical and symbolic cross-reference to the more detailed global structure documentation. The following example describes the hierarchical organization of file PAT, which contains data on hospital patients who have had clinical laboratory test work done on them:

### An Example Patient Laboratory File - PAT

The highest level repeating group is the file PAT itself, entries of which are individual patients. Access to each patient entry is by the direct access key, ID, the patient identification number.

The entry for patient contains data relating to the patient's name, date of birth, sex, admission date, hospital location, and other items, and in addition contains a repeating group called LABSEC, entries of which contain data specific to a particular clinical laboratory section, such as chemistry, hematology, or microbiology.

Direct access to a LABSEC entry, an individual laboratory section, is by LBNO, the laboratory section number. In addition to a count of number of specimens on file in that laboratory section, the entry contains three repeating groups, CUMDIR, INTDIR, and SPECLST. CUMDIR entries are result categories for which specimens are on file. Each result category entry contains a repeating group called SPECIND, which is an index of specimen entry numbers for specimens on file with tests in that result category. Access to each CUMDIR entry is by result category number RC and access to each entry in its SPECIND repeating group is by its entry number. The entries in SPECIND are indexes to be used as keys to the SPECLST repeating group described below.

INTDIR is a repeating group with a structure identical to CUMDIR, but containing an "interim" directory to only those specimens in SPECLST which have tests that are either still pending or were ordered during the current day.

Entries in the SPECLST repeating group contain detailed data on individual specimens in the file, such as accession number, accession date and time, specimen type, "stat" or emergency status, and tests requested. The access to a specimen entry is by its sequential entry number, and since the file is updated chronologically, the entry number serves to order specimens by accession date and time. Each specimen entry contains a repeating group, RCDAT, in which results for tests requested are stored according to result category.

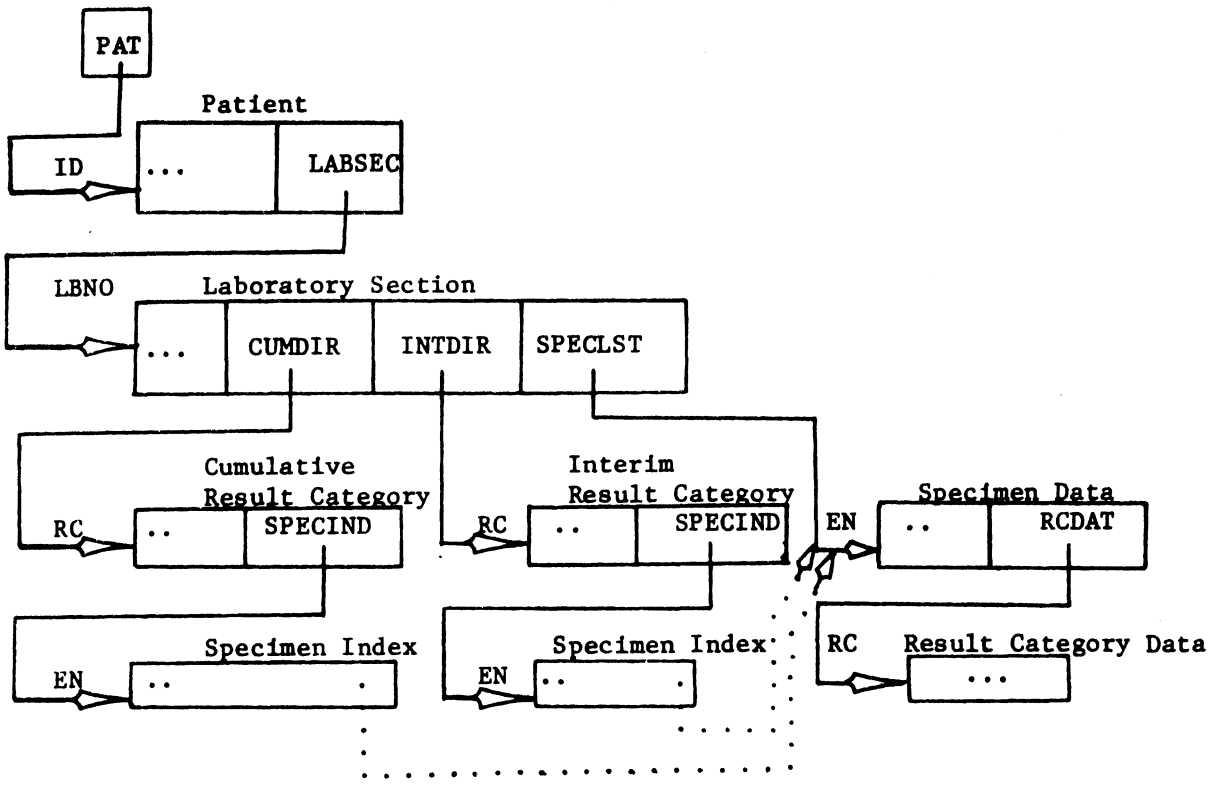
Each entry in RCDAT, accessed by the result category number RC, is a set of data values sequenced by this result category, giving the results on the tests in this category that were requested, or their status if not yet completed.

This description of the file is given more precisely in the following table, and schematically illustrated in the accompanying diagram.

## TABULAR DESCRIPTION

| Data Element | Meaning |
|---|---|
| PAT | Patient (Rep. Grp.)<br>file of patients having had laboratory work, direct access by ID:<br>--- |
| .IUF | In-use flag |
| . ID | Patient identification number |
| .NAM | Patient name |
| .DOB | Date of birth |
| .SEX | Sex |
| .ADM | Admission date |
| .LOC | Hospital location |
| .LABSEC | Laboratory Section (Rep. Grp.)<br>data on laboratory work by lab section; direct access by LBNO:<br>--- |
| ..LBNO | Laboratory section number |
| ..SPECTOT | Total number of specimens on file |
| ..CUMDIR | Cumulative Directory of Result Categories (Rep. Grp.)<br>Specimens on file for this laboratory section, organized by result categories for which they contain test requests; direct access by RC:<br>--- |
| ...RC | Result category number |
| ...SPECIND | Specimen Index (Rep. Grp.)<br>List of specimen indexes for specimens with tests in this result category; direct access by EN:<br>--- |
| ....EN | Sequential entry no. in this repeating group. Index to be used as access key to specimen data in SPECLST below |
| ..INTDIR | Interim Directory of Result Category (Rep. Grp.)<br>Today's or pending ("interim") specimen directory organized by result categories of tests requested; direct access by RC:<br>--- |
| .<br>.<br>. | ...same structure as CUMDIR above... |
| ..SPECLST | Specimen Data (Rep. Grp.)<br>Data on individual specimens on file for this lab section; direct access by EN:<br>--- |

| Data Element | Meaning |
|---|---|
| ...EN | Sequential entry no. in this repeating group |
| ...ACCNO | Accession no. of the day in this laboratory sec. |
| ...ACCDT | Accession date |
| ...ACCTM | Accession time |
| ...TYP | Specimen type |
| ...STAT | "Stat" or emergency status flag |
| ...REQ | List of tests requested |
| ...RCDAT | Result Category Data (Rep. Grp.)<br>Data on results of tests in a result category; direct access by RC:<br>--- |
| ....RC | Result category number |
| ....RES1 | Result for TEST1 of this result category |
| ....RES2 | Result for TEST2 of this result category |
| . | . |
| . | . |
| . | . |
| ....RESm | Result for TESTm of this result category --where no. of tests, m, is specific to result category |



Logical Structure of File PAT

The table above identifies the repeating groups in the file PAT, and the data elements of each are described and given mnemonic identifiers. All data in the file are considered directly accessible by specification of a sequence of data element identifiers and access key values. Logical depth in the hierarchy is indicated by the sequence of dots preceding each identifier. These dots correspond to the number of access keys required to reach the goal entry.

## 6.2 Global Structure Documentation

The documentation of a global file's data structure should include the following:

(1) Definition of the overall structure, in terms of the mapping of the repeating groups of the logical file into the structure, and the encoding of access key information.

(2) Detailed description of global structure, describing in full the nodes used at each level in the structure, the subscript computation algorithms used, the contents of the nodes, and how they relate to the logical structure.

(3) An annotated example printout of relevant portions of the global's actual contents.

(4) Miscellaneous comments on packing efficiency, accessing procedures, and other issues involved in determination of structure.

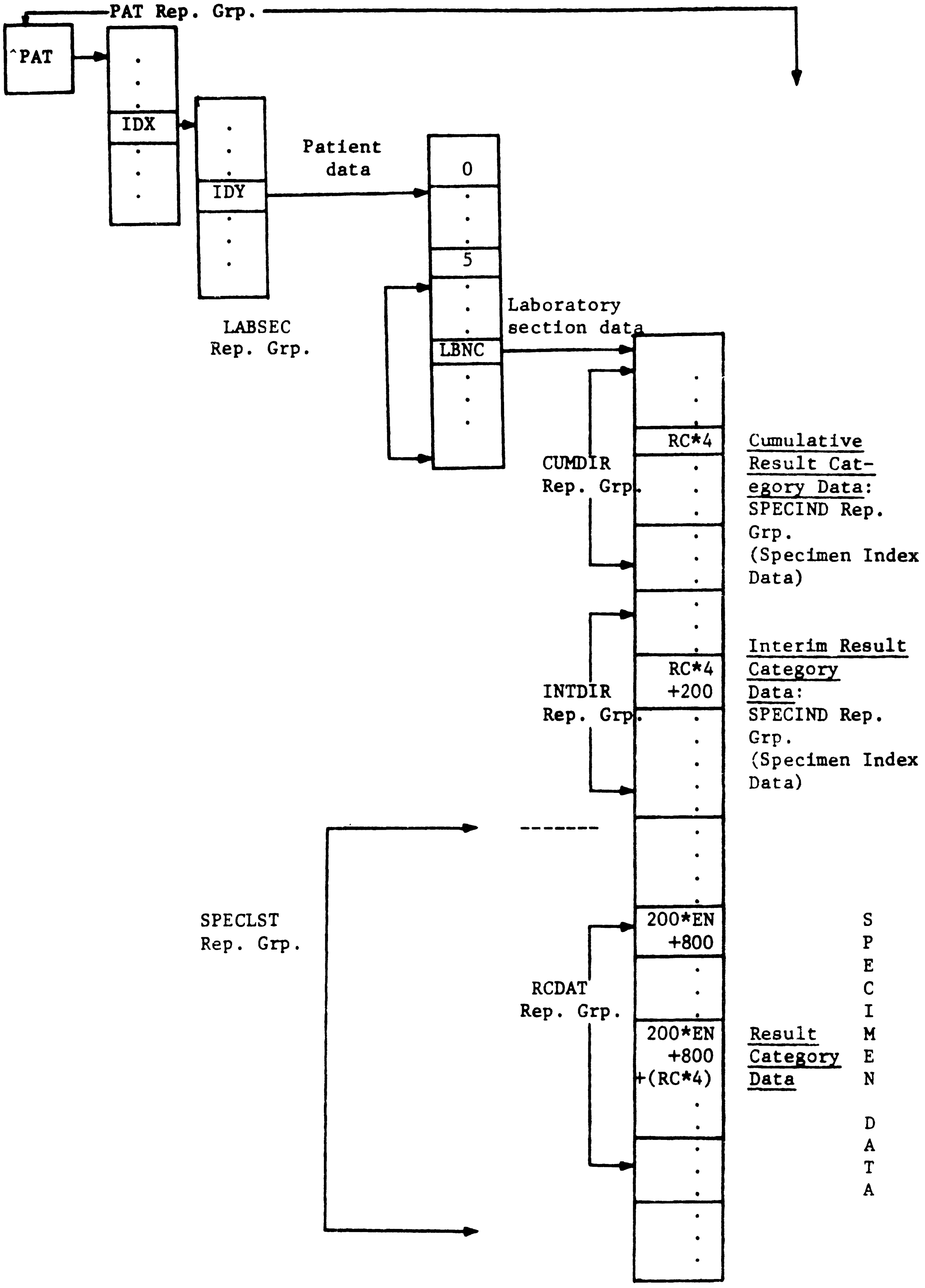
For items (1) and (2) above, a combination of pictorial, narrative and tabular description can be used. The detailed tabular description is the most essential, and should therefore be considered mandatory.

Global documentation will be illustrated with the global structure ^PAT for the logical file PAT discussed in the preceding section. Overall structure can be depicted most conveniently pictorially. The diagram on the next page shows the global structure for ^PAT, indicating the mapping of the various repeating groups of the logical structure into the global, and the mechanism of access to individual entries in the various repeating groups.

### Narrative Description

File PAT is mapped into global ^PAT. The first two levels of subscripting are both entirely pointers, providing direct access to individual entries in the patient repeating group, based on hospital identification number ID. The first level, using subscript IDX, computed as the terminal two digits of ID, divides the patient population into 100 approximately equal, randomly distributed groups, each containing only about 1% of the population. The leading four digits of ID are used as the second level subscript, IDY, to permit direct access to the patient data, stored at the third and fourth levels.

PICTORIAL REPRESENTATION



Global Structure of File PAT

Administrative and demographic data for a patient entry are stored at the third level in subscript 0-5. Data for each laboratory section, in the LABSEC repeating group, are stored primarily at the fourth level, and accessed by pointers at the third level, at subscripts indicating laboratory section number LBNO. Total number of specimens in each laboratory section, SPECTOT, is stored as a value at the third level at ^(LBNO).

The fourth level data for each laboratory section entry consists of three parts:

(1) CUMDIR, a cumulative directory of specimens on file, organized by result categories for which tests have been requested and stored at subscripts 0-199. An entry for a result category RC is accessed directly at ^(RC*4), with overflow data as necessary stored at ^(RC*4+1), etc. Each RC entry contains the SPECIND repeating group. This consists of indexes to specimen entries for this result category (primary data on which is stored in the SPECLST repeating group described below). The SPECIND repeating group is stored as a string (or strings if length is more than 72) of numerical values, representing specimen entry indexes, concatenated by commas.

(2) INTDIR, a directory of "interim" or current specimens organized by result category as for CUMDIR above. INTDIR is stored at subscripts 200-299, where its structure is analogous to CUMDIR. Each RC is stored at ^(RC*4+200) with overflow data as necessary at ^(RC*4+201), etc., and contains a SPECIND repeating group as above.

(3) SPECLST, the repeating group containing the primary data on each specimen in the laboratory section is stored at subscripts 1000 and above. Packed accession data for specimen entry EN is stored, concatenated by semicolons, at ^(200*EN+800).

A result for specimen EN in SPECLST is stored in the RCDAT repeating group, accessed by RC, the number of the result category entry to which it belongs. This repeating group is embedded at the fourth level as offsets to the specimen entry inself, in the range between ^(200*EN+804) and ^(200*EN+996). Each RC entry, stored at ^(200*EN+800+(RC*4)), contains results for requested tests, in preassigned positions, in a string of result fields concatenated by semicolons.

A detailed description of global ^PAT follows. In the logical structure, preceding dots before a data element identifier are used to indicate logical depth. In the global structure description, similarly, dots preceding a naked reference are used to indicate number of preceding subscripts implicit in the reference. Though other conventions for representing the level of a naked reference could be used, or a full reference could be stated instead, it appears that this gives readability to the documentation without cluttering it.

TABULAR DESCRIPTION

| Global | Logical | Access/Mapping/Meaning |
|---|---|---|
| | PAT | Patient (Rep. Grp.) Mapped as 3 level global, first 2 levels of which encode patient identification no. ID, as access key in the form of IDX, IDY, where: ID=(str) patient identification no. as 6N IDX=(num) terminal 2 digits of ID IDY=(num) leading 4 digits of ID |
| ^PAT(IDX) | | (ptr) to subgroup containing patients with terminal 2 digits of ID equal to IDX. Divides patient population into 100 subgroups based on possible values of IDX (0:1:99). |
| .^(IDY) | | (ptr) to individual patient having laboratory work on file |
| | | Distributing patients randomly into 100 groups by IDX minimizes search time for individual patient. IDX subscripts fit entirely in one 256 word block. Each instance of IDY level will fit in one block for up to 126 entries, making no. of blocks at IDX, IDY levels to be searched relatively insensitive to patient population size changes. |
| ..^(0) | .IUF | (num) in-use flag, where IUF=0 means patient entry free, IUF=n means in-use by process n. |
| ..^(1) | .ID | (str) patient identification no. in form 6N. |
| ..^(2) | .NAM | (str) patient name as last, first. |
| ..^(3) | | (str) SEX.";".DOB, where: |
| | .SEX | (str) Sex as "M", "F", or null if unknown. |
| | .DOB | (str) Date of birth as YYMMDD or null if unknown. |
| ..^(4) | .ADM | (dat) admission date. |
| ..^(5) | .LOC | (str) hospital location code as 1A2N. |
| | .LABSEC | Laboratory Section (Rep. Grp.) data on laboratory work by lab section. Mapped at nodes with subscripts 120, 130, 140 at current (3rd) level, accessed by LBNO: |

| Global | Logical | Access/Mapping/Meaning |
|---|---|---|
| | ..LBNO | (num) laboratory section number, where 120: chemistry, 130: hematology, 140: microbiology<br><br>Used in subscript calculation: |
| ..^(LBNO) | ..SPECTOT | (num) total number of specimens received for this laboratory section, and (ptr) to remainder of LABSEC, stored at 4th level. |
| | ..CUMDIR | Cumulative Directory of Result Categories (Rep. Grp.) Specimens on file for this laboratory section organized by result category of tests requested. Mapped at 4th level, in subscript range 0-199 accessed by subscript calculation involving RC: |
| | ...RC | (num) result category no. in range 1:1:49.<br>Used in subscript calculation: |
| ...^(RC*4) | ...SPECIND | (str) Specimen Index (Rep. Grp.) List of index values to be used as access keys to data on specimens having tests in this result category. Entries are subfields of this string concatenated by commas, accessed by entry no. EN. Form of the string is IND1, IND2,...,INDn. |
| | ...EN | (num) entry no. corresponding to subfield no. in above string value. |
| | ...IND | (str) index value in form 1NN, corresponding to entry no. EN. IND is stored in ENth subfield of string EN above. Value of IND is entry no. to SPECLST Rep. Grp. (see below), to be used as an access key to data on specimen having tests in this result category. |
| ...^(RC*4+1)<br>...^(RC*4+2)<br>.<br>.<br>.<br>(etc.) | | (str) same as ^(RC*4), defined as necessary for continuation of SPECIND, Specimen Index Rep. Grp., due to string overflow. |

| Global | Logical | Access/Mapping/Meaning |
|---|---|---|
| | ..INTDIR | Interim Directory of Result Categories (Rep. Grp.) Directory of today's or pending ("interim") specimens, by result category. Mapped at 4th level in subscript range 200-399, accessed by subscript calculation involving RC: |
| | ...RC | (num) result category no. in range 1:1:49. Used in subscript calculation: |
| ...^(RC*4+200) | ...SPECIND | Same as SPECIND in CUMDIR above. |
| ...^(RC*4+201) | | |
| ...^(RC*4+202) . . . (etc.) | | (str) same as ^(RC*4+200), defined as necessary for continuation of SPECIND, due to string overflow. |
| | ..SPECLST | Specimen Data (Rep. Grp.) Data on individual specimens on file for this laboratory section. Mapped to nodes at 4th level in subscript range 1000-32700. Individual specimen data accessed based on entry no. EN of specimen: |
| | ...EN | (num) sequential entry no. for specimen. Used in subscript calculation. |
| ...^(200*EN+800) | | (str) ACCNO.";".ACCDT.ACCTM.";".TYP.";".STAT.";".REQ, where |
| | ...ACCNO | (str) accession no. of day for lab section as 3N. |
| | ...ACCDT | (str) accession date as YYMMDD. |
| | ...ACCTM | (str) accession time to nearest half hour, encoded as single ASCII character, e.g., 12:30 AM = $001_8$, 12:00 Midnight = $030_8$. |
| | ...TYP | (str) specimen type code, e.g., "BLD". |
| | ...STAT | (str) emergency ("stat") flag indicated as "*"; otherwise null. |
| | ...REQ | (str) list of test request mnemonics, concatenated by commas. |

| Global | Logical | Access/Mapping/Meaning |
|---|---|---|
| | ...RCDAT | Result Category Data (Rep. Grp.) Results on tests in a result category. Mapped at nodes whose subscripts are offsets from above subscript for specimen, where offset is in the range 4:4:196, based on value of RC: |
| | ...RC | (num) result category number in range 1:1:49. Used in subscript offset calculation: |
| ...^(200*EN+800+(RC*4)) | | (str) RES1."";".res2.";"... ";".RESm, where: |
| | ...RES1 | (str) result for test 1 of this result category. |
| | ...RES2 | (str) result for test 2 of this result category. |
| | . | . |
| | . | . |
| | . | . |
| | ...RESm | (str) result for test m of this result category. |
| | | Note: A given test i is specific to a result category. E.g, test 2 in result category 4 (electrolytes) is "K" (potassium), whereas test 2 in result category 7 (liver function) is "Bil" (total bilirubin). If test i was not ordered or done, RESi is null. Total no. of tests m is specific to the result category. A result value of "..." means test is not complete. A non-null result is padded to a field length specific to the test as follows: One trailing blank or "#" if result is abnormal. Leading blanks as necessary for field length. |

An example printout, with annotations, of a portion of the actual global contents of file PAT follows:

```
^PAT(
-----
          Patient with ID=143204

^(4,1432,0) : 0        IUF
. . ^(1) : "143207"     ID
. . ^(2) : "CAJAL,DOLORES"  NAM
. . ^(3) : "F;421207"    SEX;DOB
. . ^(4) : 12364         ADM
. . ^(5) : "W21"         LOC
. . ^(120) : 6       SPECTOT   :  LABSEC Rep. Grp. LBNO=120(Chemistry)
. . . ^(4) : "1,4,6" RC#1     SPECIND Rep. Grp.
. . . ^(8) : "2,3,4" RC#2        "     "
. . . ^(28) : "2"    RC#7        "     "         CUMDIR Rep. Grp.
. . . ^(32) : "1,6"  RC#8        "     "
. . . ^(48) : "5"    RC#12       "     "
. . . ^(204) : "6"   RC#1        "     "         INTDIR Rep. Grp.
. . . ^(232) : "6"   RC#8        "     "
. . . ^(1000) : "124;740316 ;BLD::FBS,BUN,CRE"  Spec. 1
. . . ^(1004) : "127;;;;;;;"   RC#1 Results   RCDAT Rep. Grp.
. . . ^(1032) : ";;21;1.4;;;;" RC#8 Results
. . . ^(1200) : "127,740316 ;BLD;;CA,PO4,BIL,LDH,SGOT" Spec. 2
. . . ^(1208) : ";;;;...;..."            RC#2 Results  RCDAT
. . . ^(1228) : "1.7;;;;;;120;23;;"      RC#7 Results
. . . ^(1400) : "128;740316 ;BLD;*;E4" Spec. 3
. . . ^(1408) : "147;4.1;101;27.4;;"   RC#2 Results - RCDAT      SPECLST
. . . ^(1600) : "243;740318 ;BLD::BS,K" Spec. 4                 Rep. Grp.
. . . ^(1604) : ";108;;;;;"            RC#1 Results   RCDAT
. . . ^(1608) : "5.2;;;;"              RC#2 Results
. . . ^(1800) : "0247;740318 ;URN::VMA"Spec. 5
. . . ^(1848) : ";;;;...;;;;"          RC#12 Results -RCDAT
. . . ^(2000) : "136;740319 ;BLD;;BS,CRE" Spec. 6
. . . ^(2004) : ";107;;;;;"            RC#1 Results   RCDAT
. . . ^(2032) : ";;;;...;;;;"          RC#8 Results

. . ^(130) : 2     SPECTOT   :  LABSEC Rep. Grp. LBNO=130 (Hematology)
. . . ^(4) : "1,2"   CUMDIR
. . . ^(16) : "1"
. . . ^(204) : "2"   INTDIR
. . . ^(1000) : "076;740316 ;BLD;;WBC,RBC,HCT,PTT"
. . . ^(1004) : "4.4;;;37;6.8;;;;"
. . . ^(1016) : ";;;23;;;"           RCDAT                     SPECLST
. . . ^(1200) : "049;740319 ;BLD:*:HCT"
. . . ^(1204) ; ";;;39;;;;;"         RCDAT
          Patient with ID=150907
^(7,1509,0) : 0
. . ^(1) : "150907"
. . ^(2) : "TOWNSLEY, RICHARD"
. . ^(3) ; "M;360212"
          .
          .
          .
```

Miscellaneous comments on the global should be noted if they reflect on the rationale for the structure utilized. Example comments regarding PAT follow:

<u>Miscellaneous Comments</u>

(1) Subdivision of patient ID into 2 level subscripts IDX, IDY was done to improve access time, by reducing search for any given patient to about 1% of the population, involving retrieval of only 2 blocks.

(2) Compression of all lower level data into third and fourth levels of the global was done to improve packing density and reduce total number of accesses for related items via (a) naked references within the same level, and (b) retrieval of string values combining multiple fields.

(3) Packing efficiency (PE) by level is approximately as follows, for a clinical laboratory information system operational in a 450 bed teaching hospital, with 150 outpatient visits per day:

| level | PE | # of blocks (avg) |
|---|---|---|
| 1 | 78% | 1 |
| 2 | 32% | 1 |
| 3 | 64% | 1 |
| 4 | 87% | 3 |

# Chapter 7

# THE DOCUMENTATION SPECTRUM

The term "documentation" includes a broad spectrum of papers, documents, manuals, etc., which might be generated during the overall development of a given MUMPS application. The particular type of documentation produced usually corresponds to the current development phase of the application (system design phase, coding phase, etc.). A list of the various types of documents which might evolve is as follows:

(1) Feasibility Study
(2) Application Design Specifications
(3) Source Code Specifications
(4) File Specifications
(5) Application Test Specifications and Results
(6) User's Manual

A description of each of the above documents is given in the sections which follow. No attempt is made to specify the format of the documents, but only to describe the recommended content. It is not implied that all of these documents will be required to successfully document a MUMPS application. We often find that the feasibility document is not produced. Also, some MUMPS applications are designed to provide detailed instructions upon the user's request, and thus a user's manual may not be needed.

## 7.1 Feasibility Study

The feasibility study is a management level document that should provide the necessary information for evaluating a proposed application. The document should outline the application's objectives and capabilities, give time and cost estimates for development and implementation, and give a statement of the advantages offered if the application is implemented.

## 7.2 Application Design Specifications

The application design specifications should provide a general statement of the MUMPS application's functional purpose and operational capability in nontechnical terminology. It should also give a clear description of each job task, along with methodology and logic for doing the task. Data flow and I/O requirements should be specified. The application design specifications may be considered as both administrative level and programmer level documentation. The document can be used by management and other nonprogramming personnel to determine in general terms how an application is designed and what functions it will do. At the programmer level, the document serves as an interface between user and programmer by specifying the user's requirements of the MUMPS application.

A list and description of items recommended for inclusion in the application design specifications follow.

Application Overview

This is a narrative description of the MUMPS application's purpose and objectives in general terms. It should provide summary information on the organization and sequence of the application functions, along with information on whom the application is for and why it is being done.

Application Functions

A narrative detailed description of each function or task which the MUMPS application will perform. Input and output requirements for each function should be specified along with data file update and access procedures.

Backup and Recovery

A description of backup procedures along with procedures required to recover data, should there be a computer failure.

## 7.3 Source Code Specifications

Source code specifications are necessary for defining and updating MUMPS routines, programs, and packages. Thus, source code specifications can be considered as programmer level documentation. Chapters 3 and 4 give a complete description of the requirements and format for documenting MUMPS source code.

## 7.4 File Specifications

A description of each data file required by the application. This should include the format and description of specific data items contained in each file along with specifications on file accessibility and update requirements. A flowchart illustrating data flow and file procedures for each application function may be helpful. Chapters 5 and 6 describe the requirements and format for MUMPS file documentation.

## 7.5 Application Test Specifications and Results

The application's test specifications should provide an account of all tests and results on the application. For each test, the following information should be given:

(1) Purpose of the test.
(2) Identification - state what is being tested: routine, program, package, or logic function.
(3) Test Methodology - procedure and logic of the test; inputs and outputs required; driver or special test routines utilized; limitations, constraints, and conditions imposed by the test.
(4) Test Results - the test results should be given along with a sample listing illustrating the test procedures. The listing

can serve as documentation of how the test was performed as well as verification that it was actually done.

Probably the most difficult task in testing a MUMPS application is deciding what to test. Some programmers attempt to test every logical path through the source code. However, with many applications this is very difficult if not impossible to accomplish within a reasonable time frame. A more plausible approach is to test each logical component or function of the application. MUMPS lends itself to this approach because of its modularity; each job, section, task, etc. of the application is usually structured as a closed set of one or more MUMPS routines.

Application test specifications are considered as programmer level documentation.

## 7.6 User's Manual

This document should provide a complete description of an application's capabilities along with detailed instructions for running the application. The following are items recommended for inclusion in the User's Manual.

### Application Overview

This is a narrative description of the application's objectives and capabilities in user-oriented terms. It should provide summary information on the organization and sequence of application functions, along with information on whom the application is for and why it is being done.

### Interactions

This is the core of the User's Manual. It summarizes the application dialogue with the user, what action(s) the user can take, and what each action directs the computer to do. Certain actions taken by the user may result in an error condition. Therefore, a description of the error message, where the error condition might occur, the cause of the error, and the procedures necessary for recovery should be included.

### Examples

This would include listings illustrating the actual operation of the application. Whenever applicable, user input should be underlined to set it apart from the output.

### Glossary

A list of all data processing or otherwise unfamiliar terms used in the manual, along with their definitions.

## APPENDIX I

## GLOBAL FILE CONCEPTS

## GLOBAL FILE CONCEPTS

### The Nature of Globals

The global data management facility (sometimes called "globals") in MUMPS is noteworthy because of the high degree of flexibility it affords the programmer in designing file structures suitable to his particular application.[1] Features of symbolic data reference, multi-level data structuring, dynamic allocation and garbage collection, efficient handling of sparse arrays, and heterogeneity of element type permit globals to be used for a wide variety of data base needs.

Applications may differ greatly in the size of the data base to be accommodated, accessing methods to be supported, and multi-user coordination and communication requirements. In addition, because of the common data base handling capabilities for multi-user access and retrieval, globals can be used to communicate data among independent, concurrent, asynchronous processes, and to permit file processing and spooling tasks to run as "background" to more interactive "foreground" programs.

This versatility of globals gives rise to complexity of file design. As a consequence, there is a definite need for precise, complete documentation of a global data structure. Since a complex, multi-user data management application often requires implementation by several programmers of a large number of programs which must share files and intercommunicate in various ways, file design documentation is a first step in specifying and designing an application system. The clarity with which a global file is conceived and specified in advance of programming is an important factor in program structure and design. This emphasis on prior specification of file structure is much more significant than prior specification of program design, an ideal which is rarely followed in practice, though generally endorsed. For data management applications, in particular, file specification is an imperative antecedent to program implementation.

Because of the versatility of globals and the diversity of requirements to which they may be put, an understanding of the characteristics of a particular implementation of the global data management facility itself is necessary in order to obtain maximum efficiency and effectiveness of an application. Like any general purpose capability which can accommodate a great variety of potential usages, it is impossible to utilize globals to meet all possible needs with a high degree of efficiency. Many general purpose capabilities, however, solve this basic incompatibility between versatility and efficiency by meeting all needs with less than optimal efficiency. Globals are somewhat

---

[1] A.I. Wasserman, D.D. Sherertz, and C.L. Rogerson, MUMPS Globals and Their Implementation, MDC Doc. No. 2/1, 5/15/75.

unique in the extent to which they are implemented in such a manner that a file design may be highly tuned to implementation constraints, resulting in very high efficiency.

Achieving maximum effectiveness and efficiency with globals, in fact, requires a high degree of programmer cleverness. Yet such cleverness is often in direct contrast to clarity, which militates against a typical global file possessing a structure which is self-documenting with respect to the logical relationships of the data which led to its design. This is in contrast to certain other higher-level data definition facilities in which the logical structure inherent in the data is represented in the data structure declarations and references themselves. This is the result of the tradeoff in MUMPS between use of a high level general purpose language capability and the provision of capability for fine tuning at a systems level.

Characteristics of a global data management facility may differ widely among different implementations. An understanding of the role that particular constraints and characteristics played in a particular design is essential if program portability is to be facilitated. This is true not only for transfer of an application to other implementations, but also for maintenance on an existing implementation as peripheral device characteristics and operating system features tend to evolve over time. Globals are particularly sensitive to random access device characteristics, such as physical block size, number of blocks available per physical seek movement, total number of blocks available, and user storage costs, and to operating system characteristics, such as buffer handling, multi-user process scheduling, maximum block search times, and disc block allocation and "preallocation" facilities. These factors influence the programmer's planning and decision-making regarding the number of levels that a global data structure will have, size of any particular level, and the packing efficiency of data. It is therefore essential to document these considerations whenever they have significantly influenced the global file design.

## A Global Documentation Approach

A data base system, when considered without regard to implementation details, tends naturally to be hierarchically organized. Since globals themselves have a hierarchical structure, it is tempting to think of the logical relationships among data in terms of their realization in globals. However, many other considerations having more to do with physical implementation of a global facility influence a programmer in mapping a particular file into a global, and it is desirable to distinguish these considerations in documentation. This is especially true when one has the desire to make programs implementation-independent, and to enhance program transportability.

Much of the distinction between the "logical" or "normative" data structure of a file and the actual global structure in which it is realized or mapped could be ignored if the concern were purely local. Implementation and design in practice will rarely, after initial conceptualization, be concerned with anything but the actual global data structure. The logical structure represents an abstraction to enhance transportability.

## Logical File Structure Concepts

This section will discuss the logical relationships among data in a file. This discussion will form a basis for describing the structure of a file in implementation-independent terms, since the nomenclature and concepts involved are general to most, if not all, data management systems. It is useful, for both communication with non-MUMPS users, and for considerations of program portability among different MUMPS implementations or onto future versions of a system, to have this kind of a conceptual level in a system's documentation. This would be less of a concern if globals constituted a higher level data definition facility themselves; dynamic allocation without prior declaration, the ability to tailor structure to system efficiency constraints, and the necessity to explicitly program access methods, while powerful features of globals, lend obscurity to the use of a particular global structure, and increase the reliance on external documentation.

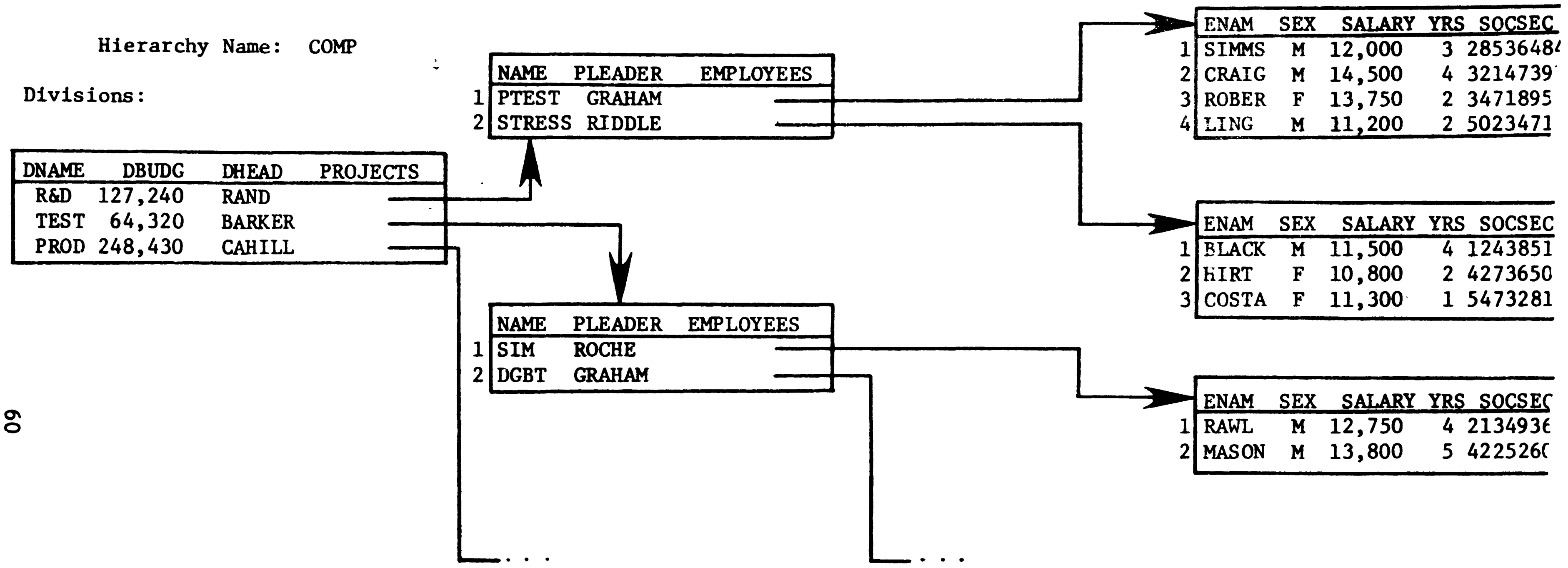
### Hierarchical Structures

A *file* may be generally defined as a collection of data that are logically related in some way. A hierarchical structure provides a natural representation for the kinds of files typically utilized in MUMPS systems. Hierarchical structures, when considered independently of physical realization, permit a representation of purely sequential files, i.e., with only one level of depth in the hierarchy, as well as complex files with multiple variable length fields and repeating subgroups, which can occupy multiple levels of depth in a hierarchy.

Consider the following hierarchical structure for a file of employees in a company, consisting of divisions, projects within divisions, and employees within projects.

Hierarchy Name: COMP

Divisions:



| DNAME | DBUDG | DHEAD | PROJECTS |
|---|---|---|---|
| R&D | 127,240 | RAND | |
| TEST | 64,320 | BARKER | |
| PROD | 248,430 | CAHILL | |

| | NAME | PLEADER | EMPLOYEES |
|---|---|---|---|
| 1 | PTEST | GRAHAM | |
| 2 | STRESS | RIDDLE | |

| | NAME | PLEADER | EMPLOYEES |
|---|---|---|---|
| 1 | SIM | ROCHE | |
| 2 | DGBT | GRAHAM | |

| | ENAM | SEX | SALARY | YRS | SOCSEC |
|---|---|---|---|---|---|
| 1 | SIMMS | M | 12,000 | 3 | 2853648 |
| 2 | CRAIG | M | 14,500 | 4 | 3214739 |
| 3 | ROBER | F | 13,750 | 2 | 3471895 |
| 4 | LING | M | 11,200 | 2 | 5023471 |

| | ENAM | SEX | SALARY | YRS | SOCSEC |
|---|---|---|---|---|---|
| 1 | BLACK | M | 11,500 | 4 | 1243851 |
| 2 | HIRT | F | 10,800 | 2 | 4273650 |
| 3 | COSTA | F | 11,300 | 1 | 5473281 |

| | ENAM | SEX | SALARY | YRS | SOCSEC |
|---|---|---|---|---|---|
| 1 | RAWL | M | 12,750 | 4 | 213493 |
| 2 | MASON | M | 13,800 | 5 | 422526 |

A Hierarchical Data Structure

At any logical depth in the hierarchy is a set of items organized as an array, in which columns are called data elements and rows are called entries or records. A data element represents an attribute associated with data, such as "employee name", "sex", "division budget". It may be labeled by a symbolic identifier such as "ENAM", "SEX", or "DBUDG", respectively. An entry is a set of values for each of the data elements for one member, and is labeled by an entry number. For example, the values of employee name, sex, salary, year of employment, and social security number in one row are those associated with an entry for an individual employee. Each cell holds the value for a given data element and entry.

The array of data elements and entries is referred to as a repeating group. The top level, the file itself, is a member of a repeating group, by convention, since it is an entry in a directory of files. Also, a repeating group may be contained in an entry of another repeating group, e.g., EMPLOYEES is a repeating group in an entry of the PROJECTS repeating group in the example, and PROJECTS is a repeating group in entries of the DIVISIONS repeating group. Thus, a data element may be associated with either an elemental datum, called a field, or with a repeating group. Note that in the concept of field, a datum is elemental only because it is used in this way. ENAM is a field for employee name, since name is treated in the application as a unit. If first, middle, and last names were individually of interest, three field data elements would be required.

For a data element associated with a repeating group contained within it, values for its entries are called pointers to instances of the repeating group. The data element with the identifier "PROJECTS" in the DIVISIONS repeating group, in the example, contains values which are pointers to instances of the PROJECTS repeating group.

Note that the concept of record or entry is ambiguous for multi-level hierarchical files, unless the logical depth of interest in the hierarchy is clearly specified, e.g., a DIVISION entry or a PROJECT entry within a specific DIVISION in the example. We shall use the term entry instead of record here, because there is less prior meaning of it to most individuals not accustomed to thinking of multi-level file structures.

## Accessing Hierarchically Structured Data

To retrieve a specific datum, it is necessary to identify both the data element desired and the entry of interest. For files structured to permit direct access, identification of the entry must be either by entry number, which is an integer identifying sequential position in the repeating groups of which it is a part, or by the values associated with one or more of its data fields. This specification of known items constitutes the access key to the entry. In MUMPS, access keys need not be sequential, as arrays can be sparse. For example, an employee's social security number may be the access key for retrieval of data about the employee, in a company's payroll file. The desired unknown items are values associated with other data elements in the goal entry. For a multi-level file, it is necessary to specify a sequence of keys to descend the hierarchy to a desired depth in order to retrieve values of data elements in an arbitrary goal entry.

The method by which keys are processed by a data management system to access the goal entry is not obvious in many systems, and depends on the access methods supported by the system. The values of entries for a particular data element may sometimes themselves be entry numbers or other access key data to permit retrieval of other entries in the same or a different repeating group. A data element which contains such access information is called a directory or index to the entries of interest. Since the entries of which the index items are a part may be accessed in some manner other than the method by which the goal entries are accessed, such directories or indexes permit secondary kinds or access to be supported. For example, a payroll file may be organized to permit direct retrieval of an employee's data by a key which is based on his social security number. This is the key for the primary access to employee information. To permit efficient retrieval in another manner, such as alphabetically by last name, or by job classification, or by salary, it is necessary only to establish a directory file, the entries of which are accessed on the basis of the secondary key, a retrieved index value of which can then be used as a key, e.g., social security number, in the primary access of the goal entry.

## Global Implementation Considerations

The MUMPS programmer has control over file design in a number of ways not available in many high level data management systems. This control is reflected in the range of choices for representation of hierarchical file structures such as discussed in the previous section. Note that a logical file need not have a one-to-one correspondence with a global name, but rather may occupy several globals, or a range of subscripts within a global. The following are some examples of the various choices available to the MUMPS programmer when designing a global file.

(1) Representation of Data Values
An individual data value may be stored as the value at a global node, it may be a subfield in a numeric or string datum, stored as the value at a node, or in either of the above cases, it may be encoded as a subscript itself or a sequence of subscripts.

(2) Attributes

An attribute of a data element may be determined explicitly or implicitly. An explicit attribute is one in which the identifier of the attribute is explicitly stored, along with the value. An implicit attribute is one in which the position of the value determines the attribute by programming convention. A fixed global subscript may be considered as an explicit identifier for an attribute, since it is precisely retrievable, and indeed mnemonic local symbols may be assigned to such fixed subscripts and used in the global reference. Consider for example, that we are at a given global level of a file in which patient data is stored, and that the patient's sex is stored as the value at a node with subscript 7. Then clearly a reference to ^(7) will retrieve the value for "sex", but assigning local symbol SEX=7 and then referencing ^(SEX) will also retrieve it. Since a global subscript is numeric, it may alternatively be considered that a subscript is an implicit identification of attribute by (numeric) position, e.g., the 7th subscript in a range of subscripts is associated with the value for "sex." In a packed string consisting of data values concatenated together, or a numeric datum calculated as an algorithmic combination of individual values, attribute is again implicitly identified.

(3) Entries

An entry may be represented as a global level, the subscripts within which indicate specific data elements of the entry. An entry may contain either single packed string or calculated numeric values, or a collection of them, subfields of which are values of the specific data elements in the entry. (As indicated above, such values may actually themselves be encoded as subscripts.) An entry may consist of the values stored at nodes with a range of subscripts within a global level, to improve packing efficiency, or it may, for reasons of optimization of access time, occupy several levels.

(4) Repeating Groups

A repeating group consists of a set of entries stored in one of the variety of ways described above, or combinations thereof. The additional issue for a repeating group is that it must contain a means for access to individual entries within it based on the access key to the entries. The additional range of options available are those concerned with ways of encoding access key information into the global. Here MUMPS globals again offer a wide variety of alternatives; the programmer must design his own representation of access key information, and structure the global accordingly. An access key may be based on either a sequential entry number of the value of some other data field(s). This key value may be encoded as one or more subscripts, permitting direct access to any entry via a multi-level global design. The key may alternatively be encoded as a calculated offset to a base subscript, permitting a repeating group to be stored in a range of subscripts in some relation to the subscript identifying the start of the repeating group. A repeating group may be string or numeric datum, or collection of such data, subfields of which are individual entries.

## Implications for Global Documentation

Infinite variety exists in the above areas, and the task of documentation involves explicitly stating the methods that were used. Globals are different from most data definition facilities that permit hierarchical data structures to be used. In many such systems, the access method to be used and the access keys can be simply declared, and are not reflected in the definition of the data structure itself. Whether the system uses the key to perform a linear search, binary search, or a hash table lookup, for example, is of little concern to the programmer concerned only with defining this data structure. In globals, by contrast, all such access procedures must be explicitly reflected in the structure of the global itself, and in the application programs written to store and retrieve data in a global. A binary search, for example, might be represented as a multi-level hierarchical organization of keys. The programmer can determine the optimal representation for the task he wishes to perform.

For reasons implied above, there is in general little correspondence between "levels" in a global structure and "depth" in the logical hierarchy of data. In the logical representation, depth corresponds to repeating groups. In a global, levels may be used to store repeating groups, parts of repeating groups, multiple repeating groups, or access keys.

In this sense, a global is not a high level facility for data structure definition, because the mapping of data items in the structure and the implementation of access methods are explicit responsibilities of the programmer. By contrast, it is a high-level facility for data structure implementation, since it provides the programmer with a large amount of versatility in design, while providing automatic capabilities for handling dynamic allocation and garbage collection.

## File Dynamics

Along with documenting static structure, both logically and as actually realized in a global, it is also desirable to describe the aspects of the file concerned with its dynamic properties. Since globals are used in multi-user data management environments, dynamic characteristics not only affect single users of a global but multiple users who may be interacting in some way through use of the structure. Issues that should be discussed, when appropriate, include the following:

(1) Range in Global File Size
Since the size of a global file may change dramatically in a relatively short period of time, it is sometimes necessary to be concerned with the extremes of file size, to ensure the availability of adequate free blocks to accommodate the global at its maximum. Size of a global should be related not only in terms of number of physical blocks required, but number of entries or size of entry, at the various logical levels of interest. Maximum, minimum, and average values for these parameters should be given where possible, since it may be necessary to optimize not only available storage for the particular global, but also storage requirements for other globals with different dynamic size characteristics.

(2) Purging Requirements and Frequency
For a dynamically changing global file, it may be necessary for the file to be purged periodically. This may be done automatically on a continuous basis, or it may require running periodic maintenance programs to accomplish the task. Alternatively, purging may be entirely a manual procedure

done at operator discretion. The mode of purging should be described, as well as the frequency and the conditions under which it is to occur. If purging is associated with a file dumping procedure, this should also be described.

(3) Communication through a Global
Another use of a global is as a means for communicating data among users, e.g., to synchronize processes, or to store a resource interlock. An example of this might be the use of a global to send messages between foreground and background processes. Use of ring buffers and similar structures, in which input/output "pointers" are maintained to the most recent message into the buffer, and the next message to be removed from the buffer, are frequent methods. Such uses and the programming conventions that are involved need to be made explicit.

## APPENDIX II

## BLANK DOCUMENTATION FORMS

# MUMPS ABSTRACT

ROUTINE ☐ PROGRAM ☐ PACKAGE ☐

IDENTIFICATION

APPLICATION AREA: MEDICAL ☐ OTHER ____________________

SHORT TITLE:

DATE LAST MODIFIED:

PURPOSE (up to 150 words; underline key words; continue on back if necessary):

SOURCE

AUTHOR (S):

CONTACT

NAME:

BUSINESS ADDRESS:

TELEPHONE NO.:

REQUIREMENTS

DIALECT

STANDARD ☐ OTHER ________ SUPPLIER ____________________

MACHINE:

PERIPHERALS

A/D ☐ CRT ☐ DECTAPE ☐ DISC ☐ MAGTAPE ☐ PRINTER ☐ OTHER ____________

PARTITION SIZE (characters):

GLOBAL SIZE (number of blocks and block size):

RESTRICTIONS (continue on back if necessary):

SIZE

NUMBER OF CHARACTERS OF CODE:

NUMBER OF ROUTINES:

AVAILABLE ITEMS

| | |
|---|---|
| COMMERCIAL SERVICE ☐ | SOURCE LISTING ☐ |
| DOCUMENTATION ☐ | USERS' MANUAL ☐ |

Return to: MUMPS Users' Group, 700 South Euclid Avenue, St. Louis, MO 63110

17 Sept. 1975

# MUMPS PACKAGE/PROGRAM FACE SHEET

**COMMON ITEMS**

PACKAGE/PROGRAM (delete one) NAME: DATE:

ENTRY ROUTINE (S):

PURPOSE (continue on separate sheet if necessary):

CONTACT'S NAME, BUSINESS ADDRESS AND TELEPHONE NUMBER:

DIALECT

STANDARD ☐ OTHER ____________________

MACHINE:

PERIPHERALS

A/D ☐ CRT ☐ DECTAPE ☐ DISC ☐ MAGTAPE ☐ PRINTER ☐ OTHER ____________

PARTITION SIZE USED:

LIST OF GLOBALS USED:

GLOBAL DOCUMENTATION APPENDED ☐

VARIABLE LIST APPENDED ☐

TEST RUN DATA APPENDED ☐

**PACKAGE ITEMS**

TOTAL NUMBER OF PROGRAMS:

PACKAGE FLOW CHART APPENDED ☐

PACKAGE INTERFACE SPECIFICATIONS APPENDED ☐

**PROGRAM ITEMS**

TOTAL NUMBER OF ROUTINES:

PROGRAM FLOW CHART APPENDED ☐

PROGRAM INTERFACE SPECIFICATIONS APPENDED ☐

| | |
|---|---|
| Package: | A collection of programs which function together in an application. |
| Program: | A closed set of routines used to perform a specific task. |

# MUMPS ROUTINE DOCUMENTATION

**ENTER THESE ITEMS ON THIS SHEET**

*ROUTINE NAME: DATE:____________________

PURPOSE:

SIZE (number of characters in routine):

**CHECK WHICH OF THESE ITEMS ARE PROVIDED ON ATTACHED SHEETS**

MULTIPLE ENTRY POINTS

☐ Line Label
☐ Purpose
☐ Requirements

LOCAL VARIABLES

☐ Name
☐ Purpose
☐ Cross-Reference Listing

GLOBALS

☐ Name
☐ Location of Global References

DETAILED DESCRIPTION

☐ Narrative Description
☐ Listing
☐ Flow Chart

* A routine is defined as a collection of MUMPS statements filed, called, and/or overlayed as a single unit.

17 Sept. 1975

# MUMPS FILE SUMMARY

FILE NAME: DATE:

GLOBAL (S):

PURPOSE:

SIZE:

NUMBER OF LEVELS:

LEVELS WITH DATA:

PACKAGES AND PROGRAMS UTILIZING THIS FILE:

DOCUMENTATION PROVIDED

| | Logical Structure | Physical Structure |
|---|---|---|
| Narrative | ☐ | ☐ |
| Tabular | ☐ | ☐ |
| Pictorial | ☐ | ☐ |

ANNOTATED REPRESENTATIVE LISTING ATTACHED ☐

RETENTION CHARACTERISTICS:

COMMENTS:

## MUMPS Development Committee Manuals

| MDC Doc. No. | Identification |
|---|---|
| NBS Handbook 118 | January 1976 (with errata sheets thru March 9, 1976), MUMPS Language Standard |
| | Part I: MDC/28, 3/12/75, MUMPS Language Specification M. E. Conway |
| | Part II: MDC/33, 9/17/75, MUMPS Transition Diagrams D. D. Sherertz and Anthony I. Wasserman |
| | Part III: MDC/34, 9/17/75, MUMPS Portability Requirements E. A. Gardner and C. B. Lazarus |
| 29 | 5/28/75, MUMPS Interpreter Validation Program User Guide J. Rothmeier and P. L. Egerman |
| 30 | 6/25/75, MUMPS Translation Methodology P. L. Egerman, C. B. Lazarus and P. T. Ragon |
| 35 | 10/14/75, MUMPS Documentation Manual L. J. Peck and R. A. Greenes |
| 1/11 | 6/13/75, MUMPS Primer M. E. Johnson and R. E. Dayhoff |
| 2/1 | 5/15/75, MUMPS Globals and Their Implementation A. I. Wasserman, D. D. Sherertz and C. L. Rogerson |
| 2/2 | 5/30/75, Design of a Multiprogramming System for the MUMPS Language A. I. Wasserman, D. D. Sherertz and R. W. Zears |
| 2/3 | 6/15/75, Implementation of the MUMPS Language Standard A. I. Wasserman and D. D. Sherertz |
| 3/5 | 8/31/76, MUMPS Programmers' Reference Manual M. E. Conway and P. L. Egerman |

Distributed by the MUMPS Development Committee. For information, contact the MUMPS Users' Group, 700 South Euclid Avenue, St. Louis, MO 63110.