

AD-A116 591

ALFRED P SLOAN SCHOOL OF MANAGEMENT CAMBRIDGE MA CEN--ETC F/6 9/2
FSTV: THE SOFTWARE TEST VEHICLE FOR THE FUNCTIONAL HIERARCHY OF--ETC(U)
JAN 82 M HSU N00039-81-C-0663

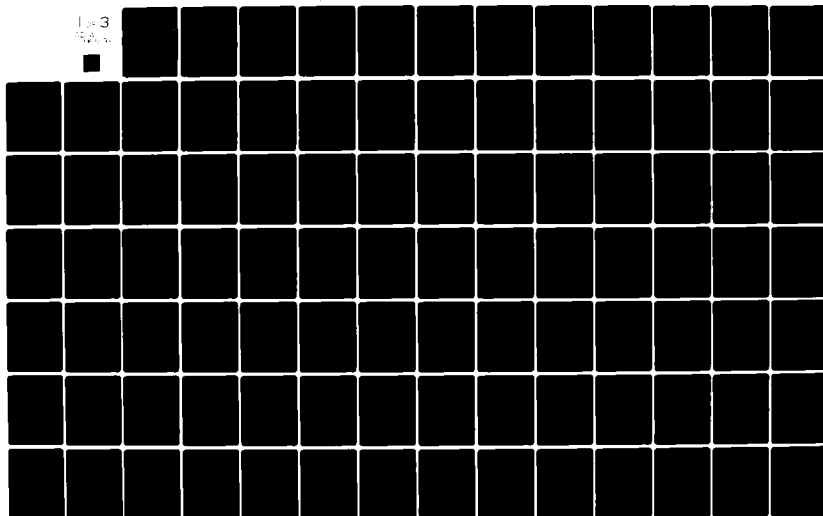
UNCLASSIFIED

CISR-M010-8201-09

NL

Box 3

Box 3



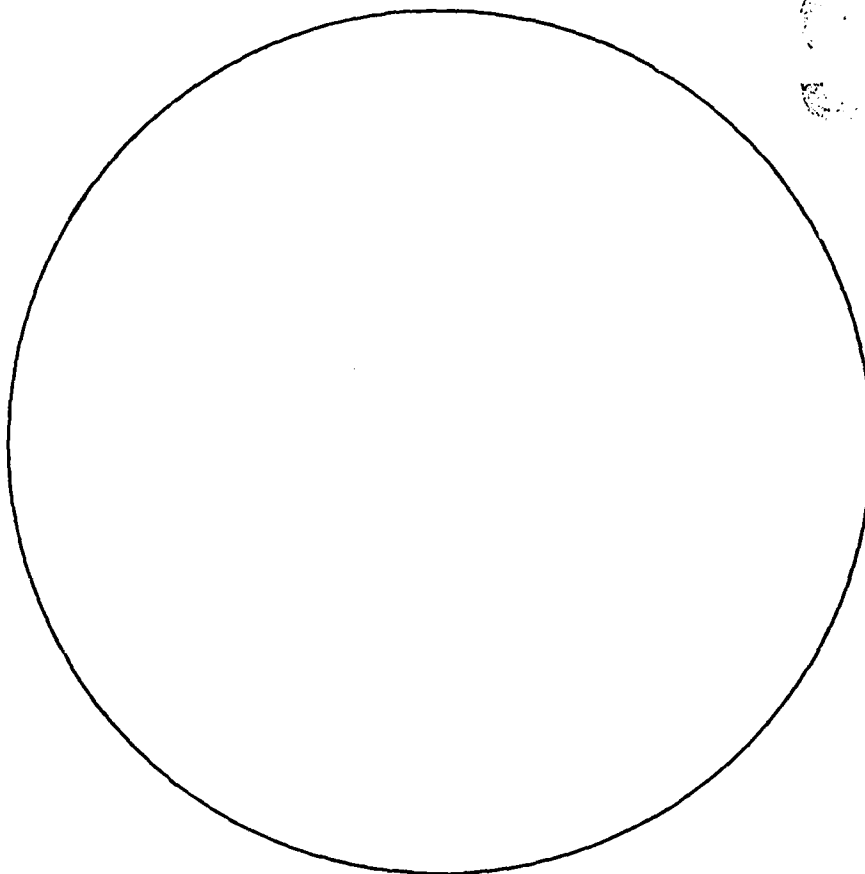
AD A116591

DMC FILE COPY



(12)

DMIC
JUL 7 1982



Center for Information Systems Research

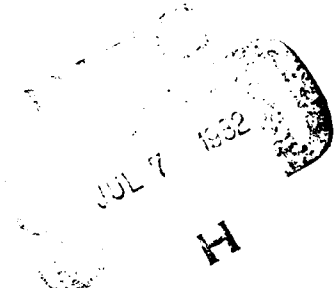
Massachusetts Institute of Technology
Sloan School of Management
77 Massachusetts Avenue
Cambridge, Massachusetts, 02139

DISTRIBUTION STATEMENT A
Approved for public release;
Distribution Unlimited

Contract Number N00039-81-0663 (MIT # 91445)
Internal Report Number M010-8201-09
Deliverable Number **1**

(12)

FSTV:
THE SOFTWARE TEST VEHICLE
FOR THE
FUNCTIONAL HIERARCHY OF THE
INFOPLEX DATABASE
COMPUTER



Technical Report # 9

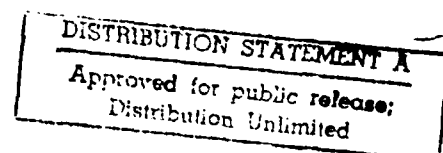
By

Meichun Hsu

January, 1982

Principle Investigator:
Professor Stuart E. Madnick

Prepared for:
Naval Electronics Systems Command
Washington, D.C.



REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER Technical Report #9	2. GOVT ACCESSION NO. AD A116 591	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) FSTV - The Software Test Vehicle For The Functional Hierarchy of The Infoplex Database Computer		5. TYPE OF REPORT & PERIOD COVERED
7. AUTHOR(s) Meichun Hsu		6. PERFORMING ORG. REPORT NUMBER M010-8201-9
9. PERFORMING ORGANIZATION NAME AND ADDRESS Sloan School of Management, Massachusetts Institute of Technology Cambridge MA 02139		8. CONTRACT OR GRANT NUMBER(s) N0039-81-c-0663
11. CONTROLLING OFFICE NAME AND ADDRESS		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office)		12. REPORT DATE January 1982
		13. NUMBER OF PAGES 270
		15. SECURITY CLASS. (of this report) Unclassified
		15a. DECLASSIFICATION/ DOWNGRADING SCHEDULE
16. DISTRIBUTION STATEMENT (of this Report) Approved for public release, distribution unlimited		
17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)		
18. SUPPLEMENTARY NOTES		
19. KEY WORDS (Continue on reverse side if necessary and identify by block number) database computer, database management system, software test vehicle, hierarchical systems		
20. ABSTRACT (Continue on reverse side if necessary and identify by block number) This report describes in detail the current implementation of the Software Test Vehicle (STV) for the Functional Hierarchy of the INFOPLEX database computer. The purpose of STV is to provide a better understanding of the architecture and functionalities of the INFOPLEX design by emulating its architecture and simulating its functionalities in software. It aims at validating the communication protocols among its various components,		

✓
tightening functional algorithms for database management and data movements, providing behavioral and preliminary performance information concerning the architecture, and serving as a test bed before realizing the design in the hardware prototype.

Implementation of FSTV is based on a preliminary design of the Functional Hierarchy presented in Hsu80. In that design, data base management functions are decomposed into hierarchical levels, each level to be implemented as a level of the Functional Hierarchy. The current version of FSTV is implemented with special attention paid to a richer set of data base capabilities and architectural compatibility.

Accession For	
NTIS GRA&I	☒
DTIC TAB	☐
Unannounced	☐
Justification	
By	
Distribution/	
Availability Codes	
Avail and/or	
Dist Special	
A	



ABSTRACT

This report describes in detail the current implementation of the software test vehicle (STV) for Functional Hierarchy of INFOPLEX data base computer.

The purpose of STV is to provide a better understanding of the architecture and functionalities of the INFOPLEX design by emulating its architecture and simulating its functionalities in software. It aims at validating the communication protocols among its various components, tightening functional algorithms for database management and data movements, providing behavioral and preliminary performance information concerning the architecture, and serving as a test bed before realizing the design in the hardware prototype.

Implementation of FSTV is based on a preliminary design of Functional Hierarchy presented in <Hsu80>. In that design, data base management functions are decomposed into hierarchical levels, each level to be implemented as a level of Functional Hierarchy. The current version of FSTV is implemented with special attention paid to a richer set of data base capabilities and architectural compatibility.

6.0	CONCLUDING REMARKS AND FUTURE DIRECTIONS	92
6.1	Memory Management Level Revisited	92
6.2	Internal Schema Level Revisited	93
6.3	The Entity Level Revisited	95
6.4	Location and Management of Catalogues and Working Buffers	95
REFERENCES		98
APPENDIX		100
	Appendix I (Sample Terminal Session)	100
	Appendix II (Macro Declarations)	116
	Appendix 3 (Complete Program Listing)	131

TABLE OF CONTENTS

Abstract	1
Table of Contents	2
1.0 OVERVIEW OF THE REPORT	4
2.0 OVERVIEW OF DATA BASE SYSTEM CAPABILITIES OF FSTV	7
2.1 Structure of User Session	9
2.2 Data Definition Session	12
2.3 Data Definition Query Session	15
2.4 Data Manipulation Session	16
2.4.1 Database Query	16
2.4.2 Creating Entities	18
2.4.3 Modifying Entities	19
2.4.4 Deleting Entities	20
2.5 Data Base Internal Schema	21
3.0 IMPLEMENTATION CHARACTERISTICS OF FSTV	25
3.1 Emulation of a Multi-level Multi-processor Architecture	25
3.1.1 Scope and Storage Class of Variables	26
3.1.2 Inter-level Communication	27
3.2 Functional Decomposition	32
4.0 FSTV INTER-LEVEL COMMUNICATION SERVICE ROUTINES	37
4.1 Insulating From SEND Protocol -- Use of SVCS1 and SVCS2	37
4.2 Insulating from Control Structure -- Additional service routines	41
5.0 OVERVIEW OF FSTV FUNCTIONAL PROGRAM MODULES	44
5.1 Memory Management Level	47
5.1.1 Data Model at the Memory Management Level	47
5.1.2 System Databases and Data Definition Modules at the Memory Management Level	48
5.1.3 Data Manipulation Modules at the MM Level	49
5.1.4 Initialization at the Memory Management Level	50
5.1.5 Subroutines at the Memory Management Level	51
5.2 Internal Schema Level	52
5.2.1 Data Structures at the Internal Schema Level	52
5.2.2 System Databases and Data Definition at Internal Schema Level	53
5.2.3 Data Manipulation at the Internal Schema Level	58
5.2.4 Initialization at Internal Schema Level	69
5.2.5 Subroutines at Internal Schema Level	70
5.3 Entity Level	71
5.3.1 Data Structures at Entity Level	71
5.3.2 System Databases and Data Definition Modules at Entity Level	72
5.3.3 Data Manipulation at Entity Level	76
5.3.4 Initialization at Entity Level	79
5.3.5 Subroutines at Entity Level	81
5.4 The User Interface Level	82
5.4.1 User Interface Modules	82
5.4.2 Other Subroutines at User Interface Level	85
5.5 The driver module - FSTV	88

1.0 OVERVIEW OF THE REPORT

This report describes in detail the current implementation of the software test vehicle (STV) for Functional Hierarchy of INFOPLEX data base computer. The basic concepts of INFOPLEX are found in <Madnick 78> and the reader is assumed to have some understanding of the proposed architecture of this data base computer.

The purpose of STV is to provide a better understanding of the architecture and functionalities of the INFOPLEX design by emulating its architecture and simulating its functionalities in software. It aims at validating the communication protocols among its various components, tightening functional algorithms for database management and data movements, providing behavioral and preliminary performance information concerning the architecture, and serving as a test bed before realizing the design in the hardware prototype.

The STV project is divided into the Functional Hierarchy STV (FSTV) and the Storage Hierarchy STV (SSTV). In addition, there is a Control Structure Program which provides an emulation of the multi-level multi-processing hardware architecture and

communication protocols. FSTV and SSTV are to be developed separately, each integrated with a separate copy of the Control Structure Program. However, the eventual goal is to produce an integrated version, in which FSTV and SSTV are coupled together through a single copy of the Control Structure Program.

Implementation of FSTV is based on a preliminary design of Functional Hierarchy presented in <Hsu80>. In that design, data base management functions are decomposed into hierarchical levels, each level to be implemented as a level of Functional Hierarchy. A 3-level version of FSTV was implemented by Bruce Blumberg <Blumberg81>. His implementation followed the philosophy of hierarchical decomposition and incorporated many distinctive features. However, his version did not take the hardware architecture of Functional Hierarchy into account, and could not be integrated with the Control Structure. Based on his experience and the availability of a Control Structure implemented by Tak To <To81>, the current version of FSTV is implemented with special attention paid to a richer set of data base capabilities and architectural compatibility.

This report is divided into 6 chapters. The next chapter provides an overview of the functionalities of FSTV as a data base management system: what it does, what data model it implements, and how it interacts with the user. Chapter three provides a description of the implementation characteristics of FSTV. In particular, it contains conventions used to preserve

architectural compatibility and to interface with the Control Structure. Chapter four and five are primarily program documentation. A description of a set of service routines devised to ease the task of interfacing with the Control Structure is presented in Chapter four, while Chapter five contains an explanation of functional program modules in the current implementation. Chapter six concludes this report by pointing out directions for future refinement. A sample terminal session is included in Appendix 1. Appendix 2 contains a list of Control Blocks (used as inter-level interface) and external/static variables. Finally, a complete program listing is found in Appendix 3.

2.0 OVERVIEW OF DATA BASE SYSTEM CAPABILITIES OF FSTV

FSTV currently implements a subset of functionalities proposed in the preliminary design <Hsu80>. The proposed architecture with respect to data abstraction is similar to that of the ANSI/SPARC design, basically a 3-level architecture which contains an external schema level, a conceptual schema level and an internal schema level. In our design, however, we have added a base data model schema level to capture data definition of the integrated data base and to support multiple conceptual schemas defined on top of, and mapped to, the base data model schema. This way, a conceptual schema may use a data model which is different from that of the base schema, effecting support of a multiple data model data base management system. External schemas can then be defined on top of, and mapped to, a conceptual schema. This 4-level architecture is depicted in the following figure.

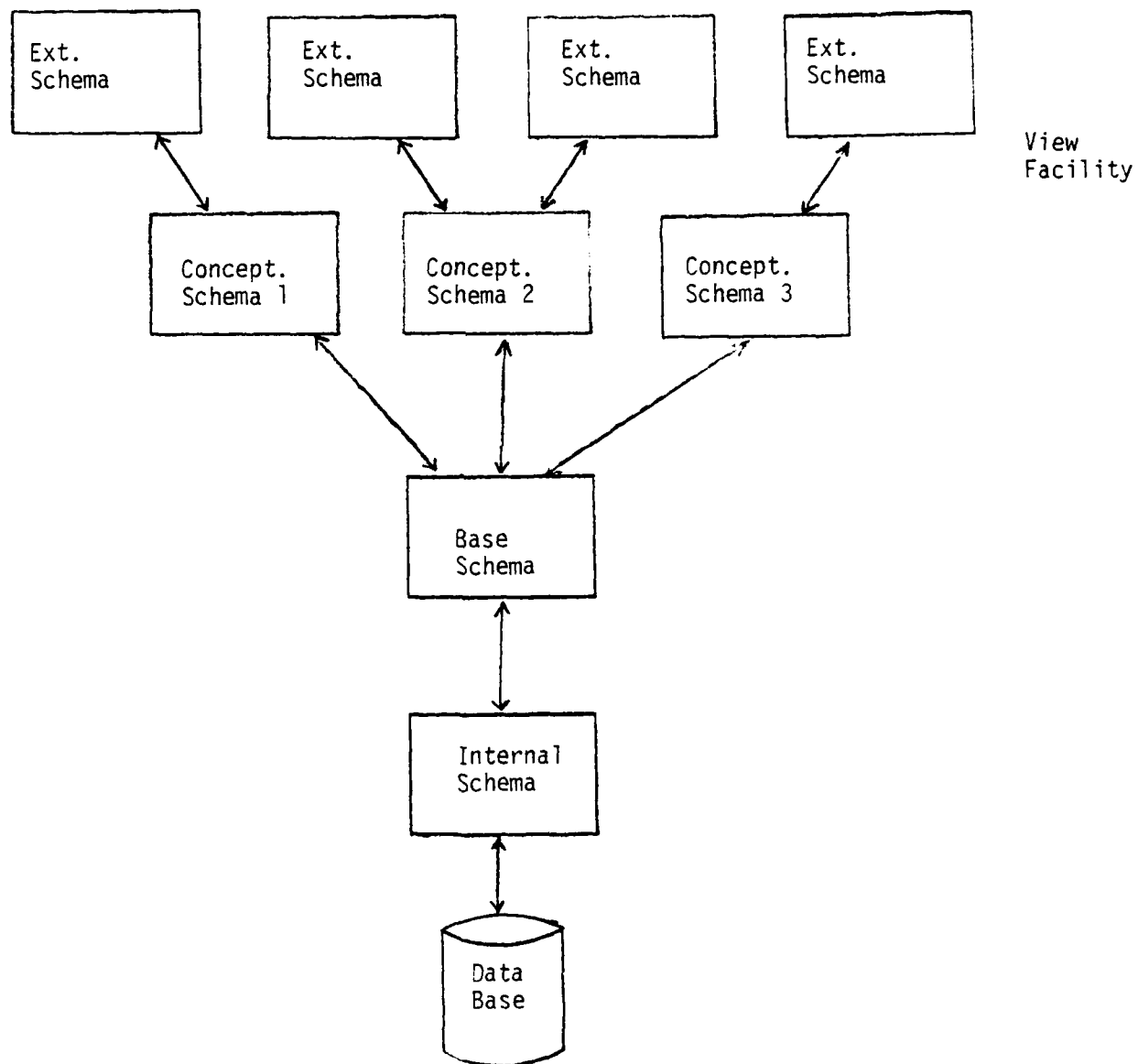


Figure: Architecture of INFOPLEX with respect to data abstraction

The upper two levels in the figure are aggregatedly called the view facility. In the current version, the view facility is not available, and the base data model implemented at the base schema level is an entity network data model. Therefore, this level is called the Base Schema Level or the Entity Level in FSTV.

This overview chapter describes what the system can do and how a user may interact with the system. It also provides a brief description of the entity network data model and the internal schema used in FSTV. A sample terminal session is included in the Appendix.

2.1 STRUCTURE OF USER SESSION

FSTV interacts with a user through a user session in the User Interface Level. Since currently only the DBA session is available, the user will enter the DBA session, in which he may define or manipulate the database. If the former is chosen, a data definition session is initiated. If the user chooses data manipulation, then he may create, delete, modify and query the database. The User Interface Level guides the user into the desired session and then accepts input strings, parses them and formulates proper requests to go down to the next level. It also displays error messages or return data at the user's terminal

based on the return from the next level. The structure of the user interface is depicted in the following figure.

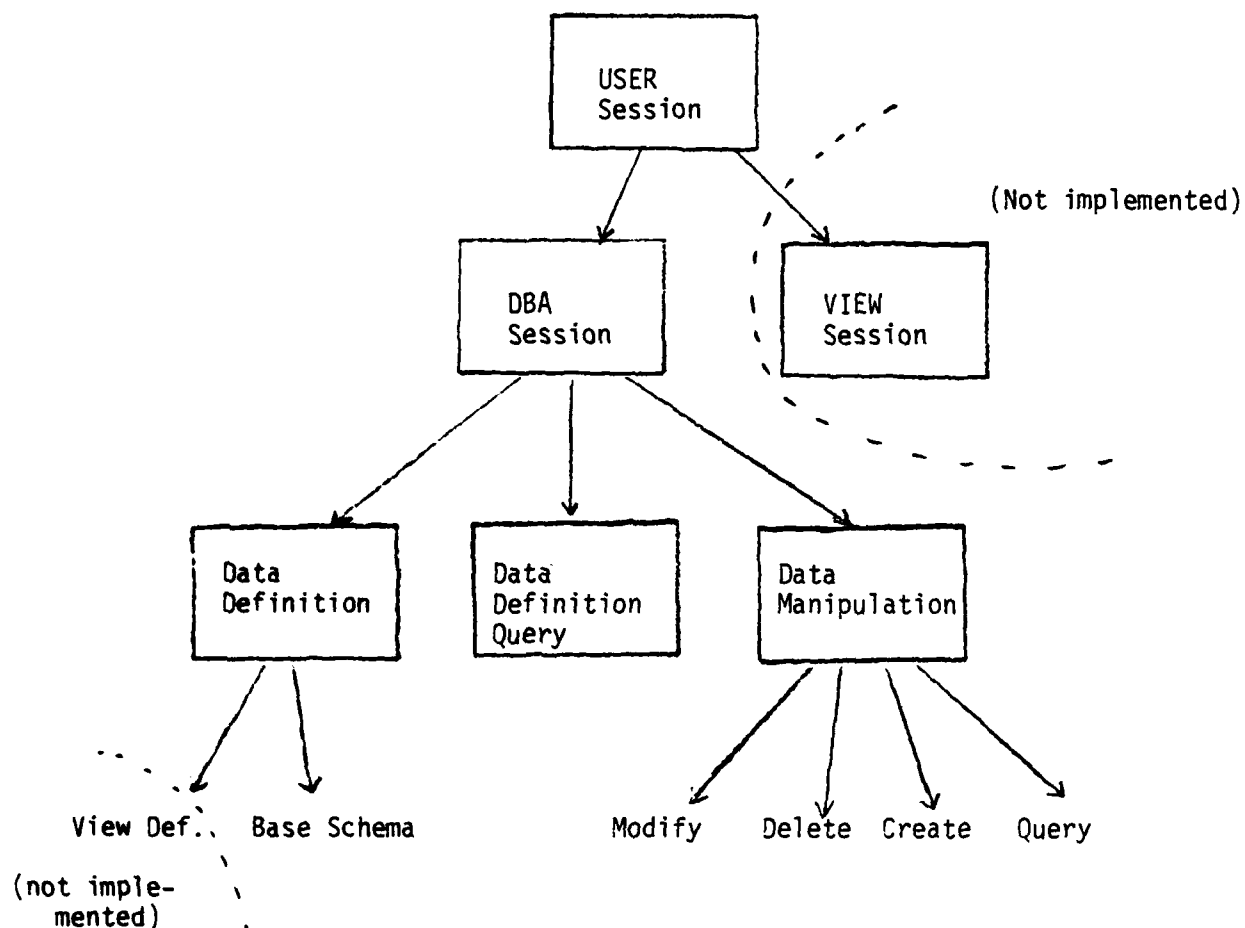


Figure: Structure of the User Sessions

2.2 DATA DEFINITION SESSION

The DBA defines the data base in an entity network data model. This model views the world as a collection of entities and attributes of these entities. Generically similar entities are grouped into entity sets. Entities in one entity set will share the same set of attributes.

The DBA will model the application environment into entity sets. For each entity set, he may define an arbitrary number of attributes. The domain of an attribute can be either a value set or an entity set. It is through the latter that the entity sets defined can be made to explicitly relate to each other and form a network.

This entity network data model bears many properties that recent research in the area of data modelling has advocated. Its basic principles are outlined in <Hsu80> and a draft of a detailed description of this model, including its retrieval and update primitives, is forthcoming. However, the current implementation extracts only a subset of its intended set of semantics. In particular, concepts such as generalization and inter-entity-set aggregation are not provided. A virtual information facility is also lacking. Nevertheless, the basic structure of the data model is preserved such that additional

semantics may be added to without a complete rewrite of the current implementation.

Before we describe the data definition session in detail, we remind the reader of a few definitions used in the entity network data model. An attribute with a value set as its domain is called a value attribute; on the other hand, if its domain is an entity set, the attribute is called an entity attribute. Also, a direct attribute is an attribute which is directly related to the entity set. In contrast, a derived attribute is an attribute of an attribute. For example, in the following figure which describes an entity network data base schema, EMPNAME is a direct attribute of the entity set EMPLOYEE, while DEPTNUM is a derived attribute of EMPLOYEE because it is a direct attribute of the entity set DEPT, which in turn is the domain of the WORKS_IN attribute of EMPLOYEE.

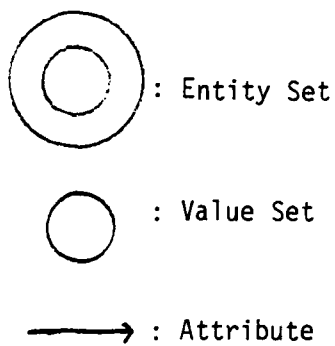
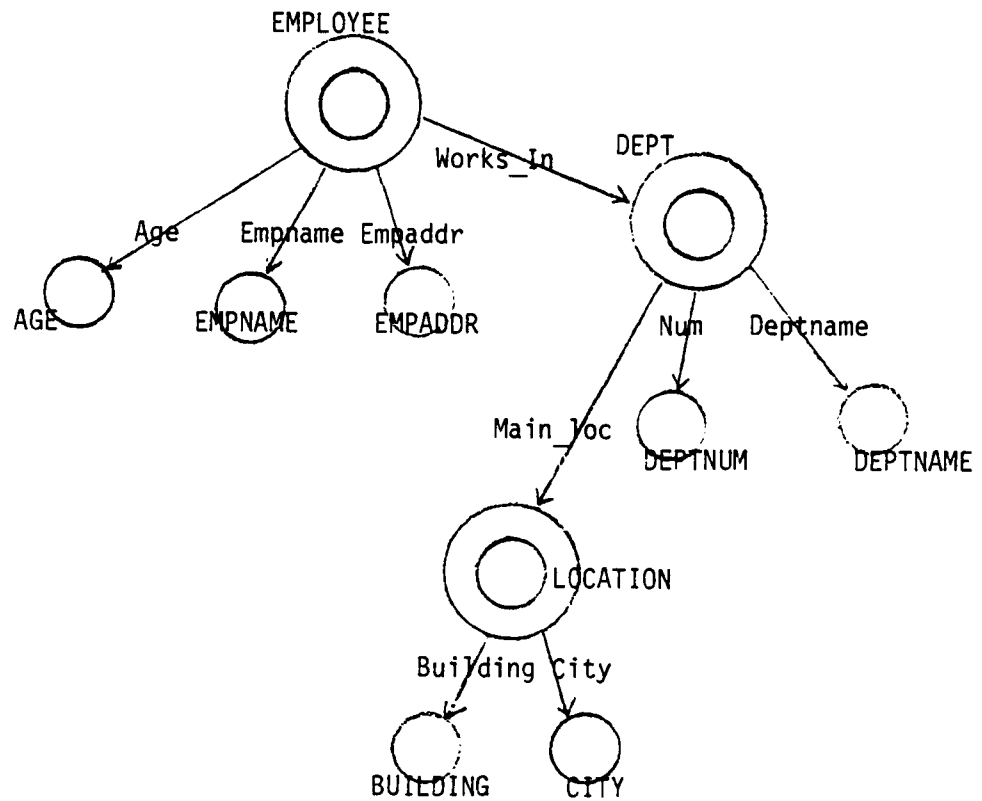


Figure: An Example of the Entity Network Schema

The Base Scehma Level, as currently implemented, will accept definitions of entity sets and their attributes. To define the base schema, the DBA enters the Base Schema Data Definitionn Session. An entity set is to be defined by giving an entity set name. An attribute definition, on the other hand, includes the name of the attribute, its domain type (i.e., value set or entity set), and its funciton type (i.e., many-to-one, one-to-one, or key). If the attribute is a value attribute, the definition also specifies the value type (i.e., numeric or character) and maximum length of the value (in number of characters), and if the value type is numeric, the allowable numeric range. If it is an entity attribute, only the name of the domain entity set need be given.

To facilitate ease of use, the user is provided with a panel to describe the data definition. Appropriate prompts are given in the panel.

Data definition can be incremental; i.e., new entity sets and additional attributes for existing entity sets may be defined after some data has been inserted into existing entity sets.

2.3 DATA DEFINITION QUERY SESSION

A user may choose to enter this session to discover what entity sets and attributes have been defined for the database. He may choose to look at the definition of all entity sets or that of a particular entity set. He may also choose to look at the definition of all attributes of an entity set or that of a particular attribute. The system will provide appropriate prompts as the user proceeds.

2.4 DATA MANIPULATION SESSION

In an entity network data base, entities may be created and deleted. Attribute values can be inserted, modified or deleted. A non-procedural query language can be used to retrieve information from the database. To do these, the user will get into the data manipulation session.

2.4.1 DATABASE QUERY

To do a query, the user enters the Query Subsession. He will give the name of the entity set and a list of attribute names of this entity set he would like to see. If any of the attributes

has a predicate (e.g., >150, ='Stu Madnick'), the user will give it along with the attribute list. Multiple predicates are allowed on an 'AND' basis. (No 'OR' capability is provided for in the current version.)

The attribute list may also include derived attributes. The user expresses the derived attribute as a series of direct attributes that would lead to that derived attribute. For example, the city of the main location of the department an employee works in, using the example database shown in the above figure, is expressed by

WORKS_IN (MAIN_LOC (CITY))

as a derived attribute of the employee entity. The derivation path can be arbitrarily deep. An example query statement is given below:

EMPLOYEE

EMPNAME, EMPADDR, AGE > 50, WORKS_IN (DEPTNUM = 15)

This query wants the employee name, address, age and the department number of the department the employee works in, for those employees whose ages are greater than 50 and who work in department number 15.

2.4.2 CREATING ENTITIES

To create entities, the user enters the Create Subsession. The user will give the name of the entity set he is to create entities for, and a list of attributes that he will give values to. He then will provide the values of these attributes at the prompt for data. The attribute list may include derived attributes, in which case the value given for that attribute is used to identify an existing entity that the new entity is to associate with. For example, in creating an employee, one of the attributes may be WORKS_IN (DEPTNUM), and the corresponding attribute value given by the user is used to identify the department the user works in. However, the new entity cannot be made to relate to a non-existing entity. In our example, if the department number given does not identify an existing department entity, the employee cannot be created.

To create an entity, the user can select the set of attributes of which values will be given. That is, an entity may be created while values of some attributes of the entity are not known.

The system will also intercept values of attributes that violate the attribute definition. This includes key violation, value type violation and numeric range violation.

2.4.3 MODIFYING ENTITIES

Values of attributes of existing entities (i.e., entities already created using the CREATE command introduced in the previous subsection) may be modified. To do so, the user must enter the Modification Subsession. Modification of attributes takes 3 forms: insert, modify and delete. However, the user must first identify the entity to be modified by giving value(s) of its attribute(s) that will uniquely identify the entity in the entity set. Then he will specify the attributes to be modified, prefixing them with a modification operator (i.e., insert, modify or delete). At the prompt for data, the user will enter new values of these attributes. The following is an example of a modify command:

```
EMPLOYEE
-ID:EMPNUM,-INSERT:WORKS_IN(DEPTNUM),-MODIFY:EMPADDR
105, 14, '550 Memorial Drive'
```

where EMPLOYEE in the first line specifies the entity set name, and the second line provides the information about what to be modified: the employee entity to be modified is identified by employee number 105, the attribute of this employee to be inserted is a derived attribute which asserts that this employee now works in department 14, and this employee's address is to be changed to '550 Memorial Drive'.

Notice that the interpretation of a derived attribute appearing in the modify command is the same as that in the create command; that is, it merely associates the entity being modified to an existing entity, but does not create any new entities or values. An error message will be given if the associated entity cannot be found in the existing data base. Also, violation of attribute definition will cause the modify command to be rejected.

2.4.4 DELETING ENTITIES

To delete an entity, the user enters into the Delete Subsession. All that the user has to specify is a set of attributes that will uniquely identify the entity to be deleted. All other attributes of this entity will automatically be removed. An example is given below:

```
EMPLOYEE
EMPNUM
105
```

This command deletes the employee entity whose number is 105.

2.5 DATA BASE INTERNAL SCHEMA

The data base internal schema describes how data in the data base is actually stored. It includes aspects such as access paths and record formats. This section provides a brief description of the internal schema used in the current version of FSTV, which is implemented at the Internal Schema Level and the Memory Management Level.

The internal schema used in FSTV is a simple network type of link list. Data elements are grouped into Primitive Sets (Psets). For each primitive set, an arbitrary number of Binary Associations (Bsets) can be defined. When a data element is to be stored, the level of FSTV above the Internal Schema Level identifies the Pset this data element belongs to, and passes this to the Internal Schema level to be stored in that Pset. Likewise, the data elements a data element associates with through certain Bsets can also be given, and later can be retrieved together with that data element. In order to do so, the upper level will have to define Psets and Bsets for the Internal Schema Level, and remember the meaning of each Pset and Bset. The Internal Schema Level will provide IDs for Psets and Bsets defined. This concept of BEU is depicted in the following figure.

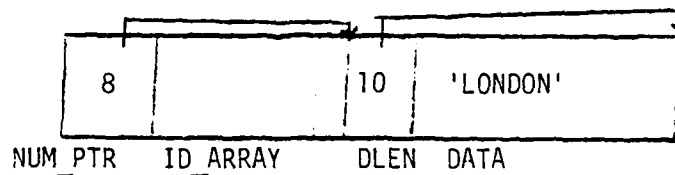


Figure: An example BEU

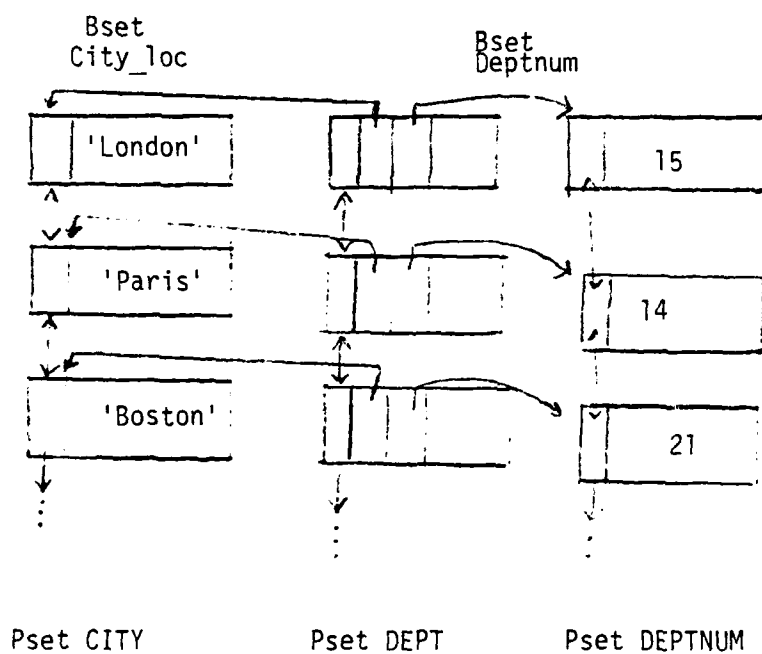


Figure: An example BEU network

The way the internal schema level organizes these data elements is through linkage. Each data element is implemented as a Basic Encoding Unit (BEU). A BEU contains the data and some control information. In the current version, a BEU contains an ID array, the size of which is indicated by the NUM_PTR field of a BEU, and a data field, the size of which is indicated by the DLEN field of a BEU. BEUs of one Pset are chained together through a bi-directional link list, making use of two slots in the ID array. Association with other data elements is implemented by storing the IDs of the latter in pre-allocated slots in the ID array. By 'pre-allocated' we mean that the position of the slot determines the meaning of the association, or the Bset id. Through this simple mechanism, data elements and their associations are remembered by the data base.

In the current version, there are no special access paths built for the purpose of improving performance of a search, such as indices and hash tables. There is also no intelligent access path selection algorithm to identify the most efficient way of locating a data element given its direct or derived associations. While these capabilities occupy the highest priority in future refinement, the basic functionalities of creating, deleting, modifying and retrieving data elements are all present.

The Memory Management Level responds to the Internal Schema Level's requests to store and retrieve BEU's. When a BEU is stored, the Memory Management Level provides an id which can be

used as a key for retrieval of the BEU. This level currently manages the virtual storage by linearly assigning BEU's to be stored in the next available address. No segmentation or clustering is provided for.

3.0 IMPLEMENTATION CHARACTERISTICS OF FSTV

In Chapter 2, discussion centered around the capabilities of the FSTV from the point of view of a DBA or a database user. In this chapter we shall describe implementation considerations which characterize FSTV as a Software Test Vehicle for the Functional Hierarchy of INFOPLEX. The content of this chapter can also be viewed as a set of strategies and programming conventions used in the STV project, and guidelines to be followed in future software modifications and additions. The following two subsections describe implementation characteristics relating to the emulation and functional decomposition aspects, respectively, of FSTV.

3.1 EMULATION OF A MULTI-LEVEL MULTI-PROCESSOR ARCHITECTURE

As mentioned earlier in Chapter 1, one of the purposes of the STV is to validate proposed algorithms in an emulated multi-level multi-processor architecture and to observe their behavior. The Control Structure Subsystem of the FSTV is devised to provide

this emulated environment. While the modules in the FSTV do not have to be aware of details of the (emulated) hardware environment, its program structure and communications between modules must conform to those demanded by the Control Structure. The Control Structure, on the other hand, will provide mechanisms for multi-processing and global bus and gateway controller emulation during execution of the FSTV modules.

Two types of programming conventions are observed during implementation of FSTV in order to facilitate working with the Control Structure. They are described in the following subsections.

3.1.1 SCOPE AND STORAGE CLASS OF VARIABLES

The context switching mechanism used in the Control Structure requires that the static and the controlled variables be used with caution. (Refer to chapter 7 of <To81>.) In essence, the FSTV program modules should make use of the based and automatic variables and refrain from using the static and controlled variables as much as possible. In the current implementation of the FSTV, static variables are used in association with the initialization modules only; i.e., once the FSTV is initialized, values of these variables will stay constant during run time.

The controlled variables are not used in any module. By observing this rule, the FSTV modules are made re-entrant, thereby making it a relatively simple task to generate the multi-threaded version of the FSTV.

In addition, the architectural requirement that there be no shared data across levels necessarily limits the scope of the variables to being within the modules of the same level only. External variables known to modules across levels should not be used. A simple rule to follow therefore is to avoid using external variables. In the current implementation, use of the external variables are associated with the initialization variables mentioned in the previous paragraph, and none are known across levels. Appendix 2 contains a list of all external and/or static variables used in the current version.

3.1.2 INTER-LEVEL COMMUNICATION

The hierarchical structure of INFOPLEX requires that inter-level procedure calls go through gateway controllers and the global bus. In simulating this architecture, the Control Structure Subsystem of the STV assumes that the functional modules observe the inter-level procedure call convention.

Inter-level communications in the FSTV are in the form of message passing between adjacent levels. Direct communication between unadjacent levels is not used.

Message passing vs. subroutine call. Each level of INFOPLEX contains a number of functional modules. In the STV, they are represented by program modules. Normally, when a module encounters a sub-task which is to be processed by another module, it uses a subroutine to activate the latter. However, since the FSTV is constructed for use in a multi-level multi-processor system, a number of pre-defined tasks are not invoked directly, but are invoked through message passing, a facility which is provided by the Control Structure. In fact, all cross-level invocations are done through message passing. This conforms to the requirement of the architecture as well as to the need of informing the Control Structure in the event of an inter-level call, so that the Control Structure can emulate the load on the gateway controller and the global bus.

The Control Structure provides a set of message passing protocols (including message passing among modules in the same level.) (Refer to chapter 5 and 7 of <To81>.) When a module at a higher level in the Functional Hierarchy needs to invoke a module at a lower level, it necessarily makes use of these protocols in place of a direct subroutine call. Likewise, when the module at the lower level completes the task, it makes use of these protocols to send the results back to the invoking module.

To this end, all modules in the FSTV are divided into two groups: the Top-Level Procedures (T-proc's) and the Subroutine Procedures (S-proc's). The T-proc's are always invoked by message passing, and the S-proc's by direct subroutine calls. Each level is consisted of a set of T-proc's which can be invoked by the upper level, and a set of S-proc's whose names are unknown to any other levels, and are not directly invocable by the upper level. It is noted that to invoke a T-proc at the same level a process will have to send a message to it rather than to use a traditional subroutine call. Therefore the convention for inter-level procedure call is more properly termed as the convention for inter-T-proc call.

Since T-proc's represent entry points to a level, special attention is given to its invocation. The arguments needed to invoke a T-proc are grouped into a set of Control Blocks. A control block is a based variable which contains a pre-defined data structure. The set of control blocks associated with a T-proc includes both the calling control blocks and the return control blocks. When the T-proc is invoked, it is passed a link list of calling control blocks; when it returns, it puts the return messages and values in a link list of return control blocks. The names of the T-proc's at a level and the control blocks associated with them collectively form the interface of the level seen by the upper level. This mechanism satisfies the requirement that "...each level of the functional hierarchy interacts with other levels through a clean set of interfaces"

<Hsu80>. Messages passed between levels are, therefore, link lists of control blocks. Details of control block formatting conventions are described in the following.

Control Block Format. According to the hardware architecture of INFOPLEX which requires that no shared memory is used between levels, when a message is to be passed to a process at another level the message is to be physically moved from the local memory of the calling level to that of the called level. In simulating this activity, the Control Structure has to be given a pointer to the message and the length of the message. While it is possible to give the Control Structure a message in a contiguous block, it is more desirable from the programmers' point of view to be able to give the Control Structure a link list and the necessary information for the Control Structure to collect all data in the link list. To this purpose, a convention for interacting with the Control Structure is devised: every element (a Control Block) of the link list contains a CNTL_INFO as the first field of the data structure as follows:

```
2 CNTL_INFO,  
  3 LEN FIXED BIN (15),  
  3 CBTP FIXED BIN (15),  
  3 PTR PTR,  
2 DATA,  
  .....
```

where LEN describes the length of the data portion of the control block, CBTP is a unique number assigned to this type of control block, and PTR points to the next control block in the link list. With the information in CNTL_INFO the Control Structure will be able to collect all elements and move them to the other level. Similary, the called process simply follows the pointers to obtain all arguments it needs. The data structure of the DATA portion of a control block list represents the agreed upon interface between sending and receiving processes. This convention applies to both argument passing and return data. Examples of Control Blocks are given in Appendix 2.

Service routines and programming conventions. In order to insulate the FSTV modules from future changes to the Control Structure-provided communication protocols, a set of service routines are used to centralize the use of these protocols. These are SVCS1 and SVCS2. In addition, since the current version of FSTV does not make use of intra-request parallelism, another set of service routines are devised on top of SVCS1 and SVCS2 in such a way that, if these routines are used to invoke protocols provided by the Control Structure, the existence of the Control Structure could be completely shielded from the FSTV. Therefore, for example, it is possible to run FSTV alone without the Control Structure, so long as the second set of service routines is replaced by a dummy set. This feature enables independent debugging and testing of the FSTV from the Control Structure, thereby reducing the complexity of the software

development process. The next chapter contains a description of these service routines, their interfaces and how to use them.

3.2 FUNCTIONAL DECOMPOSITION

The database management functions in the current version of the FSTV have been hierarchically decomposed into four levels. Each level communicates with only the adjacent level. The following figure illustrates the relationship between the current implementation and the target implementation as outlined in <Hsu80>. While the current implementation is a simplified version of the target system, it illustrates the process of reducing a reasonably complicated user query to accesses/updates in a linear address space.

Level-1

VIEW AUTHOR-
IZATION
LEVEL

USER INTERFACE /SECURITY
PROCESSING GROUP

VIEW TRANS-
LATION
LEVEL

VIEW ENFORCE-
MENT
LEVEL

VALIDITY
CHECKING
LEVEL

Level-2

VIRTUAL
INFORMATION
LEVEL

BASE DATA MODEL
PROCESSING GROUP

N-ARY ENTITY
LEVEL

N-ARY/B-ARY
LEVEL

UNARY SET
LEVEL

Level-3

DATA ENCODING
LEVEL

INTERNAL DATA MODEL
PROCESSING GROUP

Level-4

MEMORY MGMT
LEVEL

STORAGE HIERARCHY

One important 'feature' of the design is to be stressed. To eliminate duplication of data storage and its management functions, system databases (e.g., catalogues) used at a level are stored and manipulated as regular data at the next level. That is, all the catalogues used at a level are defined as data sets at the next lower level during the initialization process. This means that the local memory at a level is strictly used for storing the context of active processes, and that all the structuring information such as catalogues and indices as well as the database itself is stored in the virtual memory (therefore in the Storage Hierarchy) managed by the Memory Management Level.

The four levels in the current implementation are briefly described as follows. Details of the functional modules are the subject of chapter 5.

1. User Interface Level (Level 1): This level interacts with the user directly. It accepts users' commands to define and manipulate the databases. It parses input character strings, identifies the meaning of the input tokens, and generate proper requests to the next level for validation and execution of the input commands. For ease of use, the data definition portion of the user interface makes use of the panel facility, which enables the DBA to define the database by filling up blanks in a pre-defined panel. This level does not create or make use of any system databases.

2. Database Semantics Level (Level 2, or The Entity Level): This level manages the entity/attribute catalogues, validates database insertions and updates, and transforms the high level database definition and manipulation requests into commands to operate on the data structures of the internal schema. The system database at this level, the entity/attribute catalogues, is itself defined in terms of entities and attributes, and is stored as regular database data in the next level.

3. Internal Schema Level (Level 3, or The N-ary Level): This level manages the Pset/Bset catalogues and performs retrieval and update of the content of the pre-defined Psets and Bsets. It accepts a tree-structured database command and decides upon the P-elements to be retrieved or affected and executes the command. It ultimately implements information in the database in the form of BEU's (Basic Encoding Units); a BEU is given to the next lower level to be stored as a unit of bit string. The internal schema implemented in the current version is very simple: it implements P-elements within the same Pset as an unsorted bi-directional link list, and binary associations among the P-elements through uni-directional linkage pointers.

4. Memory Management Level (Level 4): This level manages a linear virtual address space which is initialized by the FSTV Console Program as a based area of a certain size.

Store, retrieval and update of BEU's are carried out at this level as operations on this based area.

4.0 FSTV INTER-LEVEL COMMUNICATION SERVICE ROUTINES

This chapter documents the set of service routines used by FSTV functional programs to interface with, and to insulate from, the Control Structure-provided message passing protocols. The first section describes the two service routines, SVCS1 and SVCS2, which are used by FSTV functional modules to insulate themselves from directly using the SEND protocol provided by the Control Structure. The second section describes another set of service routines, TCALL, TBEG and TRTN, which are used to insulate the FSTV completely from the Control Structure.

4.1 INSULATING FROM SEND PROTOCOL -- USE OF SVCS1 AND SVCS2

Convention for the calling procedure. When a process wishes to invoke a T-proc, it must send a message to the operating system of the target level. The message will contain the name of the procedure to be invoked and a pointer to a message to the invoked procedure. The latter message contains the return address (e.g., the mail box number) and calling arguments if any.

The calling process will prepare the calling arguments as a link list of control blocks. (The meaning and the format of a control block has been described in the previous chapter.) This link list is called a calling control block link list. The calling control block link list will then be prefixed with a special control block, the MSG control block. The purpose of this MSG control block is to inform the invoked procedure of the process id (i.e., the process address) of the calling process, so that return messages shall be routed correctly. The format of this MSG control block is exemplified by the following declaration:

```

DCL 1 MSG BASED (P),
    2 CNTL_INFO,
        3 LEN FIXED BIN (15) INIT(6),
            /* LENGTH OF DATA PORTION */
        3 DUM_CBTP FIXED BIN, /* NOT USED */
        3 PTR PTR,
            /* POINT TO FIRST CONTRO BLOCK IN CALLING
            ARGUMENT LINK LIST */
    2 DATA,
        3 PROC_ADDR,
            /* ADDRESS OF THE PROCESS INITIATING
            THIS MESSAGE */
        (4 LEVEL,
        4 VPID,
        4 BOXID) FIXED BIN (15);

```

The entire message link list is sent by calling SVCS1 service routine. SVCS1 takes 2 arguments: the name of the procedure to be invoked (CHAR(8)) and a pointer to the message link list (PTR). It does not care about the content of the message link list. SVCS1 returns a return code (FIXED BIN(15)):

```
CALL SVCS1 (PROC_NAME,PTR,RTN_CODE);
```

After the call, the calling process may continue until the point at which a return message from the call is needed. It then calls the WAIT protocol provided by the Control Structure, specifying the mailbox it waits on:

```
CALL WAIT (BOXID);
```

after which it may obtain the return message by moving the relevant data pointed to by VP.WAIT.MSG (set by the Control Structure) to the appropriate variables.

A calling process may send multiple messages at the same time, thereby achieving the benefit of intra-request parallelism.

Convention for the called procedure. All T-proc processes will have to respond to the message passing protocols set up by the Control Structure. When it is created, it must first locate the message link list passed to it from the caller in the base variable pointed to by VP.WAIT.MSG. Before it finishes, it must

have sent the return message, if any, to the calling process. The format of the message passed to it is as described in the previous subsection. To send a return message to an existing process, however, SVCS2 is used. SVCS2 also takes 2 arguments: the address of the receiving process (LEVEL,VPID,BOXID) and a pointer to the return message link list: (The return message link list is constructed in a way similar to that of the calling message; i.e., a return control block link list prefixed with a MSG control block.)

```
CALL SVCS2 (PROC_ADDR,P,RTN_CODE);
```

Note that it is possible for 2 processes to engage in a conversation, in the sense that the calling process may send a second message to the called process after the calling process has already obtained some return message from it. That is to say, a called process needs not "finish" itself immediately after sending the return message.

Summary. The functional program modules accomplish inter-T-procedure calls through SVCS1 and SVCS2 service routines. SVCS1 is used to invoke a new process, while SVCS2 is used to send (return) messages to a known process. Both SVCS1 and SVCS2 take two arguments: the name of the procedure to be invoked or the address of the existing process the message should go to, and a pointer to the message link list.

Text inclusion. A module which wishes to use SVCS1 and SVCS2 should include the macro SVCS. Declarations of MSG, PROC_ADDR, PROC_NAME are provided for in the macro.

4.2 INSULATING FROM CONTROL STRUCTURE -- ADDITIONAL SERVICE ROUTINES

To avoid direct use of WAIT and FINISH protocols and VP.WAIT.MSG provided by the Control Structure as well as SVCS1 and SVCS2, a functional module may use a set of 3 service routines to accomplish inter-T-proc communication: TBEG, TRTN and TCALL.

Convention for the called T-proc. When a T-proc process is first invoked, it calls TBEG to obtain the return address (i.e., the process address of the calling process.) The call takes the following form:

```
CALL TBEG. (PROC_ADDR,P);
```

where PROC_ADDR will be loaded with the identifier of the calling process, and P points to the first control block in the argument link list.

Note that the MSG control block described in the previous section is stripped off by TBEG and no longer concerns the functional module. When a T-proc returns, it calls TRTN to pass the return control block link list to the calling process. The call takes the following form:

```
CALL TRTN (PROC_ADDR,P);
```

where PROC_ADDR is as described above and P points to the first control block in the return control block link list. Note also that the user of TRTN no longer needs to prefix its return link list with the MSG control block, since the latter will be added to by the TRTN routine.

Convention for the calling T-proc. When a module wishes to invoke T-proc, it formats the arguments in the form of a control block link list, and then issues the following call:

```
CALL TCALL (PROC_NAME,BOXID,ARG_PTR,RTN_PTR);
```

where PROC_NAME is the name of the T-proc to be invoked, BOXID is the mailbox number where the caller wishes the return messages to be stored, ARG_PTR points to the first control block in the argument control link list, and RTN_PTR (a return value) will point to the first control block of the return control block link list.

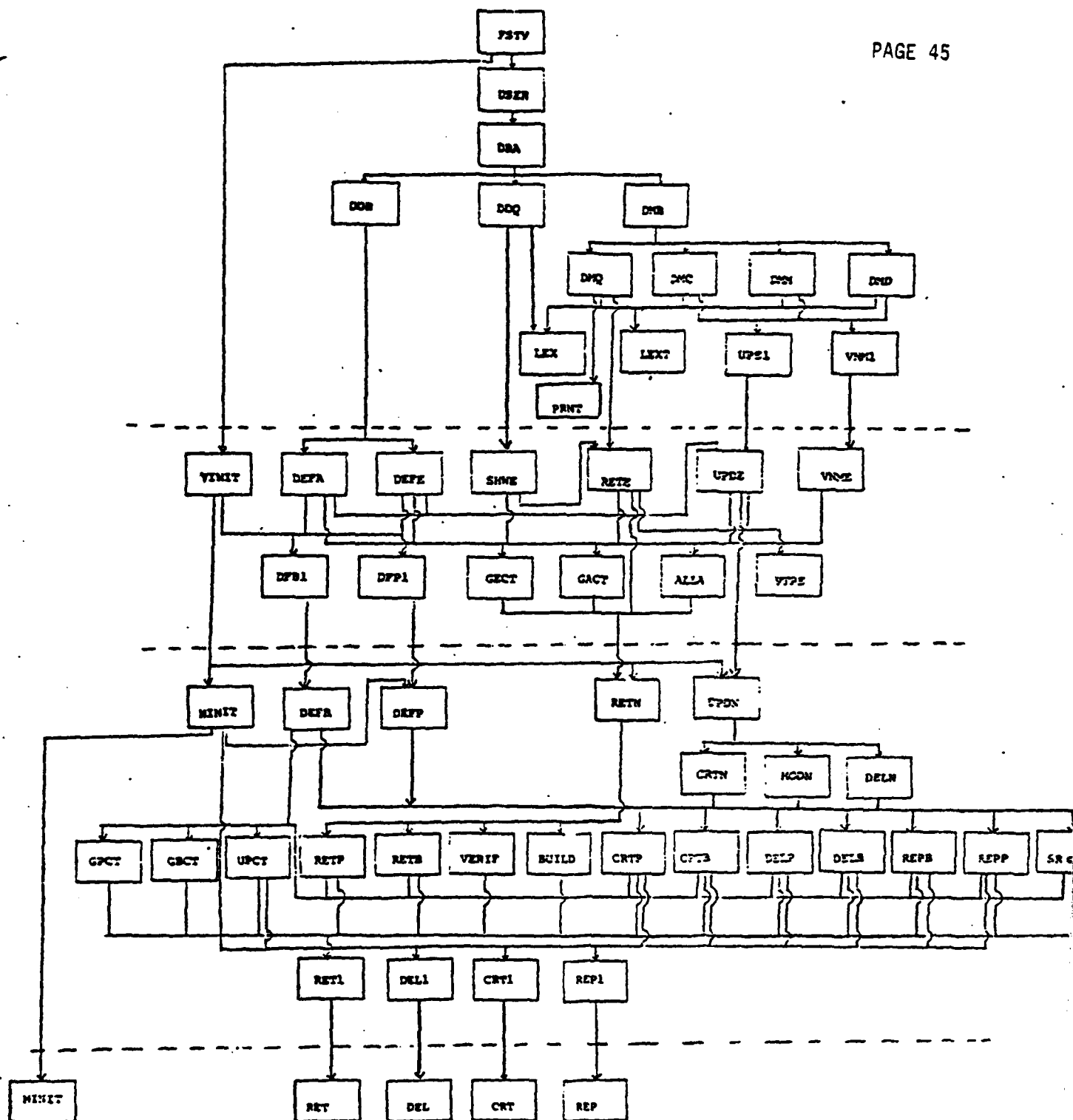
NOTES:

1. This set of service routines can not be used to engage two communicating processes in a "conversation mode" (refer to section 2), nor can it be used when there is intra-request parallelism involved.

2. To use this set of routines, the SERVICE macro must be included.

5.0 OVERVIEW OF FSTV FUNCTIONAL PROGRAM MODULES

This chapter contains a description of the logical function and the data structure of program modules in the current version of FSTV. The description follows the four hierarchical levels employed in FSTV, starting from the bottom level. For each level, we will discuss the data model (i.e. the data structure of objects processed at the level) and the system databases used, and provide an overview of the data definition, data manipulation, and level initialization modules that make up the level. The reader will notice that modules at a level are classified into T-proc's and S-proc's as discussed in the previous chapters, and that conventions and constraints mentioned previously are observed. In addition, the reader will see how information abstraction proceeds from the bottom level which recognizes only a linear virtual address space used as the database store to the database semantics level which processes high level constructs such as entities and attributes. A complete listing of these program modules is included in Appendix 3. The following figure contains a diagram showing all modules in the current implementation.



System Initialization.

Functional Hierarchy initialization consists of activities which render the functional modules in a runnable condition. Presently an assumption is made of the initial condition of the system when it is 'powered on': each level will already have all the functional modules loaded into its local memory. This may not be the case in reality. A bootstrap procedure which loads the minimum set of the functional modules (the core set) from the Storage Hierarchy may have to be added to each level. However, for the purpose of this first version, we may assume that this type of bootstrap is taken care of by the local operating system, which is simulated as part of the Control Structure and is out of the scope of FSTV.

Initialization of catalogues is also necessary before the system can respond to normal processing. For a system without an existing database (i.e., an INFOPLEX just bought off the shelf), the initialization module at a level will define the catalogue structures used at the level in terms of data sets implemented at the next lower level and establish a set of 'key' values which will be used at this level when normal processing starts. This means that initialization ought to start from the bottom level. To initialize a system which already has a database, the catalogue definition activities are omitted, and the initialization module simply restores the key values. The

individual initialization modules will be discussed further in the following subsections.

5.1 MEMORY MANAGEMENT LEVEL

The function of this level is to manage the virtual memory address space that Functional Hierarchy is provided for by Storage Hierarchy and to interface with Storage Hierarchy.

5.1.1 DATA MODEL AT THE MEMORY MANAGEMENT LEVEL

The basic element stored in SH is the Storage Encoding Unit (SEU). Each stored SEU is assigned a SEU id, which is equivalent to the virtual address of this stored SEU in the Storage Hierarchy. An SEU has the following format:

```
DCL 1 SEU BASED (SEUPTR)
  /* 2 MOVE_PTR BIT(32), NOT USED CURRENTLY */
  2 N FIXED BIN,
  2 INVAL CHAR(1) INIT('N'),
  2 DATA CHAR(I REFER (SEU.N));
```

where MOVE_PTR is an id in the SH which is always null unless the SEU has been moved, in which case MOVE_PTR contains the new location in SH, INVALID is a flag which, when set, signifies that this SEU has been deleted, and finally DATA is a character string with a variable length indicated by N and contains the data of this SEU.

The format of SEU is known to modules within this level only. The level above will pass to this level a bit string of data to be stored as a unit in return for an id, and retrieve this bit string back by presenting to this level the id. The Memory Management Level will allocate a SEU for each unit of data to be stored, and put the data in the DATA field of the SEU. It then finds a free area in the virtual memory and passes the SEU to the Storage Hierarchy.

5.1.2 SYSTEM DATABASES AND DATA DEFINITION MODULES AT THE MEMORY MANAGEMENT LEVEL

Presently a very simple memory management function is provided. Storage in the virtual memory is assigned incrementally starting from address 16. Therefore the only "system database" needed is a counter indicating the next

available virtual address. This counter is stored in address 0 of the Storage Hierarchy.

No data dictionary is used in the current version. There is also no data definition necessary.

5.1.3 DATA MANIPULATION MODULES AT THE MM LEVEL

There are four T-proc modules at this level to perform creation, deletion, modification and retrieval of data elements. Each of these functions is performed by one module.

CRT. This module responds to requests from the upper level to store data elements. Its calling control block contains the data and the length, while the return control block contains an ID, which is the bit representation of the virtual address assigned to the SEU encompassing the data. This module calls subroutine GMEM to get the virtual address, and subroutine PSEU to decompose the SEU into 8-byte packets and to store them into the Storage Hierarchy.

RET. This module processes requests to retrieve a data elements given its id. It calls subroutine GSEU to retrieve the

SEU of the data element from SH using the id and extracts the DATA portion of the SEU to return to the upper level.

MOD. This module processes requests to modify content of a data element given its id and new data. Like RET, it calls GSEU to retrieve the SEU of the data element from SH and replaces the DATA field of the SEU with the new data, and calls PSEU to put it back into SH.

DEL. This module processes requests to delete a data element given its id. It calls GSEU to retrieve the SEU, sets its INVAL field, and calls PSEU to put it back. No garbage collection is performed in the present implementation.

5.1.4 INITIALIZATION AT THE MEMORY MANAGEMENT LEVEL

MINIT. This module performs all the necessary steps to initialize this level during IPL. The calling control block contains the method of initialization ('file' or 'new'). If it initializes from a Storage Hierarchy that doesn't contain any data yet (i.e., a 'new' initialization), it bootstrap a value '16' into the first word of the Storage Hierarchy, which is to be used as a register for the next available address. If it initializes from an existing database, it retrieves a one-word

'key' for the upper level from a fixed location in SH and includes it in the return control block. (The upper level will use this key to 'recover' its system databases.) This module also provides an interface (i.e., when the OP field of the calling control block contains 'S') for the upper level to store a key.

5.1.5 SUBROUTINES AT THE MEMORY MANAGEMENT LEVEL

There are 4 subroutines at this level. GMEM (Get MEMory) returns a virtual address of a free area in SH given the length of the area requested. PSEU (Put SEU) decomposes a given SEU into 8-byte packets and stores them consecutively into a given area in SH. GSEU (Get SEU) performs the reverse of the task of PSEU: given an address of an area, reconstruct the SEU contained in that area by retrieving a packet at a time from SH. And, finally, PAKT (PAcKeTing) is a subroutine called by the above 3 subroutines to get a packet of data from SH given its byte address. PAKT is the only routine in the FSTV which is aware of the interface to the top level of SH. The interface is declared as a data structure ARG.

5.2 INTERNAL SCHEMA LEVEL

This level provides various 'stored structures' to store the database. It accepts definitions of data element sets and their relationships, and performs search, retrieval and update of these data.

5.2.1 DATA STRUCTURES AT THE INTERNAL SCHEMA LEVEL

This level envisions that the world is composed of a network of some basic data elements called Primitive elements (P-elements.) P-elements and their relationships are implemented at this level in the form of a Basic Encoding Unit (BEU). The reader is referred to section 2.5 for a description of the BEU concept employed in this implementation. A BEU is declared as follows:

```
/*DCL OF BASIC ENCODING UNIT (BEU) USED IN THE N-ARY LEVEL***/  
DCL 1 BEU BASED(P),  
    2 NUMPTR FIXED BIN,  
    2 ID_ARRAY(I REFER (BEU.NUMPTR))BIT(32) INIT ((I)UNSPEC(NULL)),  
    2 DLEN FIXED BIN,  
    2 DATA BIT(J REFER (BEU.DLEN));
```

where ID_ARRAY is an array of ID's of related BEUs'. NUMPTR indicates the number of slots its ID_ARRAY contains, DATA is the bit string of the data element, and DLEN indicates the length of the data in terms of number of bits.

The format of the BEU is known only to this level. The upper level is not aware of how PSETS and their inter-relationships are stored.

5.2.2 SYSTEM DATABASES AND DATA DEFINITION AT INTERNAL SCHEMA LEVEL

Definition of Psets and Bsets are kept in two catalogues: the Primitive Set Catalogue (PCAT) and the Binary Association Set Catalogue (BCAT). Each entry in PCAT describes a primitive set. Its format is described by the following declaration:

```
/*DCLS OF PSET CATALOGUE AT THE N-ARY LEVEL*****/
DCL 1 PINFO,
    3 NUMPTR FIXED BIN,
    3 PLEN /*IN NUMBER OF BYTES*/ FIXED BIN,
    3 PTYPE CHAR (1), /*'N' OR 'X' OR 'B' OR 'C'*/
    3 LTYPE BIT (8), /*'00000001' IS LINK LIST*/
```

```

3 L_ID BIT (32), /*ID OF THE FIRST BEU IN THIS SET*/
3 L_POS FIXED BIN , /*POS IN ID_ARRAY USED FOR LINEAR CHAINING
                      OF BEU'S WITHIN THIS SET*/
3 L_POS2 FIXED BIN,
3 MAP BIT (32) ; /*UP TO 16 PTRS ARE ALLOWED FOR A BEU*/

```

where NUMPTR describes the size of the ID_ARRAY of the BEU's in this Pset; PLEN is the length of the data element; PTYPE describes the data type of this primitive set, which can be numeric, bit or character string; PTYPE 'X' is a special type which is used when the data value of the Pset in question is insignificant (in this case the Pset is mainly used as an anchor for binary associations); L_POS and L_POS2 are the ID_ARRAY slots used for forward and backward chaining of all BEU's in the Pset; MAP is used to record assignment of ID_ARRAY slots: if the n-th slot is assigned, the n-th bit in MAP will be set; LTYPE describes how BEU's in this Pset are grouped together: currently only the bi-directional link list is in use. Among these parameters, NUMPTR, PLEN and PTYPE are provided by the upper level (refer to the description of DEFP below), while the rest are internally generated.

Each entry in BCAT describes a binary association set. Its format is described by the declaration of BCAT:

```

/*DCL OF BCAT TEMPLATE*/
DCL 1 BINFO,

```

```
2 PSETID(2) BIT (32),  
2 POS FIXED BIN, /*POS OF POINTER ARRAY OF PSETID1 USED */  
2 FUNC CHAR (1); /*'S' OR 'M'*/;
```

where PSETID(1) and PSETID(2) contain the ID's of the source Pset and the target Pset involved in this binary association (note that all binary associations are directional); POS contains the ID_ARRAY slot number used by BEU's in the source Pset to record the ID's of the associated BEU's in the target Pset; FUNC indicates the type of relationship between the source and the target: one-to-one ('S') or many-to-one ('M').

Both PCAT and BCAT are themselves implemented as Psets. Therefore each catalogue entry is implemented as a BEU. These two Psets are defined during system initialization (refer to section 5.2.4), creating the very first two PCAT entries of the system. This is exemplified by the following figure.

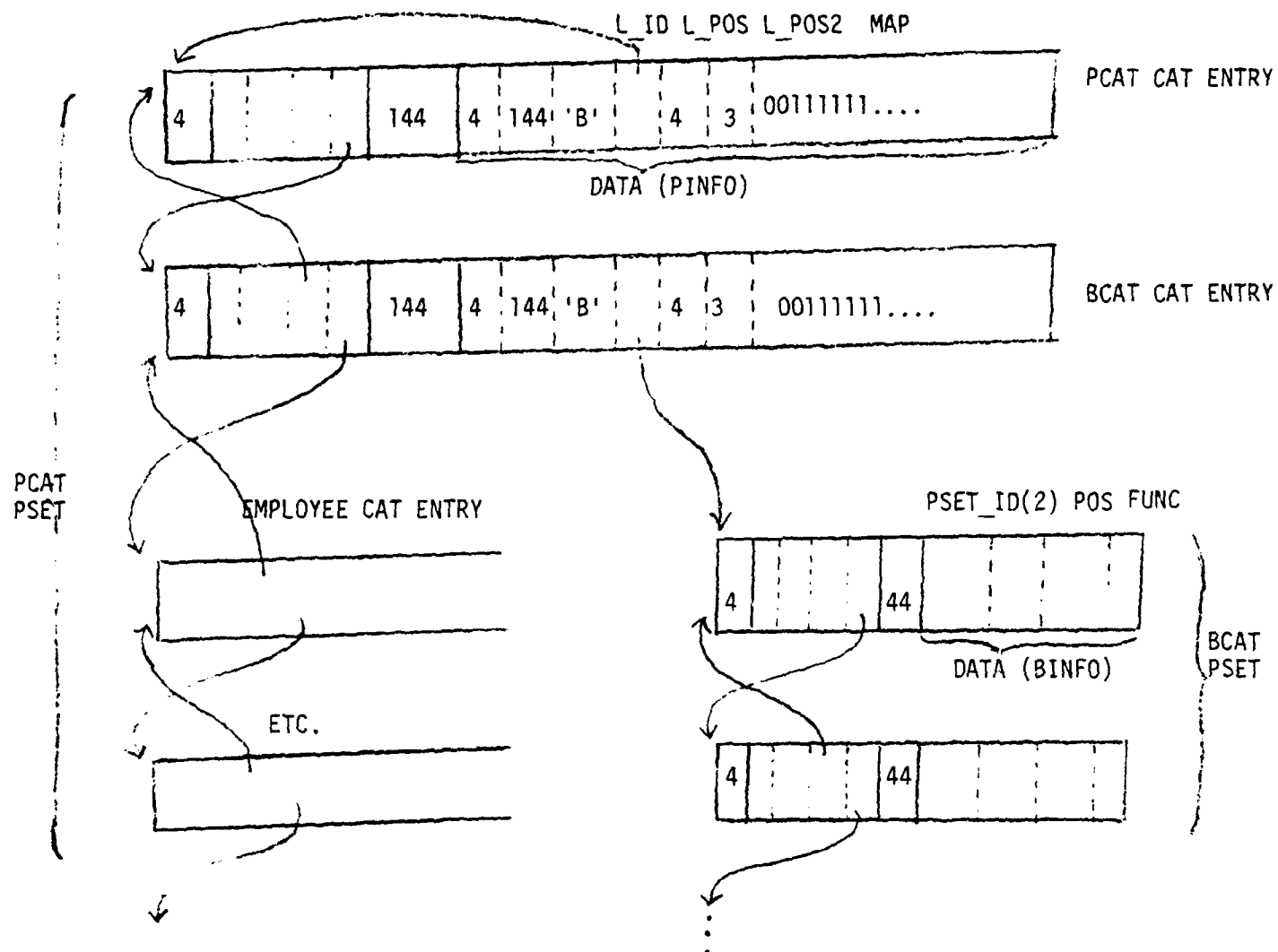


Figure: Structure of the Catalogue Implementation at the Internal Schema Level (N-ary Level)

To remember the ID's of the two BEU's that implement these two catalogue entries, two static variables are given values to during initialization: PCATID and BCATID. Their values will remain constant for the rest of the system processing. Procedures for defining Psets and Bsets will reference these two variables in order to insert new entries into the catalogue entry chain.

Two T-proc's form the data definition interface of this level. They are described below:

DEFP. This module (DEFine Pset) responds requests for primitive set definition. The calling control block contains the length and the type of the data field, and the size of the ID_ARRAY. (It could be argued that the last item shouldn't be passed since it relates to the internal, stored format of primitive elements which is theoretically unknown to its definer. Presently, however, it is included in the interface merely because a separate interface for the DBA to express his opinion on this matter is lacking. This issue will be further discussed.) The return control block contains the ID of this Pset, which is (but the upper level shouldn't care) the ID that the Memory Management Level assigned to the catalogue entry BEU of this Pset.

DEFB. This module (DEFine Bset) responds to requests for binary association set definition. The calling control block

contains the Pset ID's of the source and the target Psets involved and its relationship type (i.e., one-to-one, many-to-one). The return control block contains the ID assigned to this Bset, which is equivalent to the ID that the Memory Management Level assigned to the catalogue entry BEU of this Bset.

5.2.3 DATA MANIPULATION AT THE INTERNAL SCHEMA LEVEL

This section describes the two T-proc's that form the data manipulation interface of this level: RETN and UPDN.

RETN. This module (RETrieval at iNternal schema level) accepts a reasonably complicated query expressed in a tree form in which nodes are Psets and arcs are Bsets. An arbitrary number of nodes can be included in the retrieval tree. As an example, a query of the form

```
RETRIEVE EMP (NAME, DEPT(DNAME), PROJ(NAME)) WHERE (AGE>50 AND
DEPT(LOC) = 'New York')
```

can be expressed in a RETN tree shown in the following figure.

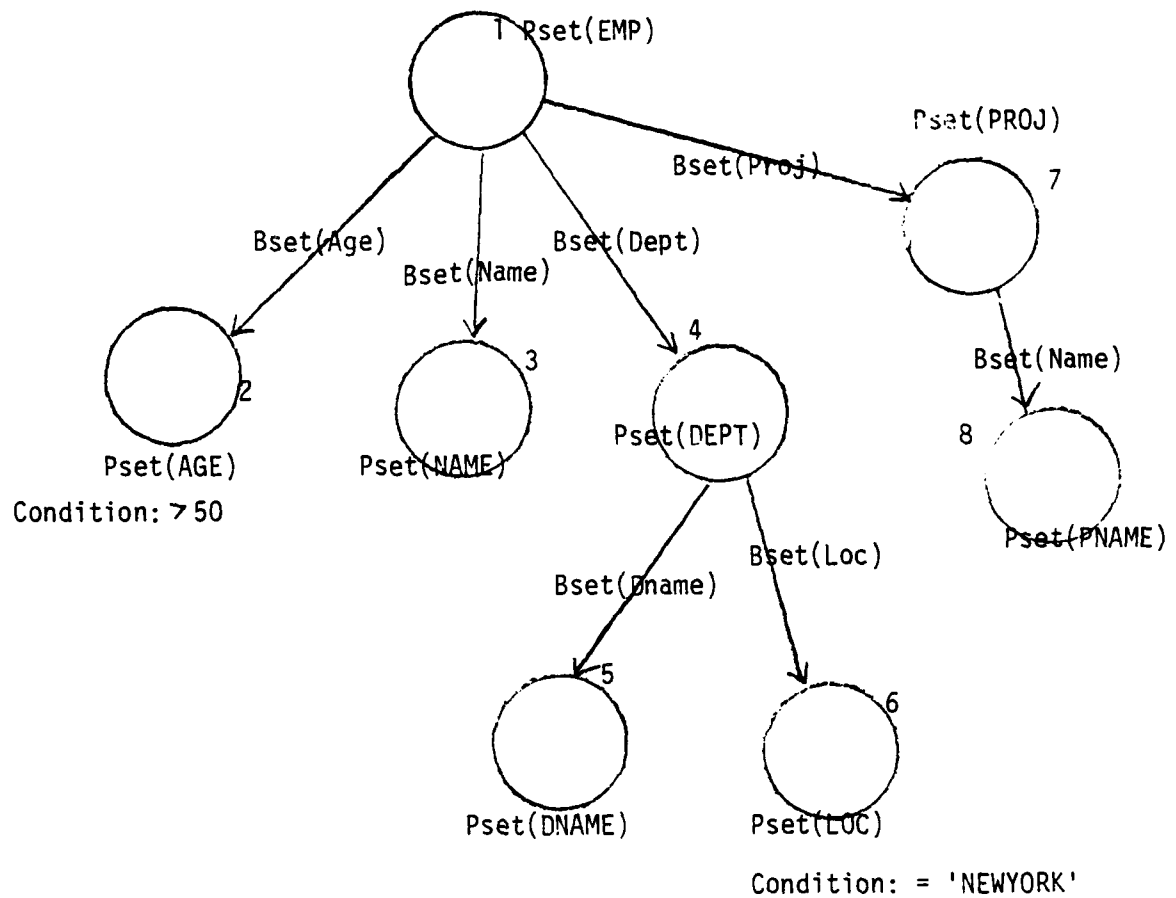


Figure: The RETN Tree of an Example Query

The caller of RETN passes a link list of RETN calling control blocks, each of the control blocks describing a node in the tree. The declaration of this control block is:

```

DCL 1 RETN_ARG    BASED(P),
      2 LEN      FIXED BIN (15) INIT (61),
      2 CBTP FIXED BIN (15) INIT( 25 ),
      2 PTR PTR INIT (NULL),
      2 NODE FIXED BIN (15) INIT(1), /*NODE NUMBER*/
      2 PSETID BIT(32), /*PSETID OF THE NODE*/
      2 PARENT FIXED BIN (15) INIT(0), /*PARENT NODE NUMBER*/
      2 BSETID BIT(32), /*BSETID OF THE PARENT PSET*/
      2 GET BIT(8) INIT ('00000000'B), /*SEE RETNGET FOR DETAIL*/
      2 N FIXED BIN INIT(0), /*NUMBER OF RELEVANT 'OR' PREDICATES*/
      2 PRED (1), /*IN CURRENT VERSION ONLY ONE PREDICATE EACH TIME
                    IS ALLOWED; FUTURE VERSION MAY USE REFER OPTION TO N*/
      3 CN FIXED BIN (15), /*CONDITION NUMBER*/
      3 SETOP BIT(8), /*RELEVANT IF BSET IF MULTI-TARGET;
                    NOT USED IN CURRENT VERSION*/
      3 OP BIT(8), /*COMPARISON OPS, SEE RETNOP FOR DETAIL*/
      3 DLEN FIXED BIN (15) INIT(0), /* LEN OF COMPARE DATA IN BYTES*/
      3 DATA BIT(320); /*40 BYTES MAXIMUM*/

```

where NODE is the node number of the node being described, PARENT is the node number of the predecessor, BSETID is the ID of the Bset which leads the parent node to this node (i.e., the 'arc'), GET indicates whether all, any or none of the qualified

occurrences of this Pset are to be retrieved, and PRED describes restriction of values of this node (i.e., a predicate). N, CN and SETOP are currently unused.

RETN returns occurrences of the tree. The return data is a link list of the RETN return control blocks. Every control block in the link list describes one of the primitive data elements retrieved. The format of the return control block is:

```

DCL 1  RETN_RTN BASED(P),
      2 LEN      FIXED BIN (15) INIT (4),
      2 CBTP FIXED BIN (15) INIT( 26 ),
      2 PTR PTR INIT (NULL),
      2 RTN_CODE FIXED BIN INIT (0),
      2 N FIXED BIN (15); /*NUMBER OF INSTANCES OF ROOT NODE RETURNED*/
DCL 1  RETN_RTN1 BASED(P),
      2 LEN      FIXED BIN (15) INIT (46),
      2 CBTP FIXED BIN (15) INIT( 27 ),
      2 PTR PTR INIT (NULL),
      2 NODE FIXED BIN (15), /*NODE NUMBER*/
      2 INDX FIXED BIN (15), /*IF MULTI OCCUR*/
      2 DLEN FIXED BIN, /*LEN OF DATA IN BYTES*/
      2 DATA BIT(320); /*DATA RETURNED*/

```

As an example, suppose two employees are found to satisfy the previous query, the link list representing the return will look like that in the following figure.

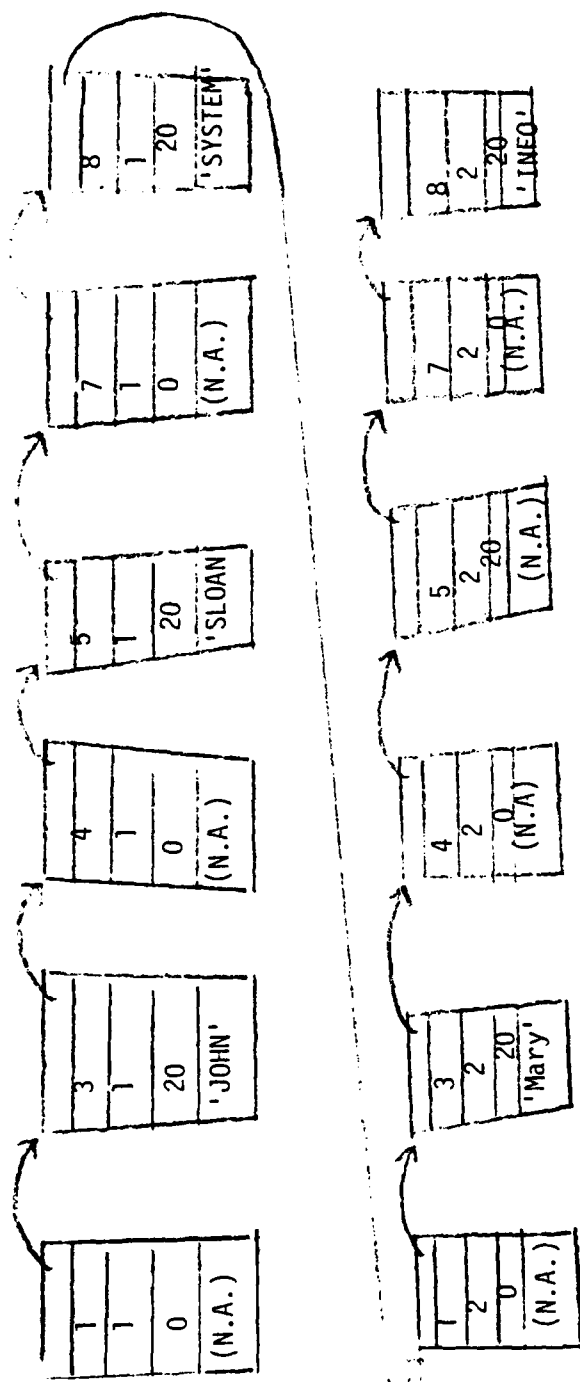


Figure: Example RETN Return Control Block Link list (Based on the RETN Request shown in the Previous Figure)

This module, given the RETN request, uses a very simple method to traverse the database. It first visits the root node and obtains the first occurrence of the root node. It then verifies the occurrence of the rest of the tree against the predicate (if any). If the occurrence is found to be valid, a subroutine BUILD is called to generate the data and chain them to the return link list. If the occurrence is not valid, then it discards the current occurrence and gets the next occurrence of the root node. It continues this procedure until all occurrences in the root node are examined, or when one valid occurrence is found if the GET field of the root node in the request tree indicated 'ANY'. The way it verifies an occurrence is to follow the RETN tree to establish occurrences of its associated elements and to verify their content against their predicates. If any mismatch is found, the current occurrence is considered invalid and a new root occurrence started.

This module uses several subroutines to complete its procedure. RETP is called to retrieve the ID of the next P-element of a Pset on the forward chain of the Pset; RETB retrieves the ID of the associated P-element in a certain Bset of a P-element whose ID is given; VERIF verifies the value of a P-element against a predicate; and BUILD builds the RETN return control block link list based on a data structure CURSOR, which contains the ID's of P-elements of one occurrence of the tree. In addition, it calls GPCT and GBCT to obtain catalogue entries of relevant Psets and Bsets.

UPDN. This module (UPDate at iNternal schema level) creates, deletes and modifies data elements and their relationship in the database. Like RETN, the update argument is expressed in a tree called an UPDN request tree. The UPDN tree is slightly different from RETN tree in that the operations are different. However, the general format, i.e., a link list of control blocks each describing a node in the tree, is the same.

Update operations in a database ultimately include creating, deleting and modifying primitive data elements and their relationship to others. To alleviate the caller from the burden of creating individual P-elements one by one and then creating linkages among them one by one, the tree form enables it to express a meaningful aggregate of them in one UPDN request. For example, a database update request 'Remove employee John from Project A' can be translated by the levels above this level into a UPDN tree shown in the following figure.

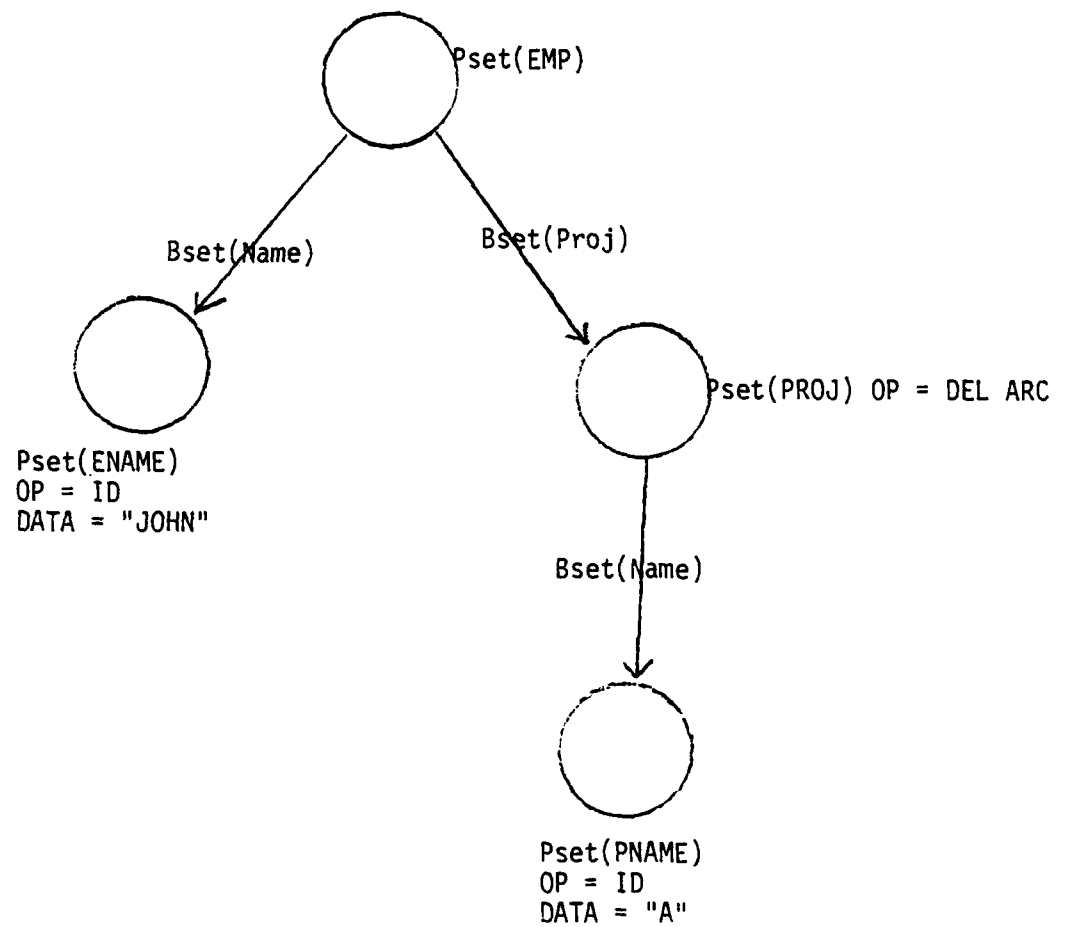


Figure: Example of a UPDN Tree

As shown in the figure, each non-root node carries with it an identifier for the arc (i.e., the Bset ID), an operation (i.e., OP) and possibly a data (i.e., ='John'). The possible operations are Insert arc, Delete arc, Replace arc, Create P-element and then insert arc, Delete arc and also the P-element, Replace value of P-element, and Identity. The identify operator is used when the node does not require any update but is included in the tree for the purpose of identifying the occurrence of its predecessor nodes. The root node operations are Delete, Create and Modify. (These operations are declared in the data structure UPDNOP.) In our example, the operation involved is to 'de-associate' the P-element representing employee John from the P-element representing Project A. Therefore an arc between these two P-elements must be deleted. However, to 'pin point' the occurrence of these two P-elements, two ID nodes are used (i.e., ENAME='John' and PNAME='A'.)

The UPDN request is expressed as a link list of control blocks UPDN_ARG. Declaration of UPDN_ARG is as follows:

```
DCL 1 UPDN_ARG    BASED(P),
      2 LEN      FIXED BIN (15) INIT (56),
      2 CBTP     FIXED BIN (15) INIT( 30 ),
      2 PTR     PTR INIT (NULL),
      2 NODE     FIXED BIN (15) INIT(1), /*NODE NUMBER OF UPDN TREE*/
      2 PSETID   BIT (32), /*PSETID OF THE NODE*/
      2 PARENT   FIXED BIN (15) INIT(0), /*PARENT NODE NUMBER*/
```

```

2 BSETID BIT (32), /*BSETID OF PARENT PSET LEADING HERE*/
2 OP FIXED BIN (15), /*OP CODE FOR UPDN; SEE UPDNOP FOR DETAIL*/
2 DLEN FIXED BIN (15) INIT(0), /*IN NUMBER OF BYTES*/
2 DATA BIT(320); /*MAX 40 BYTES OF IMMED DATA */

```

where NODE, PSETID, PARENT and BSETID have the same meaning as those in RETN_ARG described above, OP is the operation code, and DLEN and DATA are data to be created or data used to identify itself and others.

The UPDN_RTN control block returns a return code either confirming the operation or indicating problems encountered in the processing.

The method used by UPDN to process a request is as follows:

1. Make a pass through the UPDN request to fill up a data structure TEMP which is used to remember all nodes which have any child node whose operation code is 'ID' (i.e., Identity);
2. Use TEMP, working from the bottom of the tree to top, resolve all predicates; reduce the original tree to a 2-level tree without any ID-operation nodes;
3. Pass the reduced tree to subroutines CRTN, DELN or MODN depending on UPDN.OP at the root node.

The UPDN module makes use of RETN to resolve identity of nodes which have child nodes with OP code 'ID'. Subroutines MODN, DELN and CRTN will further call CRTP, CRTB, DELP, DELB, REPP, and REPB

subroutines to perform actual update of BEU's based on the OP codes of the nodes in the reduced tree.

5.2.4 INITIALIZATION AT INTERNAL SCHEMA LEVEL

NINIT. This module (iNternal schema level INITialization) is invoked during IPL by the upper level to initialize system databases. It first invokes MINIT which performs initialization of all levels below. If initialization is 'file', then MINIT will return a one-word key which NINIT can use to recover its system databases by retrieving a BEU using the key as its ID. This BEU contains values for PCATID, BCATID and another key that NINIT shall return to the upper level. In this way NINIT sets up the static variable PCATID and BCATID, which are to be read-accessed by several modules in this level when normal processing starts.

If the initialization is 'new', then NINIT has to bootstrap its system databases. It does this by allocating the PCAT catalogue entry BEU and directly calls subroutine CRT1 to store it in the Storage Hierarchy. (CRT1 is a subroutine which interfaces with the CRT procedure at the MM Level.) Once PCAT catalogue entry is in the database, NINIT uses DEFP to define the second Pset of the system database: the BCAT Pset. After both

Psets are defined, PCATID and BCATID values will be set and NINIT then returns.

NINIT also provides an interface for the upper level to store a one-word key for that level. This key, together with PCATID and BCATID, will be stored as a BEU in the Storage Hierarchy. The ID of this BEU then becomes the key of this level, and is stored by calling the MINIT Store Key function.

5.2.5 SUBROUTINES AT INTERNAL SCHEMA LEVEL

Most of the subroutines in this level have been briefly described in the previous subsection. There are 3 for catalogue management: GPCT (Get Pset CaTalogue entry), GBCT (Get Bset CaTalogue entry) and UPCT (UPdate Pset CaTalogue entry), 4 for retrieval operations: RETP (RETrieve next P-element), RETB (RETrieve the Binary associated element), VERIF (VERIFy) and BUILD, 6 for update operations: CRTP (CReaTe P-elemtn), CRTB (CReaTe Binary association), DELP (DELeTe P-element), DELB (DELeTe Binary association), MODP (MODify P-element) and MODB (MODify Binary association), and 4 for interfacing with the MM Level: CRT1, DEL1, REP1 and RET1. The last group each interfaces with one procedure at the MM Level, and basically takes a BEU as input and generate appropriate request to be passed down.

5.3 ENTITY LEVEL

This level processes semantics of the database. It accepts definition of entity sets, attributes and constraints and responds to commands to manipulate entities.

5.3.1 DATA STRUCTURES AT ENTITY LEVEL

This level views the world as a collection of entities and attributes of these entities. This entity network data model has been described in section 2.2.

Mapping between the entity network and the data structures implemented in the Internal Schema Level is as follows. Every node in the entity network graph is implemented as a Pset, and every arc in the graph a Bset. While the Entity Level is responsible for making these Psets and Bsets known to the Internal Schema Level (through DEFP and DEFB calls), it is not concerned about how data in these sets are actually stored.

5.3.2 SYSTEM DATABASES AND DATA DEFINITION MODULES AT ENTITY LEVEL

This level keeps two catalogues: the entity catalogue and the attribute catalogue. An entry in the former describes an entity set and an entry in the latter describes an attribute of an entity set. These two catalogues are themselves implemented as two entity sets: E*ESET and E*ASET. They are displayed in the following figure.

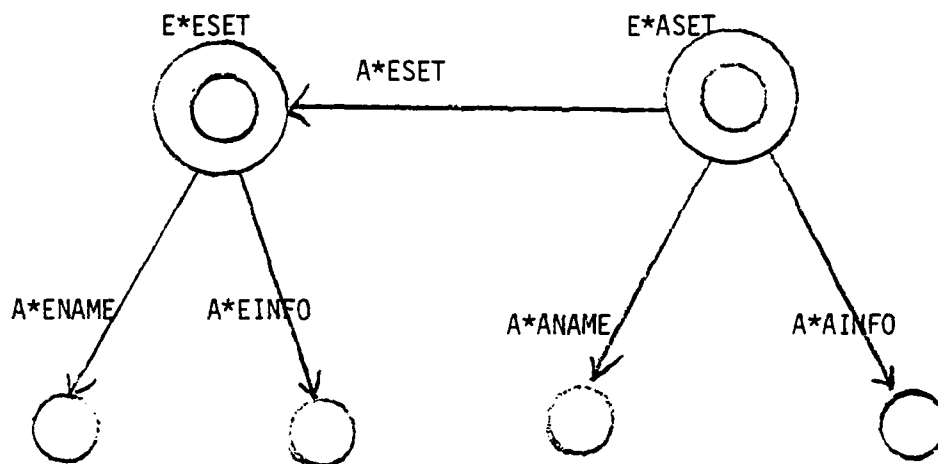


Figure: Catalogue structure at Entity Level

E*ESET has two 'attributes' A*ENAME and A*EINFO. The former contains the name of the entity set this catalogue entry is describing, and the latter contains the ID of the Pset implementing this entity set, which is provided for by the Internal Schema Level in response to a DEFP request.

E*ASET has three 'attributes'. A*ANAME contains the name of the attribute. A*AINFO, an encoded attribute, contains a data structure declared as follows:

```

/** CATALOGUE STRUCTURES AT THE ENTITY LEVEL***/
DCL 1 AINFO,
    2 BSETID BIT (32), /*BSETID OF THIS ATTRIBUTE**/
    2 PSETID BIT (32), /*PSETID OF TARGET NODE**/
    2 ENAME CHAR (8), /*IF TARGET IS A ENTITY SET*/
    2 MAX FIXED BIN (31),
    2 MIN FIXED BIN (31),
    2 MLEN FIXED BIN,
    2 FUNC CHAR(1),
    2 TYPE CHAR (1),
    2 VTYPE CHAR (1);

```

where BSETID is the ID of the Bset implementing the association between the Pset that implements the entity set this attribute belongs to and the Pset that implements the domain of this attribute. This ID is produced by the Internal Schema Level in response to a DEFB request. PSETID is the ID of the Pset

implementing the domain of the attribute. ENAME is the name of the domain entity set if this attribute is an entity attribute. TYPE specifies the type of this attribute (i.e., entity or value). If it is a value type attribute, VTYPE specifies the value type (i.e., character string or numeric), and MLEN specifies the maximum length in terms number of characters or number of digits this attribute will have, and MAX and MIN provide the value range of the numeric attribute. These parameters are given by the DBA during the data definition session described in section 2.2.

The third attribute of E*ASET is A*ESET. It is an entity attribute whose domain is the E*ESET entity set. This attribute has a relationship type many-to-one, which means that many E*ASET entities may be related to one E*ESET entity. This attribute specifies the entity set this attribute belongs to.

Two T-proc's form the data definition interface at this level: DEFE (DEFine Entity set) and DEFA (DEFine Attribute). They are invoked by the DDB module in the User Interface Level. Both modules will check a definition request against duplicate names. They will then call subroutines DFP1 and DFB1 to formulate DEFP and DEFB requests to be passed to the Internal Schema Level. To update catalogues, both modules make use of UPDE, the data manipulation module at this level, since the catalogues are themselves entity sets. This greatly simplifies the logic of these two modules.

Another module, SHWE (SHoW Entity level catalogue) will, when invoked, retrieve the catalogue entries and decode the attribute information for display by the User Interface Level. In retrieving catalogue entries, all three modules (DEFE, DEFA and SHWE) make use of subroutines GECT, GACT and ALLA, which will be described later.

5.3.3 DATA MANIPULATION AT ENTITY LEVEL

RETE. This module (RETrieve Entities) is passed a tree-like retrieval request. The root node describes the entity set of which information is being requested, and the leaf nodes describe the attributes of interest of this entity set. Predicates on the attributes, if any, are passed along with the leaf nodes. The request is represented by a link list of RETE calling control blocks RETE_ARG. Each control block describes one node in the retrieval tree. Declaration of RETE_ARG is as follows:

```
DCL 1 RETE_ARG    BASED(P),
      2 LEN      FIXED BIN (15) INIT (61),
      2 CBTP     FIXED BIN (15) INIT( 21 ),
      2 PTR     PTR INIT (NULL),
      2 NODE     FIXED BIN (15) INIT(1), /*NODE NUMBER*/
      2 NAME     CHAR (8), /*ANAME OR ENAME*/
```

```

2 PARENT FIXED BIN (15) INIT(0), /*PARENT NODE NUMBER*/
2 GET BIT(8) INIT ('00000000'B), /*SEE RETEGET FOR DETAIL*/
2  N   FIXED   BIN INIT(0), /*NUMBER OF RELEVANT 'OR'
PREDICATES*/
2 PRED (1)/*IN THIS VERSION ONLY ONE PREDICATE EACH TIME IS
    ALLOWED; IN THE FUTURE MAY USE REFER OPTION TO N*/,
3 CN FIXED BIN (15) INIT(1),/*CONDITION NUMBER*/
3 SETOP BIT(8), /*RELEVANT IF BSET IF MULTI-TARGET;
    NOT USED IN CURRENT VERSION*/
3 OP BIT(8), /*COMPARISON OPS, SEE RETEOP FOR DETAIL*/
3 DLEN FIXED BIN (15) INIT(0), /* LEN OF COMPARE DATA IN
BYTES*/
3 CDATA CHAR (40);/*40 BYTES MAXIMUM*/

```

where NODE and PARENT have similar meaning as those in the RETN_ARG control block (section 5.2.3), NAME is either the entity set name if it is a root node, or the attribute name if it is a leaf node, GET indicates whether to get all, any or none of the qualified occurrences of this attribute. Under PRED, OP refers to a comparison operator (i.e., >, < or =), DLEN and CDATA are the length and the character form of the predicate data. N, CN and SETOP are currently not used.

The RETE module retrieves catalogue entries of all entity sets and attributes involved in the RETE request by subroutine calls to GACT and GECT, which retrieve attribute information and entity set information. Any undefined name found will cause the

request to be rejected with the unfound name pointed out. Catalogue information is also used to verify and transform the predicate values passed along with the request. Transformation is necessary since the internal representation of data may not always take the character form. Data type transformation and verification are done through a subroutine VTPE. Once all these are done, a RETN request will be formulated and passed to the RETN module at the Internal Schema Level. When the latter returns, its answer set will be used to generate RETE's answer set, which is a link list of RETE_RTN control blocks. The RETE_RTN control block is similar to that of the RETN_RTN control block except that DATA is now in character form ready to be displayed.

UPDE. This module (UPDate Entities) is invoked to create, delete and modify entities in the entity sets already defined. UPDE is also passed a tree-like request, where the root node describes the entity set of which occurrences are to be updated, and the leaf nodes describe the attributes which are either used to identify the occurrences to be updated, created or added to. The UPDE request and its return control blocks are similar to those of the UPDN module.

UPDE performs validity checking on the attributes being passed. Any violation of validity constraints defined for the values of the attributes will cause the request to be rejected entirely, with the faulty attribute pointed out. UPDE also calls

subroutines GECT, GACT and VTPE for catalogue retrieval and data type verification and transformation. After all these are done, UPDE formulates a UPDN request to be passed to the Internal Schema Level to update P-elements and their relationships.

UPDE returns a return code confirming the success of the operation or pointing out why it has failed.

VNME. This module (Verify NaMEs) is invoked by some User Interface Modules to verify the entity names and attribute names a user has typed in before he enters the data. This is provided so that a user will get a more responsive system interface. The calling control block VNME_ARG contains an array of names structured in a tree form. VNME will verify these names against the catalogues at the Entity Level and point out problems if there is any.

5.3.4 INITIALIZATION AT ENTITY LEVEL

VINIT. This module initializes system databases at this level. Like NINIT, when it is invoked, it first calls the initialization module at the lower level to initialize all the lower levels. Depending on the method of initialization (file or new), it either receives a one-word key back based on which it

recovers its system databases, or it has to bootstrap the system databases into the Storage Hierarchy.

To bootstrap its system databases which are the first two entity sets of the system, VINIT calls DEFP 6 times and DEFB 5 times to set up the Psets and Bsets involved in the catalogues. The 11 ID's thus generated are then stored in yet another special Pset (the 7th Pset that VINIT has defined) by calling UPDN. The ID of the 7th Pset is then stored as the key of this level by invoking the Store Key function of the NINIT procedure. The 11 ID's are also retained in a static array KEY which can be read-accessed by the catalogue management subroutines GACT, GECT and ALLA at this level.

Another subroutine EBOT (Entity catalog BObstrap) is called by VINIT to put into these 6 Psets and 5 Bsets the definition of the catalogues. (Since the catalogues are themselves entity sets with attributes, their definitions will be the first to appear as data in these Psets and Bsets.) EBOT uses 7 UPDN calls to accomplish this task.

If initialization is from an existent database, VINIT will retrieve the 11 ID's back by accessing the Pset whose ID is returned to VINIT by NINIT as the one-word key.

5.3.5 SUBROUTINES AT ENTITY LEVEL

The six subroutines have been introduced briefly in the previous subsections. Three are catalogue management subroutines: GECT (Get Entity CaTalogue entry), GACT (Get Attribute CaTalogue entry) and ALLA (get ALL Attributes). ALLA is invoked by SHWE when the user wishes to find out all attributes defined for a particular entity set. All three modules make use of the KEY array to get the ID's of the Psets and Bsets forming the catalogues, and formulates a RETN request to obtain these data. Subroutines for retrieving attribute catalogue entries also perform the task of decoding a bit string into elements in the AINFO data structure for direct use by the caller.

Theoretically there is no need for a set of special retrieval routines just to retrieve the catalogue entries. Since the catalogue entries are themselves attributes of entities in the database, RETE should be able to accomplish the same task. These special routines are devised mainly for performance reasons. RETE, as it turns out, is a very expensive module. Using RETE to retrieve catalogue entries would degrade performance dramatically. While performance of the system is not a primary concern at this stage, it was felt that this deviation from theoretical elegance was worthwhile in the current implementation. Performance of a multi-layered database

architecture is a future research issue which may provide clues to whether this deviation can be (or should be) avoided.

Another two of the subroutines, DFB1 and DFP1, are used to generate DEFB and DEFP requests in the proper format to be passed to the next level. Finally, subroutine VTPE (Verify TyPE) is used to verify and transform data passed along with the RETE or UPDE requests.

5.4 THE USER INTERFACE LEVEL

This level interacts directly with the user. It does not create or use any system database. Its function is primarily one which parses users' requests and generates in proper formats requests to be passed down to the Entity Level. Therefore the usual classification of modules into data definition, data manipulation and initialization modules is not used here.

5.4.1 USER INTERFACE MODULES

USER. This module is invoked when a user 'logs on'. (Presently there is no security checking facility.) USER simply asks for the subsystem the user wishes to get in. Only DBA subsystem is currently available. USER then passes control to the DBA module.

DBA. This module (Data Base Administrator session) takes control when a user enters the DBA session. It calls various subroutines depending on what the user wishes to do: data definition (DDB), data definition query (DDQ) or data manipulation (DDM). These three subroutines are described in the following paragraphs.

DDB. This module (Data Definition for Base schema) is invoked to interface with a DBA's request to do data definition for the base schema. (A base schema is a schema which describes the integrated database in terms of constructs in the entity network data model. It is different from a view schema which describes a view in terms of a data model which is not necessarily the base data model. Presently no view facility is available.) A user may define an entity set and then its attributes. He may also add attributes to an entity set defined previously. The DDB module interfaces with the user through a panel facility. The panel facility will display pre-defined panels at the user's terminal and allow the user to respond by filling out blanks on the panel.

This module will formulate DEFE and DEFA requests to be passed down to the Entity Level based on parameters the user specifies in the panels.

DDQ. This module (Data Definition Query) is invoked to display content of data definitions. Its major task is to formulate properly a SHWE request to be passed to the Entity Level. When SHWE returns, DDQ displays the result at the user's terminal.

DMB. This module (Data Manipulation for Base data) is invoked when a user enters a data manipulation session. It calls subroutines DMQ, DMC, DMM and DMD based on the type of manipulation operations. These subroutines are described below.

DMQ. This module (Data Manipulation - Query) accepts a user's query and parses it into meaningful tokens to be placed in a RETE request. It calls subroutines LEX to break the input character string into tokens. Then it uses subroutine LEXT to recognize the tree structure of the query. Once data structures describing the tree have been produced by LEXT, DMQ formulates a RETE request and passes it to the Entity Level. Upon receiving return data from the latter, DMQ calls subroutine PRNT to display the result at the user's terminal.

DMC. This module (Data Manipulation - Creating entities) accepts a user's request to create entities in pre-defined entity

sets. The user first provides a list of attributes whose values will be provided when an entity is created. DMC will, like DMQ, call LEX and LEXT to build a tree out of the list of attribute names and formulate a VNME request to be passed to the Entity Level. The latter validates these names against the entity level catalogues. If no error is detected in the attribute list, DMC then asks the user to enter data for these attributes. Every line of data entered will cause one entity to be created. DMC uses subroutine UPE1 to input data lines and to formulate UPDE requests. Violation of constraints on data types or data values will be signaled through a return code from UPDE and DMC will display proper error messages based on the return code.

DMM and DMD. These two modules (Data Manipulation - Modify entities and Data Manipulation - Delete entities) are similar to DMC in their operations.

5.4.2 OTHER SUBROUTINES AT USER INTERFACE LEVEL

This subsection describes the 5 subroutines used by modules described in the previous subsection.

LEX. This module (LEXical analyzer) breaks input lines into tokens. Token delimiters recognized are ',' and ':'. Delimiters

enclosed in quotes are not treated as exceptions. More than one line of input can be strung together by ending previous lines in back slashes. Blanks can be optionally taken out of the input character string. However, blanks enclosed in quotes are not suppressed. This module returns an array TOKEN which contains the tokens and their lengths. (Note that the fact that delimiters found in quotes are not taken as exceptions means that this current implementation of lexical analyzer cannot accommodate data values that contain these special characters.)

LEXT. This module (LEXical Tree structure) is given the array TOKEN produced by subroutine LEX and is asked to identify the hierarchical structure of names and data values implied by the nested parantheses. For example, if TOKEN(1) is 'EMPLOYEE(', TOKEN(2) is 'EMPNAME' and TOKEN(3) is 'EMPADDR)', then EMPNAME and EMPADDR are the children names of the name EMPLOYEE. This module is also required to recognize non-name tokens. There are two types of non-name tokens: the modification operators (MOP) and the predicate values. The former is distinguished from regular name tokens by its special starting character '-'. The latter always trails a comparison operator ('=', '>', '<') behind a name. Therefore if a token is found to contain a comparison operator, it is broken into a name token and a value token. If a value is surrounded by quotes, the quotes are removed.

This module returns several data structures describing its parsing results. Array T1 contains the hierarchical structure of

names, array PRED contains value tokens and array MOP contains modification operators found in the input character string.

UPE1. This module (UPdE interface) is used by data update modules (DMC, DMD and DMM) to get input data and to formulate a UPDE request for each input line. The caller will pass to UPE1 the tree structure of entity and attribute names that have been recognized by LEXT as well as the update operators (e.g., insert, delete, replace) corresponding to the input data items. UPE1 makes use of these data structures, calls LEX to break the input line into data items, and formulates proper UPDE requests to be passed to the Entity Level.

VNM1. This module (VNME interface) is invoked by data update modules (DMC, DMD and DMM) to formulate proper VNME requests to be passed to the Entity Level. This is done to confirm the legality of entity and attribute names before accepting the update data from the user. This module is passed a data structure describing the tree structure of the names and will return a return code indicating any problem found.

PRNT. This module (PRiNT) performs the task of displaying the result of a user query. It is passed a pointer pointing to the RETE_ARG chain and a pointer pointing to the RETE_RTN chain and will use information contained in these chains to display header lines and data lines at the terminal.

5.5 THE DRIVER MODULE - FSTV

The FSTV module is invoked by the Control Structure when starting the simulation run. It is the driver program of the Functional Hierarchy. The purpose of this module is to start up initialization of the entire Functional Hierarchy and to invoke the user session.

System Traces. FSTV first calls subroutine SETFH which sets up run time parameters. The only run time parameters used in the current implementation are the trace options. There are five types of trace options. The TCALL trace option will display messages when a T-proc is invoked as well as when it returns. This trace can be selected on a level basis. It is implemented through the TCALL program, since all T-proc invocations necessarily go through TCALL.

The system error message trace will display messages when an unusual situation is detected. This normally signals that some system software errors have been detected. This trace can also be turned on and off on a level basis.

The third type of trace is the memory request trace. If this is turned on, every data element that has been created, deleted or updated in the Storage Hierarchy will be displayed along with its virtual address. It displays these data elements using the

BEU format. It is implemented within subroutines CRT1, RET1, DEL1 and REP1.

The fourth type of trace is the Internal Schema Level data manipulation request trace. It displays all the requests that go into the RETN and UPDN modules. This trace is implemented within these two modules.

The last type of trace is the timing report option. It keeps track of the virtual CPU time consumed by each T-proc (including subroutine calls within the T-proc) and the number of times a T-proc is invoked. A sample timing report is shown in the following figure. This option is implemented through the TCALL module, and the result is displayed at the end of the run by the program FSTV.

LEVEL	PROCNAME	COUNT	TOT ELAPSED TIME	TOT RUN TIME	RUN TIME/INVOCATION
1	USER	1	1505283	80955	80955
2	VINIT	1	123976	5233	5233
2	VNME	0	0	0	0
2	DEFE	0	0	0	0
2	RETE	1	1146343	17482	17482
2	UPDE	0	0	0	0
2	DEFA	0	0	0	0
2	SHWE	1	1424328	11921	11921
3	DEFP	0	0	0	0
3	DEFB	0	0	0	0
3	RETN	8	1503249	567142	70892
3	UPDN	0	0	0	0
3	NINIT	1	10419	5637	5637
4	RET	138	939625	939625	6808
4	DEL	0	0	0	0
4	REP	0	0	0	0
4	CRT	0	0	0	0
4	MINIT	1	1264	1264	1264
5	L1	801	180119	180119	224

Figure: Sample Timing Report

After SETFH is finished, FSTV invokes VINIT to initialize the Functional Hierarchy. Then FSTV invokes USER, the T-proc at the User Interface Level to start up the user session.

6.0 CONCLUDING REMARKS AND FUTURE DIRECTIONS

This report describes the current implementation of the Software Test Vehicle of the Functional Hierarchy of the INFOPLEX database computer. It provides an introduction to the STV project and the relationships among its various components, and documents the programming conventions used in FSTV.

This current version of FSTV is implemented with future research and expansions in mind. It is not meant to provide detailed performance statistics. Nor is it meant to incorporate all the functionalities that a large DBMS should have. However, it has provided some interesting insights into issues that ought to be explored further, and a basis for incorporating future efforts.

6.1 MEMORY MANAGEMENT LEVEL REVISITED

Segmentation and garbage collection are among the more important functions that are missing in the current

implementation. An interface which allows the upper level to define segments and to request that a certain element be stored in a pre-defined segment will produce a much more logical grouping of data elements in the virtual memory address space. This means that the Memory Management Level will maintain its own system database which describe the availability of each segment. Where and how this system database is maintained are also issues that need to be resolved.

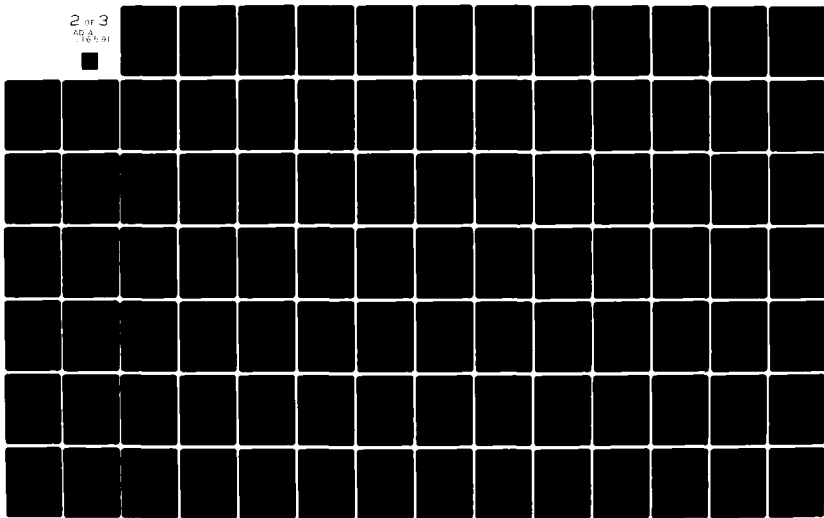
6.2 INTERNAL SCHEMA LEVEL REVISITED

The purpose of this level is to provide the upper levels with a powerful data structure management facility. However, capabilities implemented in the current version leave very much to be desired. In terms of functionalities, the binary association types, currently limited to one-to-one and many-to-one, should be expanded to include one-to-many and many-to-many. Accompanying this expansion is the enrichment of the set of primitives used to express retrieval and update predicates. Set-oriented primitives such as INCLUDE, BELONG_TO, ALL and ANY are more efficiently processed at this level than at a higher level, therefore they ought to be included in the RETN requests. The predicate structure should also include the OR boolean operator. In addition, the INVERSE function should also

AD-A116 591

ALFRED P SLOAN SCHOOL OF MANAGEMENT CAMBRIDGE MA CEN--ETC F/G 9/2
FSTV: THE SOFTWARE TEST VEHICLE FOR THE FUNCTIONAL HIERARCHY OF--ETC(U)
JAN 82 M HSU
N00039-81-C-0663
UNCLASSIFIED CISR-M010-8201-09 NL

2 of 3
AD-A116 591



be provided for definition of a binary association which is the inverse of one already defined. This provides more flexibility for the user in phrasing a query while leaves the access path selection problem entirely to the Internal Schema Level.

In terms of interface, the current definition and manipulation interface structure should be more flexible. For example, several Bsets whose source Psets are the same may be defined in one DEFB request. Additional construct definition may be desirable. For example, an often used retrieval tree may be defined to be a construct such that future retrieval requests may simply cite the ID of that construct without constructing the same tree again. These enhancements will improve convenience of use as well as efficiency of operations.

Those described above are potential improvements that the upper level can see. Equally important are the improvements of internal processing. Intelligent query decomposition and more powerful search mechanisms will have to be incorporated to cope with problems of traversing a large data base. A preliminary design which further decomposes this level into two levels has been proposed, where the lower level is responsible for fast search in a flat file and the upper level is responsible for query decomposition and linkage of data. A major 'surgery' is expected on the current implementation in order to bring about these improvements.

6.3 THE ENTITY LEVEL REVISITED

The main purpose of this level is to build semantics and enforce constraints. One powerful arm in building semantics, the Virtual Information Facility, is largely missing in the current implementation. This facility may run parallel to the current level on a 'pipeline' basis. That is, retrieval requests will first pass through a 'virtual information filter' which transforms the request into one which contains only references to actual, stored entities and attributes. When the latter has been retrieved by the RETE facility, the return data will go through the second stage of the virtual information facility, a 'virtual information encoder', which will compute the virtual information values based on virtual information definitions. Architecturally this facility could lie above the Entity Level and form a Virtual Information Level; however, data structures being dealt with in this facility are the same as those in the Entity Level. That is to say, there will not be another level of data abstraction.

6.4 LOCATION AND MANAGEMENT OF CATALOGUES AND WORKING BUFFERS

There exists a spectrum of design philosophies for a stratified architecture. On the one extreme, all software at one

level may be treated as 'data' at the next level, effecting a hierarchy of levels each of which implements its 'virtual address space' at the next lower level; on the other extreme, the software at one level is divided into two distinct parts; one part (i.e., code and catalogues) always exists at the local level, while the other part (e.g., user data) always exists at the next level (and eventually in the Storage Hierarchy.)

The strategy used in the preliminary design of the Functional Hierarchy is somewhere in between: catalogues and regular data are stored as data at the next lower level while program modules and their contexts reside within a level.

Intuitively, some duplication of catalogue and data within a level may be desirable based on performance considerations. As the FSTV runs show, catalogue retrievals at the higher levels consume a great percentage of the system processing power. This overhead may prove to be even more costly if the database is large and more functionalities are incorporated.

Another aspect in this issue is that of a 'stored procedure' at the lower levels. In other words, in addition to storing data at the next lower level, a level may choose to declare some frequently encountered requests to the next lower level, so that the next lower level may have some specialized, more efficient code to process them. This is analogous to a 'compiled query' mechanism adopted in System R.

Yet another issue is that of a 'program module overflow'. As a level develops to include more functions, its program size may become too large to be all stored economically within the level's local memory. Less frequently used modules may be removed and stored as data at the next lower level, which in turn will store them as data at the next lower level, until it is stored as part of the data in the Storage Hierarchy. Similar arguments may be devised for a program context of a request that requires a large amount of buffer space.

However, while these are issues FSTV has exposed, they are not likely to be resolved without a rigorous investigation of tradeoffs among alternatives. It also requires efforts in designing additional algorithms that will ensure data integrity in the case of data duplication.

REFERENCES

- Blumberg81: Blumberg, B. INFOSAM - A sample database management system. Master Thesis, Sloan School of Management, MIT (May 1981)
- Hsu80: Hsu, M. A preliminary architectural design for the Functional Hierarchy of the INFOPLEX database computer. TR #:M010-8011-05, Sloan School of Management, MIT (Nove 1980)
- Madnick79: Madnick, S.E. The INFOPLEX database computer: Concepts and directions. Proc. IEEE Computer Conference (Feb 1979)
- To81: To, T. Shell: A simulation for the Software Test Vehicle of the INFOPLEX database computer. Bachelor Thesis, Sloan School of Management, MIT (may 1981)

APPENDIX I

SAMPLE TERMINAL SESSION (a)

Sample Terminal Session

FSTV 1881-804

ENTER TCALL TRACE OPTION (4 BITS)

:
0000
ENTER SYSTEM ERROR MESSAGE TRACE OPTION (4 BITS)

:
0000
ENTER MEMORY REQUEST TRACE OPTION (1 BIT):
0
ENTER UPDN/RETN REQUEST TRACE OPTION (1 BIT):
0

--FUNCTIONAL HIERARCHY STV CONSOLE PROGRAM--

INITIALIZATION: FILE OR NEW:
FILE

FILE NAME?

:
TEST20 ← *Load file created in the previous session*

FSTV CALLS VINIT

FSTV: VINIT RETURNS

SUBSYSTEMS: DBA, BV(BASE VIEW) OR RV(RELATIONAL VIEW)?

:
DBA

-- DATABASE ADMINISTRATOR (DBA) SESSION --

DBA: DATA DEFINITION (DD) OR DATA MANIPULATION (DM) OR DATA DEFINITION QUERY (DDQ)?

:
DD

DD: BASE DATA (BASE) OR VIEW DATA (VIEW)

:

DBA: DATA DEFINITION (DD) OR DATA MANIPULATION (DM) OR DATA DEFINITION QUERY (DDQ)?

:
DDQ

-- DATA DEFINITION QUERY SESSION --

ENTITY SET NAME? USE * IF ALL ENTITY SET NAMES ARE DESIRED.

: * ← Want all Entity Sets defined in the system

NUMBER OF ENTITY SETS DEFINED

5

PROJECTS
DEPT
EMPLOYEE
E*ASET
E*ESET

5 entity sets in the system

ENTITY SET NAME? USE * IF ALL ENTITY SET NAMES ARE DESIRED.

:
PROJECTS

ATTRIBUTE NAMES? SEPERATE BY COMMAS. USE * IF ALL ATTRIBUTES ARE DESIRED

Want all attributes defined for entity set PROJECTS

ENTITY SET NAME PROJECTS

ATTRIBUTE NAME FUNCTION TYPE ENAME VTYPE MAX LEN MAX VALUE MIN VALUE

UNDER M:1 E DEPT

PROJNUM KEY V N

99999 -99999

PROJNAME 1:1 V C
20

3 attributes defined for PROJECTS

ENTITY SET NAME? USE * IF ALL ENTITY SET NAMES ARE DESIRED.

:
EMPLOYEE

DBA: DATA DEFINITION (DD) OR DATA MANIPULATION (DM) OR DATA DEFINITION QUERY (DDQ)?

:
DM

-- DATA MANIPULATION SESSION --

DM: ENTER MANIPULATION COMMAND:

CREATE(CRT), MODIFY(MOD), DELETE(DEL), QUERY(QUE)

:
CRT ← *enter CREATE session*

CREATE: ENTER ENTITY SET NAME

:
DEPT

ENTER NAMES OF ATTRIBUTES SEPARATED BY COMMA

:
DEPTNUM,DEPTNAME

ENTER DATA:

15, 'Sloan School'

:
14, Economics

CREATE: ENTER ENTITY SET NAME

:
← *leave CREATE session*

DM: ENTER MANIPULATION COMMAND:

CREATE(CRT), MODIFY(MOD), DELETE(DEL), QUERY(QUE)

:
QUE ← *enter QUERY session*

--QUERY SESSION--

TYPE YES IF NEED HELP

:

ENTER ENTITY SET NAME

:
DEPT

ENTER ATTRIBUTE NAMES AND PREDICATE. SEPARATED BY COMMAS

:
DEPTNAME,DEPTNUM

ENTITY SET NAME DEPT

| DEPTNAME | DEPTNUM |

| ECONOMICS | 14 |

| 'Sloan School' | 15 |

Answer to the query

ENTER ENTITY SET NAME

:
← Leave QUERY session

DN: ENTER MANIPULATION COMMAND:

CREATE(CRT), MODIFY(MOD), DELETE(DEL), QUERY(QUE)

:
MOD ← Enter MODIFY session

TYPE YES IF NEED HELP

:

MODIFY: ENTER ENTITY SET NAME

:
DEPT

ENTER MODIFICATION OPERATORS AND NAMES OF ATTRIBUTES SEPARATED BY COMMAS

: ID:DEPTNUM, -REP: DEPTNAME ←

the way to modify: identify by DEPTIDUM, then replace DEPTNAME

ENTER DATA:
15. Sloan School

← Change DEPTNAME of DEPT with DEPT NUM 15 to Sloan School

MODIFY: ENTER ENTITY SET NAME

← leave modify session

DM: ENTER MANIPULATION COMMAND:

CREATE(CRT), MODIFY(MOD), DELETE(DEL), QUERY(QUE)

: QUE ← back to QUERY session

--QUERY SESSION--

TYPE YES IF NEED HELP

ENTER ENTITY SET NAME

: DEPT

ENTER ATTRIBUTE NAMES AND PREDICATE. SEPARATED BY COMMAS

: DEPTNUM, DEPTNAME

ENTITY SET NAME DEPT

: DEPTNUM : DEPTNAME :

: 14 : Economics :

: 15 : Sloan School :

} Answer to query

ENTER ENTITY SET NAME

DEPT

ENTER ATTRIBUTE NAMES AND PREDICATE, SEPARATED BY COMMAS

: DEPTNAME, DEPTNUM=15 ← *a query with a predicate*

ENTITY SET NAME

DEPT

| DEPTNAME | DEPTNUM |

| Sloan School | 15 |

ENTER ENTITY SET NAME

DEPT

ENTER ATTRIBUTE NAMES AND PREDICATE, SEPARATED BY COMMAS

: DEPTNAME='Sloan School', DEPTNUM

ENTITY SET NAME

DEPT

| DEPTNAME | DEPTNUM |

| Sloan School | 15 |

ENTER ENTITY SET NAME

← *Leave Query Session*

DM: ENTER MANIPULATION COMMAND:

CREATE(CRT), MODIFY(MOD), DELETE(DEL), QUERY(QUE)

CRT ← *Enter (P)lease session*

Answer for the query

Answer

CREATE: ENTER ENTITY SET NAME

: DEPT

ENTER NAMES OF ATTRIBUTES SEPARATED BY COMMA

: DEPTNUM,DEPTNAME

ENTER DATA:

6.EE-Computer Science

: 21.Humanities

create two other DEPT entities

CREATE: ENTER ENTITY SET NAME

DM: ENTER MANIPULATION COMMAND:

CREATE(CRT), MODIFY(MOD), DELETE(DEL), QUERY(QUE)

: CRT

CREATE: ENTER ENTITY SET NAME

: EMPLOYEE

ENTER NAMES OF ATTRIBUTES SEPARATED BY COMMA

: EMPNAME,EMPADDR,AGE,WORKS,IN(DEPTNUM)

ENTER DATA:

Hoo-min Toong, E53-300, 50, 15

: Mike Abraham, Somewhere in Somerville, 40, 14

create 2 other EMPLOYEE entities

CREATE: ENTER ENTITY SET NAME

DM: ENTER MANIPULATION COMMAND:

CREATE(CRT), MODIFY(MOD), DELETE(DEL), QUERY(QUE)

: QUE

--QUERY SESSION--

TYPE YES IF NEED HELP

ENTER ENTITY SET NAME

: EMPLOYEE

ENTER ATTRIBUTE NAMES AND PREDICATE, SEPARATED BY COMMAS

: EMPNAME,WORKS_IN(DEPTNUM,DEPTNAME)

ENTITY SET NAME EMPLOYEE

: EMPNAME | WORKS_IN (DEPTNUM | DEPTNAME) |

: Mike Abraham | 14 | Economics |

: Hoo-mIn Toong | 15 | Sloan School |

ENTER ENTITY SET NAME

: EMPLOYEE

ENTER ATTRIBUTE NAMES AND PREDICATE, SEPARATED BY COMMAS

: EMPNAME, AGE<45 ← Quantifies query with predicate

ENTITY SET NAME EMPLOYEE

! EMPNAME ! AGE !

Answer to the query

! Mike Abraham ! 40 !

ENTER ENTITY SET NAME

: ← get out

DM: ENTER MANIPULATION COMMAND:

CREATE(CRT), MODIFY(MOD), DELETE(DEL), QUERY(QUE)

: ← get out

DBA: DATA DEFINITION (DD) OR DATA MANIPULATION (DM) OR DATA DEFINITION QUERY (DDQ)?

: ← get out

-- END OF DBA SESSION --

SUBSYSTEMS: DBA, BV(BASE VIEW) OR RV(RELATIONAL VIEW)?

: ← get out

-- END OF USER SESSION--

SAVE FILE: FILE NAME?

: TEST21 ← Save under TEST21

TEST ENDS

R: T=12.30/14.94 23:14:04
spool console stop

APPENDIX I

SAMPLE TERMINAL SESSION(b)

GENMOD FSTV MODULE Q (FROM PLISTART
R: T=1.46/2.51 22:45:07
fstv 125(-90k)

ENTER TCALL TRACE OPTION (4 BITS)

turn off all traces

0000 ENTER SYSTEM ERROR MESSAGE TRACE OPTION (4 BITS)

0000 ENTER MEMORY REQUEST TRACE OPTION (1 BIT):

0 ENTER UPDN/RETN REQUEST TRACE OPTION (1 BIT):

--FUNCTIONAL HIERARCHY STV CONSOLE PROGRAM--

INITIALIZATION: FILE OR NEW:
FILE

FILE NAME?

: TEST19 *TEST19 already contains an initial database*

FSTV CALLS VINIT

FSTV: VINIT RETURNS

SUBSYSTEMS: DBA, BV(BASE VIEW) OR RV(RELATIONAL VIEW)?

: DBA

-- DATABASE ADMINISTRATOR (DBA) SESSION --

DBA: DATA DEFINITION (DD) OR DATA MANIPULATION (DM) OR DATA DEFINITION QUERY (DDQ)?

: DD

DD: BASE DATA (BASE) OR VIEW DATA (VIEW)

: BASE

-BASE DATA DEFINITION SESSION-

YOU MAY DEFINE NEW ENTITY SETS OR ADD ATTRIBUTES TO EXISTING ENTITY SETS.

NEW ENTITY SET (NEW) OR EXISTING ENTITY SET (OLD)?

: OLD

← ATTRIBUTE AGE DEFINED

a panel session followed which defined an AGE attribute for an existing entity set EMPLOYEE

PRESS ENTER TO CONTINUE

: ATTRIBUTE BOSS DEFINED

Another panel session defined the BOSS attribute

PRESS ENTER TO CONTINUE

NEW ENTITY SET (NEW) OR EXISTING ENTITY SET (OLD)?

: NEW

← ENTITY SET DEPT DEFINED

a panel session defined the DEPT entity set

PRESS ENTER TO CONTINUE

: ATTRIBUTE DEPTNUM DEFINED

Attribute DEPTNUM defined here

PRESS ENTER TO CONTINUE

: ATTRIBUTE DEPTNAME DEFINED

Attribute DEPTNAME defined here

PRESS ENTER TO CONTINUE

NEW ENTITY SET (NEW) OR EXISTING ENTITY SET (OLD)?

NEW
Entity set PROJECTS defined here.
ENTITY SET PROJECTS DEFINED

PRESS ENTER TO CONTINUE

Attribute PROJNAME here
ATTRIBUTE PROJNAME DEFINED

PRESS ENTER TO CONTINUE

Attribute PROJNUM defined here
ATTRIBUTE PROJNUM DEFINED

PRESS ENTER TO CONTINUE

NEW ENTITY SET (NEW) OR EXISTING ENTITY SET (OLD)?

OLD
defined attribute WORKS_IN for EMPLOYEE
ATTRIBUTE WORKS_IN DEFINED

PRESS ENTER TO CONTINUE

NEW ENTITY SET (NEW) OR EXISTING ENTITY SET (OLD)?

OLD
defined attribute UNDER for PROJECTS entity set
ATTRIBUTE UNDER DEFINED

PRESS ENTER TO CONTINUE

NEW ENTITY SET (NEW) OR EXISTING ENTITY SET (OLD)?

DD: BASE DATA (BASE) OR VIEW DATA (VIEW)

DBA: DATA DEFINITION (DD) OR DATA MANIPULATION (DM) OR DATA DEFINITION QUERY (DDQ)?

-- END OF DBA SESSION --

SUBSYSTEMS: DBA, BV(BASE VIEW) OR RV(RELATIONAL VIEW)?

TEST20

TEST20 IS NOT A VALID COMMAND

SUBSYSTEMS: DBA, BV(BASE VIEW) OR RV(RELATIONAL VIEW)?

-- END OF USER SESSION--

SAVE FILE: FILE NAME?

TEST20 ← Save as TEST20

TEST ENDS

R; T-9.86/12.17 22:59:08
spool console class h start
R; T-0.01/0.02 23:00:41

APPENDIX 2

MACRO DECLARATIONS

(Control Blocks, Catalogues, Entries)

FILE: DATA MACLIB A VM/SP CONVERSATIONAL MONITOR SYSTEM

```

LIBPDS
DCL 1 CRT_ARG BASED(P),
  2 LEN FIXED BIN (15) INIT (102),
  2 CBTP FIXED BIN (15) INIT(3),
  2 PTR PTR INIT (NULL),
  2 N FIXED BIN, /*LEN OF DATA IN BIT*/
  2 DATA BIT(800); /*UP TO 100 BYTES LONG PER DATA UNIT*/

DCL 1 CRT_RTN BASED(P),
  2 LEN FIXED BIN (15) INIT ( 6),
  2 CBTP FIXED BIN (15) INIT(4),
  2 PTR PTR INIT (NULL),
  2 RTN_CODE FIXED BIN INIT (0),
  2 ID BIT (32);

/*
/*DCLS OF ARGUMENTS TO COMMUNICATE WITH THE MODLE DEFA*/
DCL 1 DEFA_ARG BASED(P),
  2 LEN FIXED BIN (15) INIT (37),
  2 CBTP FIXED BIN (15) INIT( 5),
  2 PTR PTR INIT (NULL),
  2 ENAME1 CHAR (8), /*PARENT ENTITY NAME*/
  2 NAME CHAR (8) INIT ((8)' '), /*ATTRIBUTE NAME*/
  2 TYPE CHAR (1) INIT (' '), /*'E'OR 'V'*/
  2 VTYPE CHAR (1) INIT (' '), /*'N'OR 'C'*/
  2 MLN FIXED BIN, /*DEFAULT IS 20*/
  2 MAX FIXED BIN(31), /*DEFAULT IS 99999*/
  2 MIN FIXED BIN(31), /*DEFAULT IS -99999*/
  2 ENAME CHAR (8) INIT ((8)' '),
  2 FUNC CHAR (1) INIT (' '); /*'S' FOR 1:1, 'M' FOR M:1, 'K'FOR KEY*/

DCL 1 DEFA_RTN BASED(P),
  2 LEN FIXED BIN (15) INIT (2 ),
  2 CBTP FIXED BIN (15) INIT( 6 ),
  2 PTR PTR INIT (NULL),
  2 RTN_CODE FIXED BIN (15) INIT(0);

/*
DCL 1 DEFB_ARG BASED(P),
  2 LEN FIXED BIN (15) INIT (17),
  2 CBTP FIXED BIN (15) INIT( 7 ),
  2 PTR PTR INIT (NULL),
  2 FUNC CHAR (1), /*'S' = SINGULAR(I.E.,1:1), 'M' = MULTIPLE
  (I.E.,M:1), 1:N AND N:M NOT AVAILABLE IN
  CURRENT VERSION*/
  2 PSETID (2) BIT(32), /*PSETID OF THE ORIGINATING PSET AND THE
  TARGET PSET OF THIS BINARY RELATION*/
  2 IMP,
  3 LINK FIXED BIN (15); /*SEE DEFBLINK FOR DETAIL*/
  /*THE FOLLOWING FIELD IS NOT USED IN CURRENT VERSION
  3 REV_LINK FIXED BIN (15); WHICH DESCRIBES THE IMP OF REV LINK*/
DCL 1 DEFBLINK,
  2 EMB FIXED BIN INIT(1), /*TARGET ELEM EMBEDDED IN SOURCE ELEM*/
  2 POINTER FIXED BIN INIT (3), /*POINTER CONNECTION*/
  /*THE FOLLOWING ARE NOT USED IN CURRENT VERSION*/
  2 FEMB FIXED BIN INIT (2), /*FREE FORMAT EMBEDDING*/
  2 NONE FIXED BIN INIT (4); /*NOT IMPLEMENTED, RELY ON REVERSE*/
DCL 1 DEFB_RTN BASED(P),

```

PAGE 002

FILE: DATA MACLIB A VM/SP CONVERSATIONAL MONITOR SYSTEM

```

2 LEN      FIXED BIN (15) INIT (12).
2 CBTP FIXED BIN (15) INIT( 8 ).
2 PTR PTR INIT (NULL).
2 RTN CODE FIXED BIN INIT (0).
2 BSETID BIT(32);

/ /
/*DCLS OF ARGUMENTS TO COMMUNICATE WITH THE MODLE DEFE*/
DCL 1 DEFE_ARG      BASED(P).
2 LEN      FIXED BIN (15) INIT (8).
2 CBTP FIXED BIN (15) INIT( 9 ).
2 PTR PTR INIT (NULL).
2 NAME CHAR (8) INIT ((8)' ');
DCL 1 DEFE_RTN BASED(P).
2 LEN      FIXED BIN (15) INIT (2 ).
2 CBTP FIXED BIN (15) INIT( 10 ).
2 PTR PTR INIT (NULL).
2 RTN_CODE FIXED BIN (15) INIT (0);

/ /

/*ARGUMENT DCLS FOR T-PROC DEFP*/
DCL 1 DEFP_ARG      BASED(P),
2 LEN      FIXED BIN (15) INIT (6).
2 CBTP FIXED BIN (15) INIT( 11 ).
2 PTR PTR INIT (NULL).
2 PTYPE CHAR (1), /*C'-CHAR, 'N'-NUM, 'B'-BIT, 'X'-NULL*/
/*NULL TYPE PSET IS A PSET OF TOKENS WHICH DO NOT CONTAIN
VALUE DATA. RETRIEVAL OF NULL TYPE PSET RESULTS IN
A RETURN DATA WITH LENGTH 0.****/
2 PLEN FIXED BIN (15), /*LEN OF PRIMITIVE ELEM OF THE PSET,
IN BYTES, IF --1 THEN VARIABLE IN LENGTH*/
2 IMP.
3 HOW BIT (8), /*SEE DEFPHOW FOR DETAIL*/
3 NUMPTR FIXED BIN (15); /*RELV ONLY IF HOW = SA*/

DCL 1 DEFPHOW,
2 SA BIT(8) INIT ('00000001'B).
2 EMB BIT (8) INIT ('00000000'B);
DCL 1 DEFP_RTN      BASED(P),
2 LEN      FIXED BIN (15) INIT (6).
2 CBTP FIXED BIN (15) INIT( 12 ).
2 PTR PTR INIT (NULL),
2 RTN CODE FIXED BIN (15), /*0 IS O.K.*/
2 PSETID BIT (32); /*ID OF PSET JUST DEFINED*/

/ /
DCL 1 DEL_ARG BASED(P),
2 LEN      FIXED BIN (15) INIT (4 ).
2 CBTP FIXED BIN (15) INIT( 13 ).
2 PTR PTR INIT (NULL).
2 ID BIT(32);
DCL 1 DEL_RTN BASED(P),
2 LEN      FIXED BIN (15) INIT ( 2).
2 CBTP FIXED BIN (15) INIT( 14 ).
2 PTR PTR INIT (NULL),
2 RTN_CODE FIXED BIN INIT (0);

/ /
DCL 1 MINIT_ARG BASED(P).

```

```

F1A00220
F1A00230
F1A00240
F1A00250
F1A00260

F1A00020
F1A00030
F1A00040
F1A00050
F1A00060
F1A00070
F1A00080
F1A00090
F1A00100
F1A00110
F1A00120

F1A00020
F1A00030
F1A00040
F1A00050
F1A00060
F1A00070
F1A00080
F1A00090
F1A00100
F1A00110
F1A00120
F1A00130
F1A00140
F1A00150
F1A00160
F1A00170
F1A00180
F1A00190
F1A00200
F1A00210
F1A00220
F1A00230
F1A00240
F1A00250

F1A00020
F1A00030
F1A00040
F1A00050
F1A00060
F1A00070
F1A00080
F1A00090
F1A00100
F1A00110
F1A00020
F1A00020

```

```

2 LEN FIXED BIN (15) INIT(13).
2 CBTP FIXED BIN (15) INIT( 15 ).
2 PTR PTR INIT(NULL()).
2 OP CHAR (1). /*'I'=-INIT. 'S'=-STORE KEY*/
2 FNAME CHAR(8). /*NAME OF FILE*/
2 KEY BIT(32);
DCL 1 NINIT_RTN BASED(P).
2 LEN FIXED BIN (15) INIT(6).
2 CBTP FIXED BIN (15) INIT( 16 ).
2 PTR PTR INIT(NULL()).
2 RTN_CODE FIXED BIN (15) INIT (0). /*O=O.K.*/
2 KEY BIT(32); /*KEY VALUE RETURNED IN A FILE INIT*/
/ /
DCL 1 NINIT_ARG BASED(P).
2 LEN FIXED BIN (15) INIT(13).
2 CBTP FIXED BIN (15) INIT( 17 ).
2 PTR PTR INIT(NULL()).
2 OP CHAR (1). /*'I'=-INIT. 'S'=-STORE KEY*/
2 FNAME CHAR(8). /*NAME OF FILE*/
2 KEY BIT(32);
DCL 1 NINIT_RTN BASED(P).
2 LEN FIXED BIN (15) INIT(6).
2 CBTP FIXED BIN (15) INIT( 18 ).
2 PTR PTR INIT(NULL()).
2 RTN_CODE FIXED BIN (15) INIT (0). /*O=O.K.*/
2 KEY BIT(32); /*KEY VALUE RETURNED IN A FILE INIT*/
/ /
DCL 1 REP_ARG BASED(P).
2 LEN FIXED BIN (15) INIT (108).
2 CBTP FIXED BIN (15) INIT ( 1 ).
2 PTR PTR INIT (NULL).
/*-----*/
2 ID BIT (32).
2 N FIXED BIN. /*LEN OF DATA IN BIT*/
2 DATA BIT(800); /*UP TO 100 BYTES LONG*/
DCL 1 REP_RTN BASED(P).
2 LEN FIXED BIN (15) INIT ( 2 ).
2 CBTP FIXED BIN (15) INIT ( 2 ).
2 PTR PTR INIT (NULL).
/*-----*/
2 RTN_CODE FIXED BIN INIT (0);
/ /
DCL 1 RET_ARG BASED(P).
2 LEN FIXED BIN (15) INIT (4).
2 CBTP FIXED BIN (15) INIT( 19 ).
2 PTR PTR INIT (NULL).
2 ID BIT (32);
DCL 1 RET_RTN BASED(P).
2 LEN FIXED BIN (15) INIT (104).
2 CBTP FIXED BIN (15) INIT( 20 ).
2 PTR PTR INIT (NULL).
2 RTN_CODE FIXED BIN INIT (0).
2 N FIXED BIN. /*LEN OF DATA IN BIT*/

```

FILE: DATA MACLIB A VM/SP CONVERSATIONAL MONITOR SYSTEM

```

/ /
2 DATA BIT(800); /*UP TO 100 BYTES LONG*/
F1A00140

DCL 1 RETE_ARG BASED(P),
2 LEN FIXED BIN (15) INIT (61),
2 CBTP FIXED BIN (15) INIT( 21 ),
2 PTR PTR INIT (NULL),
2 NODE FIXED BIN (15) INIT(1), /*NODE NUMBER*/
2 NAME CHAR (8), /*ANAME OR ENAME*/
2 PARENT FIXED BIN (15) INIT(0), /*PARENT NODE NUMBER*/
2 GET BIT(8) INIT ('0000000'B), /*SEE RETEOP FOR DETAIL*/
2 N FIXED BIN INIT(0), /*NUMBER OF RELEVANT 'OR' PREDICATES*/
2 PRED (1)/*IN THIS VERSION ONLY ONE PREDICATE EACH TIME IS
ALLOWED; IN THE FUTURE MAY USE REFER OPTION TO N*/.
3 CN FIXED BIN (15) INIT(1), /*CONDITION NUMBER*/
3 SETOP BIT(8), /*RELEVANT IF BSET IF MULTI-TARGET;
NOT USED IN CURRENT VERSION*/
3 OP BIT(8), /*COMPARISON OPS. SEE RETEOP FOR DETAIL*/
3 OLEN FIXED BIN (15) INIT(0), /* LEN OF COMPARE DATA IN BYTES*/
3 CDATA CHAR (40); /*40 BYTES MAXIMUM*/
DCL 1 RETE_ARG1 BASED (P), /*USED FOR 'BELONG' COMP OP*
/*NOT USED IN CURRENT VERSION*/
2 LEN FIXED BIN INIT (8 ), /*PLUS VALUE LEN*/
2 CBTP FIXED BIN (15) INIT( 23 ),
2 PTR PTR, /*POINT TO THE NEXT RETE_ARG*/
2 N FIXED BIN, /* # OF ELEMENTS IN THIS IMMED. VALUE SET*/
2 VALUE (1 REFER(RETE_ARG1.N)) CHAR(40); /*MAX OF 40 BYTES*/
DCL RETEVALUE_LEN FIXED BIN (15) INIT (40);
DCL 1 RETEGET,
2 ANY BIT(8) INIT ('00000001'B),
2 ALL BIT (8) INIT ('00000011'B),
2 NO BIT (8) INIT ('00000000'B); /*NOTE ROOT NODE CANNOT
ASSUME THIS VALUE*/
DCL 1 RETEOP,
2 EQ BIT (8) INIT ('00000000'B),
2 GT BIT(8) INIT ('00000010'B),
2 LT BIT(8) INIT ('00000001'B),
2 BEL BIT(8) INIT ('00000011'B),
2 NULL BIT (8) INIT ('00000100'B); /*NOT SPECIFIED*/
DCL 1 RETE_RTN BASED(P),
2 LEN FIXED BIN (15) INIT (4),
2 CBTP FIXED BIN (15) INIT( 22 ),
2 PTR PTR INIT (NULL),
2 RTN_CODE FIXED BIN INIT (0),
2 N FIXED BIN (15); /*NUMBER OF INSTANCES OF ROOT NODE RETURNED*/
DCL 1 RETE_RTN1 BASED(P),
2 LEN FIXED BIN (15) INIT (46),
2 CBTP FIXED BIN (15) INIT( 24 ),
2 PTR PTR INIT (NULL),
2 NODE FIXED BIN (15), /*NODE NUMBER*/
2 INDX FIXED BIN (15), /*IF MULTI OCCUR*/
2 OLEN FIXED BIN, /*LEN OF DATA IN BYTES*/
2 CDATA CHAR(40); /*DATA RETURNED*/
/ /
DCL 1 RETN_ARG BASED(P),
2 LEN FIXED BIN (15) INIT (61),
F1A00020
F1A00030
F1A00040
F1A00050
F1A00060
F1A00070
F1A00080
F1A00090
F1A00100
F1A00110
F1A00120
F1A00130
F1A00140
F1A00150
F1A00160
F1A00170
F1A00180
F1A00190
F1A00200
F1A00210
F1A00220
F1A00230
F1A00240
F1A00250
F1A00260
F1A00270
F1A00280
F1A00290
F1A00300
F1A00310
F1A00320
F1A00330
F1A00340
F1A00350
F1A00360
F1A00370
F1A00380
F1A00390
F1A00400
F1A00410
F1A00420
F1A00430
F1A00440
F1A00450
F1A00460
F1A00470
F1A00480
F1A00490
F1A00500
F1A00510
F1A00020
F1A00030

```

```

2 CBTP FIXED BIN (15) INIT( 25 ).
2 PTR PTR INIT (NULL).
2 NODE FIXED BIN (15) INIT(1). /*NODE NUMBER*/
2 PSETID BIT(32). /*PSETID OF THE NODE*/
2 PARENT FIXED BIN (15) INIT(0). /*PARENT NODE NUMBER*/
2 BSETID BIT(32). /*BSETID OF THE PARENT PSET*/
2 GET BIT(8) INIT ('0000000'B). /*SEE RETNGET FOR DETAIL*/
2 N FIXED BIN INIT(0). /*NUMBER OF RELEVANT 'OR' PREDICATES*/
2 PRED (1). /*IN CURRENT VERSION ONLY ONE PREDICATE EACH TIME
    IS ALLOWED; FUTURE VERSION MAY USE REFER OPTION TO N*/
3 CN FIXED BIN (15). /*CONDITION NUMBER*/
3 SETOP BIT(8). /*RELEVANT IF BSET IF MULTI-TARGET;
    NOT USED IN CURRENT VERSION*/
3 OP BIT(8). /*COMPARISON OPS. SEE RETNOP FOR DETAIL*/
3 DLEN FIXED BIN (15) INIT(0). /*LEN OF COMPARE DATA IN BYTES*/
3 DATA BIT(320); /*40 BYTES MAXIMUM*/
DCL RETNPRED_LEN FIXED BIN INIT (46);
DCL 1 RETN_ARG1 BASED (P). /*USED FOR 'BELONG' COMP OP*
    *NOT USED IN CURRENT VERSION*/
2 ARG_LEN FIXED BIN INIT (8 ). /*PLUS VALUE LEN*/
2 PTR PTR. /*POINT TO THE NEXT RETN_ARG*/
2 N FIXED BIN. /*# OF ELEMENTS IN THIS IMMED. VALUE SET*/
2 VALUE (1 REFER(RETN_ARG1.N)) BIT(320); /*MAX OF 40 BYTES*/
DCL RETNVALUE_LEN FIXED BIN (15) INIT (40);
DCL 1 RETNGET,
    2 ANY BIT(8) INIT ('00000001'B),
    2 ALL BIT (8) INIT ('00000011'B),
    2 NO BIT (8) INIT ('00000000'B); /*NOTE ROOT NODE CANNOT
    ASSUME THIS VALUE*/

DCL 1 RETNOP,
    2 EQ BIT (8) INIT ('00000000'B),
    2 GT BIT(8) INIT ('00000010'B),
    2 LT BIT(8) INIT ('00000001'B),
    2 BEL BIT(8) INIT ('00000011'B),
    2 NULL BIT (8) INIT ('00000100'B); /*NOT SPECIFIED*/
DCL 1 RETN_RTN BASED(P),
    2 LEN FIXED BIN (15) INIT (4),
    2 CBTP FIXED BIN (15) INIT( 26 ),
    2 PTR PTR INIT (NULL),
    2 RTN_CODE FIXED BIN INIT (0),
    2 N FIXED BIN (15); /*NUMBER OF INSTANCES OF ROOT NODE RETURNED*/
DCL 1 RETN_RTN1 BASED(P),
    2 LEN FIXED BIN (15) INIT (46),
    2 CBTP FIXED BIN (15) INIT( 27 ),
    2 PTR PTR INIT (NULL),
    2 NODE FIXED BIN (15). /*NODE NUMBER*/
    2 INDX FIXED BIN (15). /*IF MULTI OCCUR*/
    2 DLEN FIXED BIN. /*LEN OF DATA IN BYTES*/
    2 DATA BIT(320); /*DATA RETURNED*/

/ /
DCL 1 SHWE_ARG BASED(SHWE_ARG_P),
    2 CTL,
    3 LEN FIXED BIN (15) INIT (210),
    3 CBTP FIXED BIN (15) INIT (36),
    3 PTR PTR INIT (NULL),

```

PAGE 006

VM/SP CONVERSATIONAL MONITOR SYSTEM

```

FILE: DATA      MACLIB  A
2 DATA.
3 NAME1 CHAR (8).
3 NAME2(25) CHAR (8).
3 NUM_NAME2 FIXED BIN;

DCL 1 SHWE_RTN BASED(SHWE_RTN_P).
2 CTL.
3 LEN      FIXED BIN (15) INIT (2).
3 CBTP FIXED BIN (15) INIT (37).
3 PTR PTR INIT (NULL).
2 DATA.
3 RTN_CODE FIXED BIN INIT (0);

OCL 1 SHWE_RTN1 BASED (SHWE_RTN_P).
2 CTL.
3 LEN FIXED BIN (15) INIT (29).
3 CBTP FIXED BIN (15) INIT (38).
3 PTR PTR INIT (NULL).
2 DATA.
3 ANAME CHAR (8).
3 ENAME CHAR (8). /* THE FOLLOWING SAME AS AINFO */
3 MAX FIXED BIN (31).
3 MIN FIXED BIN (31).
3 MLN FIXED BIN.
3 TYPE CHAR (1).
3 FUNC CHAR (1).
3 VTYPE CHAR (1);
DCL (SHWE_ARG P. SHWE_RTN P) PTR;
DCL 1 SHWE_RTN2 BASED_(SHWE_RTN_P).
2 CTL.
3 LEN FIXED BIN INIT (202).
3 CBTP FIXED BIN INIT (39).
3 PTR PTR INIT (NULL).
2 DATA.
3 NUM ENAME FIXED BIN.
3 ENAME (25) CHAR (8);

/ /
DCL 1 UPDE_ARG BASED(P).
2 LEN      FIXED BIN (15) INIT (56).
2 CBTP FIXED BIN (15) INIT ( 28 ).
2 PTR PTR INIT (NULL).
2 NODE FIXED BIN (15) INIT(1). /*NODE NUMBER OF UPDE TREE*/
2 NAME CHAR(8). /*ENAME OR ANAME*/
2 PARENT FIXED BIN (15) INIT(0). /*PARENT NODE NUMBER*/
2 OP FIXED BIN (15). /*OP CODE FOR UPDE: SEE UPDE.OP FOR DETAIL*/
2 DLEN FIXED BIN (15) INIT (0). /*IN NUMBER OF BYTES*/
2 COATA CHAR(40); /*MAX 40 BYTES OF IMMED DATA */
/*THE FOLLOWING NOT USED CURRENTLY
2 DLEN2 FIXED BIN (15).
2 DATA2 BIT (320).
2 DTYPE2 CHAR (1);*/
DCL 1 UPDEOP.
3 ID FIXED BIN INIT(1).
3 1ST FIXED BIN INIT(2). /*INSERT ATTR*/
3 ADD FIXED BIN INIT(3). /*ADD ATTR*/

```

F1A00070
 F1A00080
 F1A00090
 F1A00100
 F1A00110
 F1A00120
 F1A00130
 F1A00140
 F1A00150
 F1A00160
 F1A00170
 F1A00180
 F1A00190
 F1A00200
 F1A00210
 F1A00220
 F1A00230
 F1A00240
 F1A00250
 F1A00260
 F1A00270
 F1A00280
 F1A00290
 F1A00300
 F1A00310
 F1A00320
 F1A00330
 F1A00340
 F1A00350
 F1A00360
 F1A00370
 F1A00380
 F1A00390
 F1A00400
 F1A00410
 F1A00420
 F1A00020
 F1A00030
 F1A00040
 F1A00050
 F1A00060
 F1A00070
 F1A00080
 F1A00090
 F1A00100
 F1A00110
 F1A00120
 F1A00130
 F1A00140
 F1A00150
 F1A00160
 F1A00170
 F1A00180
 F1A00190

PAGE 007

FILE: DATA MACLIB A VM/SP CONVERSATIONAL MONITOR SYSTEM

```

3 DEL FIXED BIN INIT(4). /*DEL ATTR*/
3 SUB FIXED BIN INIT(6). /*SUBTRACT ATTR*/
3 REP FIXED BIN INIT(6). /*REPLACE ATTR*/
3 MOD FIXED BIN INIT(12). /*MODIFY ENTITY*/
3 CRT FIXED BIN INIT(13). /*CREATE ENTITY*/
3 DEE FIXED BIN INIT(11). /*DELETE ENTITY*/
3 NULL FIXED BIN INIT(0). /*NOT ASSIGNED BY USER*/

DCL 1 UPDN_RTN BASED(P).
2 LEN FIXED BIN (15) INIT(2).
2 CBTP FIXED BIN (15) INIT( 29 ).
2 PTR PTR INIT (NULL).
2 RTN_CODE FIXED BIN (15) INIT(0); /*O-D.K.*/

/ /
DCL 1 UPDN_ARG BASED(P).
2 LEN FIXED BIN (15) INIT( 56 ).
2 CBTP FIXED BIN (15) INIT( 30 ).
2 PTR PTR INIT (NULL).
2 NODE FIXED BIN (15) INIT(1). /*NODE NUMBER OF UPDN TREE*/
2 PSETID BIT (32). /*PSETID OF THE NODE*/
2 PARENT FIXED BIN (15) INIT(0). /*PARENT NODE NUMBER*/
2 BSETID BIT (32). /*BSETID OF PARENT PSET LEADING HERE*/
2 OP FIXED BIN (15). /*OP CODE FOR UPDN; SEE UPDNOP FOR DETAIL*/
2 DLEN FIXED BIN (15) INIT(0). /*IN NUMBER OF BYTES*/
2 DATA BIT(320); /*MAX 40 BYTES OF IMMED DATA */
/*THE FOLLOWING NOT USED CURRENTLY
2 DLEN2 FIXED BIN (15).
2 DATA2 BIT (320).
2 DTYPE2 CHAR (1);*/

DCL 1 UPDNOP.
3 ID FIXED BIN INIT(1).
3 IS1 FIXED BIN INIT(2). /*INSERT ARC*/
3 ADD FIXED BIN INIT(3). /*ADD ARC-FOR MULTI-TARG*/
3 DEL FIXED BIN INIT(4). /*DEL ARC*/
3 SUB FIXED BIN INIT(5). /*SUBTRACT ARC-FOR MULTI-TARG*/
3 REP FIXED BIN INIT(6). /*REPLACE ARC*/
3 CI FIXED BIN INIT(14). /*CREATE TARGET NODE THEN INST ARC*/
3 DD FIXED BIN INIT(15). /*DELETE ARC AND TARGET NODE TOO*/
3 REPP FIXED BIN INIT(16). /*REPLACE VALUE OF TARG NODE*/
3 NULL FIXED BIN INIT(0). /*RESERVED*/
3 MOD FIXED BIN INIT(12). /*MODIFY TREE--ROOT NODE OP ONLY*/
3 CRT FIXED BIN INIT(13). /*CREATE TREE--ROOT NODE OP ONLY*/
3 DEE FIXED BIN INIT(11). /*DELETE TREE--ROOT NODE OP ONLY*/

DCL 1 UPDN_RTN BASED(P).
2 LEN FIXED BIN (15) INIT(2).
2 CBTP FIXED BIN (15) INIT( 31 ).
2 PTR PTR INIT (NULL).
2 RTN_CODE FIXED BIN (15) INIT(0); /*O-D.K.*/

/ /
DCL 1 VINIT_ARG BASED(P).
2 LEN FIXED BIN (15) INIT(13).
2 CBTP FIXED BIN (15) INIT( 32 ).
2 PTR PTR INIT (NULL).
2 OP CHAR(1). /*NOT USED IN CURRENT VERSION*/
2 FNAME CHAR(8).
2 KEY BIT (32); /*NOT USED IN CURRENT VERSION*/

```

F1A00200
 F1A00210
 F1A00220
 F1A00230
 F1A00240
 F1A00250
 F1A00260
 F1A00270
 F1A00280
 F1A00290
 F1A00300
 F1A00310

 F1A00020
 F1A00030
 F1A00040
 F1A00050
 F1A00060
 F1A00070
 F1A00080
 F1A00090
 F1A00100
 F1A00110
 F1A00120
 F1A00130
 F1A00140
 F1A00150
 F1A00160
 F1A00170
 F1A00180
 F1A00190
 F1A00200
 F1A00210
 F1A00220
 F1A00230
 F1A00240
 F1A00250
 F1A00260
 F1A00270
 F1A00280
 F1A00290
 F1A00300
 F1A00310
 F1A00320
 F1A00330
 F1A00340
 F1A00350

 F1A00020
 F1A00030
 F1A00040
 F1A00050
 F1A00060
 F1A00070
 F1A00080

PAGE 008

FILE: DATA MACLIB A VM/SP CONVERSATIONAL MONITOR SYSTEM

```

DCL 1 VINIT_RTN BASED(P),
  2 LEN FIXED BIN (15) INIT (6 ),
  2 CBTP FIXED BIN (15) INIT( 33 ),
  2 PTR PTR INIT (NULL),
  2 RTN CODE FIXED BIN INIT (0 ),
  2 KEY_BIT(32);
/
DCL 1 VNAME_ARG BASED(P),
  2 LEN FIXED BIN (15) INIT (260),
  2 CBTP FIXED BIN (15) INIT( 34 ),
  2 PTR PTR INIT (NULL),
  2 T,
  3 ENAME CHAR(8),
  3 TN FIXED BIN,
  3 T1 (25),
  4 PARENT FIXED BIN,
  4 ANAME CHAR(8);
DCL 1 VNAME_RTN BASED(P),
  2 LEN FIXED BIN (15) INIT (4),
  2 CBTP FIXED BIN (15) INIT( 35 ),
  2 PTR PTR INIT (NULL),
  2 RTN CODE FIXED BIN INIT (0),
  2 N FIXED BIN;
/
/*DCL OF BCAT TEMPLATE*/
DCL 1 BINFO,
  2 PSETID(2) BIT (32),
  2 POS FIXED BIN, /*POS OF POINTER ARRAY OF PSETID1 USED */
  2 FUNC CHAR (1); /*'S' OR 'M'*/;
/
/*DCL OF BASIC ENCODING UNIT (BEU) USED IN THE N-ARY LEVEL***/
DCL 1 BEU BASED(P),
  2 NUMPTR FIXED BIN,
  2 ID_ARRAY(I REFER (BEU.NUMPTR))BIT(32) INIT ((1)UNSPEC(NULL)),
  2 DLEN FIXED BIN,
  2 DATA BIT(J REFER (BEU.DLEN));
/
/* CATALOGUE STRUCTURES AT THE ENTITY LEVEL*****/
DCL 1 AINFO,
  2 BSETID BIT (32), /*BSETID OF THIS ATTRIBUTE**/
  2 PSETID BIT (32), /*PSETID OF TARGET NODE**/
  2 ENAME CHAR (8), /*IF TARGET IS A ENTITY SET*/
  2 MAX FIXED BIN (31),
  2 MIN FIXED BIN (31),
  2 MLEN FIXED BIN,
  2 FUNC CHAR(1),
  2 TYPE CHAR (1),
  2 VTYPE CHAR (1);
/
/* DCL OF STATIC KEYS USED AS THE ENTITY LEVEL *****/
/* ONLY MODULE VINIT HAS THE WRITE ACCESS: OTHER MODULES R/O**/
DCL KEY (11) BIT (32) EXT STATIC;
DCL 1 KY /*INDEX OF 11 KEYS*/ STATIC,
  2 PESET FIXED BIN INIT (1),

```

```

F1A00090
F1A00100
F1A00110
F1A00120
F1A00130
F1A00140

F1A00020
F1A00030
F1A00040
F1A00050
F1A00060
F1A00070
F1A00080
F1A00090
F1A00100
F1A00110
F1A00120
F1A00130
F1A00140
F1A00150
F1A00160
F1A00170

F1D00020
F1D00030
F1D00040
F1D00050
F1D00060

F1D00020
F1D00030
F1D00040
F1D00050
F1D00060
F1D00070

F1D00020
F1D00030
F1D00040
F1D00050
F1D00060
F1D00070
F1D00080
F1D00090
F1D00100
F1D00110
F1D00120
F1D00130

F1D00020
F1D00030
F1D00040
F1D00050
F1D00060

```

PAGE 009

FILE: DATA MACLIB A VM/SP CONVERSATIONAL MONITOR SYSTEM

```

2 PENAME FIXED BIN INIT (2).
2 PEINFO FIXED BIN INIT (3).
2 PASET FIXED BIN INIT (4).
2 PANAME FIXED BIN INIT (5).
2 PAINFO FIXED BIN INIT (6).
2 BENAME FIXED BIN INIT (7).
2 BEINFO FIXED BIN INIT (8).
2 BESET FIXED BIN INIT (9).
2 BANAME FIXED BIN INIT (10).
2 BAINFO FIXED BIN INIT (11);

```

```

/*DCL OF KEYS (BCATID AND PCATID) USED AT THE N-ARY LEVEL. ONLY
** NINIT MODULE HAS THE WRITE ACCESS. OTHER MODULES WITHIN N-ARY
** LEVEL MAY HAVE READ ACCESS*****
DCL (PCATID,BCATID) BIT (32) EXTERNAL STATIC;

```

```

/*DCLS OF PSET CATALOGUE AT THE N-ARY LEVEL*****/
DCL 1 PINFO,
3 NUMPTR FIXED BIN,
3 PLEN /*IN NUMBER OF BYTES*/ FIXED BIN,
3 LTYPE BIT (8), /*'N' OR 'X' OR 'B' OR 'C'*/
3 LID BIT (32), /*ID OF THE FIRST BEU IN THIS SET*/
3 L_POS FIXED BIN, /*POS IN ID ARRAY USED FOR LINEAR CHAINING
                     OF BEU'S WITHIN THIS SET*/
3 L_POS2 FIXED BIN,
3 MAP BIT (32); /*UP TO 16 PTRS ARE ALLOWED FOR A BEU*/
DCL 1 PINFOLTYPE,
2 LL BIT (8) INIT ('00000001'B);

```

```

/*DCL OF SEU (STORAGE ENCODING UNIT) USED AT THE MM LEVEL AND THE
** STORAGE HIERARCHY VIRTUAL MEMORY*****
DCL SH AREA (30000) BASED(SH_BASE),
SH_BASE PTR EXT;
DCL 0 OFFSET (SH);
DCL 1 SEU BASED (0),
2 MOVE_PTR OFFSET(SH) INIT (NULL()),
2 N FIXED BIN,
2 INVAL BIT(8) INIT ('00000000'B),
2 DATA BIT (1 REFER (SEU.N));

```

```

DCL 1 FDEBUG,
2 LEVO BIT (1) INIT ('1'B),
2 LEV1 BIT (1) INIT ('1'B),
2 LEV2 BIT(1) INIT ('1'B),
2 LEV3 BIT(1) INIT ('1'B),
2 LEV4 BIT(1) INIT ('1'B),
2 LEV5 BIT(1) INIT ('1'B);

```

```

/*DCL OF SEU (STORAGE ENCODING UNIT) USED AT THE MM LEVEL AND THE
** STORAGE HIERARCHY VIRTUAL MEMORY: FOR INTEGRATED VERSION *****/
DCL 1 SEU BASED (SEUPTR),
/* 2 MOVE_PTR OFFSET(SH) INIT (NULL()), NOT USED CURRENTLY */

```

```

F1D00070
F1D00080
F1D00090
F1D00100
F1D00110
F1D00120
F1D00130
F1D00140
F1D00150
F1D00160
F1D00170

```

```

F1D00020
F1D00030
F1D00040
F1D00050
F1D00060
F1D00070
F1D00080
F1D00090
F1D00100
F1D00110
F1D00120
F1D00130
F1D00140

```

```

F1D00020
F1D00030
F1D00040
F1D00050
F1D00060
F1D00070
F1D00080
F1D00090
F1D00100
F1D00110
F1D00120
F1D00130
F1D00140

```

```

F1D00020
F1D00030
F1D00040
F1D00050
F1D00060
F1D00070
F1D00080
F1D00090
F1D00100
F1D00110
F1D00120

```

```

FDE00020
FDE00030
FDE00040
FDE00050
FDE00060
FDE00070
FDE00080

```

```

DSE00020
DSE00030
DSE00040
DSE00050

```

VLM/SP CONVERSATIONAL MONITOR SYSTEM

FILE: DATA MACLIB A

```

2 N FIXED BIN,
2 INVAL CHAR(1) INIT ('N'),
2 DATA CHAR(1 REFER (SEU.N)),
( (SEUPTR) PTR;
/
DCL 1 ARCH(19),
2 LEVEL FIXED BIN(31) INIT (1,2,2,2,2,2,
2,2,2,3,3,
3,3,3,4,4,
4,4,4,5),
2 PROCNAME CHAR (8) INIT ('USER','VINIT','VNAME','DEFE','RETE',
'UPDE','DEFA','SHWE','DEFP','DEFB',
'RETN','UPDN','NINIT','RET','DEL',
'REP','CRT','MINIT','L1');
/
/
/* SIMULATED STORAGE HIERARCHY, KNOWN TO DUMFSTV, DUML1,FHL1 */
DCL SH AREA (30000) BASED (SH_BASE);
DCL SH_BASE PTR EXT;
DCL PACKET(3000) CHAR (8) BASED(PTR), PP OFFSET(SH) EXT;
/
/
/* THIS IS THE PROC_TBL TO BE INCLUDED IN THE SVCS1 ROUTINE **/
DCL 1 PROC_TBL (18),
2 N CHAR(7) INIT ('VINIT','DEFA','DEFE','RETE','UPDE','SHWE',
'VNAME',
'RETN','UPDN','DEFP','DEFB','NINIT',
'RET','REP','CRT','DEL','MINIT','USER'),
2 LEVEL FIXED BIN INIT ((7)6,(5)5,(5)0,7);
/* NOTE THAT T-PROC'S 1-7 ARE IN THE ENTITY LEVEL (6) AND 8-12
ARE IN INTERNAL SCHEMA LEVEL (ASSIGNED LEVEL 5) AND T-PROC'S
13-17 ARE IN THE MEMORY MGMT LEVEL (ASSIGNED LEVEL 0). 'L1'
IS THE INTERFACE MODULE AT THE SH-STV (ASSIGNED LEVEL 1). */
DCL PROC_TBL_LEN FIXED BIN INIT (18);
/
/
ACRT ADEFA ADEFB ADEFP ADEFP
ADEL AMINI >ANINI ARET
ARETE YARETN ASHWE AUPDE AUPDN
#AVNAME #AVNIT 10RCAT DBEU DECAT
DKKEY DNKEY EDPCAT DSEU QFDEB
DSEU1 FSTRUC 5SH SVGTB SVGTB
DSEU0060
DSEU0070
DSEU0080
DSEU0090
DSEU0100
DSEU0110
DSEU0120
DSEU0130
FIFO0020
FIFO0030
FIFO0040
FIFO0050
FIFO0060
FIFO0070
FIFO0080
FIFO0090
FIFO0100
FISO0020
FISO0030
FISO0040
FISO0050
SVC00010
SVC00020
SVC00030
SVC00040
SVC00050
SVC00060
SVC00070
SVC00080
SVC00090
SVC00100
SVC00110
SVC00120
SVC00130

```

PAGE 001

FILE: ENTRY MACLIB A VM/SP CONVERSATIONAL MONITOR SYSTEM

LIBPDS

```

DCL VTPE ENTRY /*ENTRY DCL FOR MODULE VTPE*/
  (1,2 BIT(32), 2 BIT(32), 2 CHAR(8),
   2 FIXED BIN (31), 2 FIXED BIN (31), 2 FIXED BIN,
   2 CHAR (1), 2 CHAR(1), 2 CHAR (1), FIXED BIN,
   BIT (320), CHAR(40), FIXED BIN, FIXED BIN);
//
DCL ALLA ENTRY /*ENTRY DCL FOR MODULE ALLA*/
  (CHAR (8), (25) CHAR (8), FIXED BIN, FIXED BIN );
//
DCL GBCT ENTRY /*ENTRY DCL FOR MODULE GBCT WHICH RETRIEVES BIN*/
  (BIT(32), 1,2 (2) BIT(32), 2 FIXED BIN, 2 CHAR(1), FIXED BIN);
//
/*ENTRY DCL OF MODULE BNV*/
DCL BNV ENTRY;
//
DCL CRTB ENTRY /*ENTRY DCL FOR MODULE CRTB*/
  (BIT(32), BIT(32), BIT(32), BIT(*), CHAR(1), FIXED BIN);
//
DCL CRTN ENTRY /*ENTRY DCL FOR MODULE CRTN*/
  (PTR, (25) BIT(1), FIXED BIN);
//
DCL CRTP ENTRY /*ENTRY DCL FOR MODULE CRTP*/
  (BIT(32), BIT(*), BIT (32), FIXED BIN);
//
/*ENTRY DECL OF DBA MODULE*/
DCL DBA ENTRY;
//
DCL DOB ENTRY; /*ENTRY DCL FOR MODULE DOB*/
//
DCL DDV ENTRY; /*ENTRY DCL FOR MODULE DDV*/
//
DCL DELB ENTRY /*ENTRY DCL FOR MODULE DELB*/
  (BIT(32), BIT(32), BIT(32), CHAR(1), FIXED BIN);
//
DCL DELN ENTRY /*ENTRY DCL FOR MODULE DELN*/
  (PTR, (25) BIT (1), FIXED BIN);
//
DCL DELP ENTRY /*ENTRY DCL FOR MODULE DELP*/
  (BIT(32), BIT(32), FIXED BIN);
//
DCL DFB1 ENTRY /*ENTRY DCL FOR MODULE DFB1*/
  (CHAR(1), BIT (32), BIT(32), FIXED BIN, BIT(32));
//
DCL DFP1 ENTRY (CHAR(1), FIXED BIN, CHAR(1), FIXED BIN, BIT(32));
//
DCL DMB ENTRY; /*ENTRY DCL FOR MODULE DMB*/
//
DCL GACT ENTRY /*ENTRY DCL FOR MODULE GACT*/
  (CHAR(8), CHAR(8), 1,2 BIT(32), 2 BIT(32), 2 CHAR(8),
   2 FIXED BIN (31), 2 FIXED BIN (31), 2 FIXED BIN,
   2 CHAR (1), 2 CHAR(1), 2 CHAR (1), FIXED BIN);
//
DCL GECT ENTRY /*ENTRY DCL FOR MODULE GECT*/
  (CHAR(8), BIT(32), FIXED BIN);

```

F1E00020

F1E00030

F1E00040

F1E00050

F1E00060

F1E00020

F1E00030

F1E00020

F1E00030

F1E00020

F1E00030

F1E00020

F1E00030

F1E00020

F1E00030

F1E00020

F1E00030

F1E00020

F1E00030

F1E00020

F1E00030

F1E00020

F1E00030

F1E00020

F1E00030

F1E00020

F1E00030

F1E00020

F1E00030

F1E00020

F1E00030

F1E00020

F1E00030

F1E00020

F1E00030

F1E00020

F1E00030

F1E00020

F1E00030

PAGE 002

FILE: ENTRY MACLIB A VM/SP CONVERSATIONAL MONITOR SYSTEM

```

/ / DCL LEX ENTRY /*ENTRY DCL FOR MODULE LEX*/
      (FIXED BIN, 1 (25), 2 CHAR (40), 2 FIXED BIN, FIXED BIN,
      FIXED BIN);
/ /
/ / DCL LEXT ENTRY /*ENTRY DCL FOR MODULE LEXT*/
      (FIXED BIN, 1 (25), 2 CHAR (40), 2 FIXED BIN, 1 (25),
      2 FIXED BIN, 2 CHAR (8), FIXED BIN, 1 (25), 2 CHAR (1),
      2 CHAR (40), 2 FIXED BIN, (25) CHAR(1), FIXED BIN);
/ /
/ / DCL MOON ENTRY /*ENTRY DCL FOR MODULE MOON*/
      (PTR,(25) BIT(1), FIXED BIN);
/ /
/ / DCL MPST ENTRY /*ENTRY DCL FOR MODULE MPST*/
      (BIT(32),FIXED BIN);
/ /
/ /*ENTRY DCL FOR PCAT SERVICE SUBROUTINES*/
DCL GPCT ENTRY /*ENTRY DCL FOR MODULE GPCT:GET PCAT*/
      (BIT(32), 1,2 FIXED BIN, 2 FIXED BIN, 2 CHAR(1), 2 BIT (8),
      2 BIT (32), 2 FIXED BIN, 2 FIXED BIN, 2 BIT (32), FIXED BIN);
DCL UPCT ENTRY /*ENTRY DCL FOR MODULE UPCT:UPDATE PCAT*/
      (BIT(32), 1,2 FIXED BIN, 2 FIXED BIN, 2 CHAR(1), 2 BIT (8),
      2 BIT (32), 2 FIXED BIN, 2 FIXED BIN, 2 BIT (32), FIXED BIN);
/ /
/ / DCL PRNT ENTRY /*ENTRY DCL FOR MODULE PRINT*/
      (PTR,PTR,FIXED BIN);
/ /
/ /*ENTRY DCL OF MODULE RELV*/
DCL RELV ENTRY;
/ /
/ / DCL REPB ENTRY /*ENTRY DCL FOR MODULE REPB*/
      (BIT(32),BIT(32),BIT(32),BIT(*),CHAR(1),FIXED BIN);
/ /
/ / DCL SRCH ENTRY /*ENTRY DCL FOR MODULE SRCH*/
      (BIT(32),BIT(*),BIT (32),FIXED BIN);
/ /
/ / DCL RET1 ENTRY /*ENTRY DCL FOR N-ARY LOW LEVEL SUBROUTINE RET1*/
      (BIT(32),PTR,FIXED BIN);
DCL DEL1 ENTRY /*ENTRY DCL FOR N-ARY LOW LEVEL SUBROUTINE DEL1*/
      (BIT(32), FIXED BIN);
DCL REP1 ENTRY /*ENTRY DCL FOR N-ARY LOW LEVEL SUBROUTINE REP1*/
      (BIT(32),PTR, FIXED BIN);
DCL CRT1 ENTRY /*ENTRY DCL FOR N-ARY LOW LEVEL SUBROUTINE CRT1*/
      (BIT (32), PTR, FIXED BIN);
/ /
/ / DCL UPE1 ENTRY /*ENTRY DCL FOR MODULE UPE1*/
      (CHAR(1),1,2 CHAR(8),2 FIXED BIN,2 (25),3 FIXED BIN,3 CHAR(8),
      (25) CHAR(1), FIXED BIN);
/ /
/ /*DCL OF ENTRY FOR USER */
DCL USER ENTRY;
/ /
/ / DCL VNM1 ENTRY /*ENTRY DCL FOR MODULE VNM1*/

```

F1E00020
F1E00030
F1E00040F1E00020
F1E00030
F1E00040
F1E00050
F1E00060F1E00020
F1E00030F1E00020
F1E00030F1E00020
F1E00030
F1E00040
F1E00050
F1E00060
F1E00070
F1E00080F1E00020
F1E00030F1E00020
F1E00030F1E00020
F1E00030F1E00020
F1E00030F1E00020
F1E00030F1E00020
F1E00030
F1E00040
F1E00050
F1E00060
F1E00070
F1E00080
F1E00090F1E00020
F1E00030
F1E00040
F1E00050EUS00020
EUS00030

F1E00020

PAGE 003

FILE: ENTRY MACLIB A VM/SP CONVERSATIONAL MONITOR SYSTEM

```

(1,2 CHAR(8),2 FIXED BIN,2 (25),3 FIXED BIN,3 CHAR(8),
  FIXED BIN, FIXED BIN);

/*SERVICE ROUTINES DCLS****/
DCL TCALL ENTRY (CHAR(7), FIXED BIN, PTR, PTR);
DCL TRIN ENTRY (1, 2 FIXED BIN, 2 FIXED BIN, 2 FIXED BIN, PTR);
DCL TBEG ENTRY (1, 2 FIXED BIN, 2 FIXED BIN, 2 FIXED BIN, PTR);
DCL SVC3 ENTRY (PTR);
DCL PRDC NAME CHAR(7);
DCL 1 PROC_ADDR,
  (2 LEVEL,
   2 VPID,
   2 BOXID) FIXED BIN;

/*DCLS OF SVC SEND SERVICES*/
DCL SVCS1 ENTRY (CHAR(7),PTR, FIXED BIN);
DCL SVCS2 ENTRY (1,2 FIXED BIN,2 FIXED BIN,2 FIXED BIN,PTR, FIXED BIN);
DCL 1 MSG BASED (P),
  2 LEN FIXED BIN (15) INIT(12),
  2 DUM CBTP FIXED BIN (15) INIT (0),
  2 PTR_PTR, /*POINT TO ARG LIST*/
  2 RTN_ADDR, /*ADDRESS OF THE PROCESS WHICH INITIATES THE MSG*/
  (3 LEVEL,
   3 VPID,
   3 BOXID) FIXED BIN (15);

DCL PROC_NAME CHAR(7); /*USED IF SVCS1 IS CALLED*/
DCL 1 PROC_ADDR, /*USED IF SVCS2 IS CALLED*/
  (2 LEVEL,
   2 VPID,
   2 BOXID) FIXED BIN (15);

/*
DCL HEX4 EXT ENTRY (PTR) RETURNS(CHAR(8));
/* ROUTINE TO CONVERT A PTR TO HEX STRING */
DCL HEX EXT ENTRY (PTR) RETURNS(CHAR(6));

DCL VERIF ENTRY /*ENTRY DCL FOR MODULE VERIF*/
  ( 1, 2 FIXED BIN, 2 FIXED BIN, 2 CHAR (1), 2 BIT (8),
   2 BIT (32), 2 FIXED BIN, 2 FIXED BIN, 2 BIT (32),
   PTR, BIT(8) ALIGNED,
   BIT(*) VAR ALIGNED,
   FIXED BIN, BIT(1) ALIGNED);

DCL RETB ENTRY /*ENTRY DCL FOR MODULE RETB*/
  (BIT(32) ALIGNED, 1,2 (2) BIT (32),2 FIXED BIN,2 CHAR (1), PTR, PTR,
   FIXED BIN);

DCL RETP ENTRY /*ENTRY DCL FOR MODULE RETP*/
  (1,2 FIXED BIN,2 FIXED BIN,2 CHAR (1), 2 BIT (8), 2 BIT (32),
   2 FIXED BIN, 2 FIXED BIN, 2 BIT (32),
   PTR, PTR, FIXED BIN);

DCL BUILD ENTRY /*ENTRY DCL FOR MODULE BUILD*/
  ((25) PTR, (25) BIT(32) ALIGNED,

```

F1E00030
F1E00040
F1E00050SER00010
SER00020
SER00030
SER00040
SER00050
SER00060
SER00070
SER00080
SER00090
SER00100
SER00110F1E00020
F1E00030
F1E00040
F1E00050
F1E00060
F1E00070
F1E00080
F1E00090
F1E00100
F1E00110
F1E00120
F1E00130
F1E00140
F1E00150
F1E00160
F1E00170HEX00010
HEX00020
HEX00030F1E00020
F1E00030
F1E00040
F1E00050
F1E00060
F1E00070F1E00020
F1E00030
F1E00040F1E00020
F1E00030
F1E00040
F1E00050F1E00020
F1E00030

PAGE 004

FILE: ENTRY MACLIB A VM/SP CONVERSATIONAL MONITOR SYSTEM

```

PTR, FIXED BIN, PTR, FIXED BIN);
/ /
DCL PAKT ENTRY /*ENTRY DCL FOR MODULE PAKT*/
(FIXED BIN(31),CHAR(8),CHAR(1));
/ /
DCL GMEM ENTRY /*ENTRY DCL FOR MODULE GMEM*/
(FIXED BIN,BIT(32));
/ /
DCL GSEU ENTRY /* ENTRY DCL FOR GSEU SUBROUTINE */
(BIT(32),PTR,FIXED BIN);
/ /
DCL PSEU ENTRY /* ENTRY DCL FOR PSEU SUBROUTINE */
(BIT(32),PTR,FIXED BIN);
/ /
EVTPE    EALLA    EBCT    EBNV    ECRTB
ECRTN    ECRTP    EDDB    EDDV
EDELN    EDLP    EDBA    EDDP1
EDMB    EGACT    EDELN    EDEXT
EMOON    EMPST    EGECT    EELX
EREPB    ESRCH    EPCT    EPRNT    ERELV
EVNM1    >SERVICE    IESUB1    )EUPE1    EUSER
ERETB    )ERETP    ESVCS    *HEX    *WEVERI
EPSEU    EPSEU    EBUIL    UEPAKT    YEGMEM
                 EPSEU

```

.

m

APPENDIX 3

PROGRAM LISTING -- FSTV

```

ALLA: PROC (ENAME,ANAMES,NUM_ANAMES,RTN_CODE);
/* A-PROC AT THE ENTITY LEVEL, INVOKED IN RESPONSE TO A CALL TO
RETRIEVE ALL ATTRIBUTE NAMES OF AN ENTITY SET WHOSE NAMES IS GIVEN
AS ENAME. AND THESE ATTRIBUTE NAMES ARE TO BE RETURNED IN THE
ARRAY ANAMES */
XINCLUDE SERVICE, DECAT, ARETE;
DCL (P.LP,TP,PP,RP) PTR, NULL BUILTIN, (RTN_CODE,1) FIXED BIN;
DCL ENAME CHAR (8), ANAMES (25) CHAR (8), NUM_ANAMES FIXED BIN;

/* BEGIN */
  ALLOCATE RETE_ARG; RP,LP=P;
  RETE_ARG.NAME='E*ASET';
  RETE_ARG.GET=RETEGET.ALL;

  DO I = 2 TO 4; /* NODES 2 TO 4 */
    ALLOCATE RETE_ARG; LP->RETE_ARG.PTR=P; LP=P;
    RETE_ARG.NODE=I;
    SELECT (1);
      WHEN (4) DO;
        RETE_ARG.NAME='A*ANAME'; /* ANAME NODE */
        RETE_ARG.GET=RETEGET.ALL;
        RETE_ARG.PARENT=1;
        END /* WHEN 4 */;
      WHEN (2) DO;
        RETE_ARG.NAME='A*ESET';
        RETE_ARG.PARENT=1;
        RETE_ARG.GET=RETEGET.NO;
        END /* WHEN 2 */;
      WHEN (3) DO;
        RETE_ARG.NAME='A*ENAME'; /* ENAME PRED NODE */
        RETE_ARG.GET=RETEGET.NO;
        RETE_ARG.PARENT=2;
        RETE_ARG.N=1; /* 1 PRED */
        RETE_ARG.OP(1)=RETEOP.EQ;
        RETE_ARG.DLEN(1)=8;
        RETE_ARG.CDATA=ENAME;
        END /* WHEN 3 */;
    END /* SELECT */;
    END /* DO I */;

  CALL TCALL ('RETE',1,RP,TP);
  CALL SVC3 (RP);

  /* EXAMINE RETURN */
  P=TP;
  IF RETE_RTN_CODE^=0 THEN DO;
    /**** RTN_CODE 0+ IS FROM RETE CALL *****/
    CALL SVC3 (P); GO TO RTN; END;

  NUM_ANAMES = RETE_RTN.N;

```

```

ALLO0010
ALLO0020
ALLO0030
ALLO0040
ALLO0050
ALLO0060
ALLO0070
ALLO0080
ALLO0090
ALLO0100
ALLO0110
ALLO0120
ALLO0130
ALLO0140
ALLO0150
ALLO0160
ALLO0170
ALLO0180
ALLO0190
ALLO0200
ALLO0210
ALLO0220
ALLO0230
ALLO0240
ALLO0250
ALLO0260
ALLO0270
ALLO0280
ALLO0290
ALLO0300
ALLO0310
ALLO0320
ALLO0330
ALLO0340
ALLO0350
ALLO0360
ALLO0370
ALLO0380
ALLO0390
ALLO0400
ALLO0410
ALLO0420
ALLO0430
ALLO0440
ALLO0450
ALLO0460
ALLO0470
ALLO0480
ALLO0490
ALLO0500
ALLO0510
ALLO0520
ALLO0530
ALLO0540
ALLO0550

```

FILE: ALLA PLIOPT A1 VM/SP CONVERSATIONAL MONITOR SYSTEM

```

IF NUM_ANAMES=0 THEN DO; /* NO ATTRIBUTES DEFINED */
  CALL SVC3 (P); GO TO RTN; END;

/* COPY ATTRIBUTE NAMES FOUND FOR NORMAL PROCESSING */
DO I = 1 TO NUM_ANAMES;
  DO P=RETE_RTN_PTR REPEAT RETE_RTN1_PTR WHILE (P^=NULL);
    IF RETE_RTN1_NODE = 4 /* NODE 2 IS ANAME NODE */
      THEN DO; ANAMES(I)=RETE_RTN1_CDATA;
        GO TO NEXT; /* GET OUT OF P LOOP */
      END;
  END /* P */;
NEXT: END /* DO I */;
CALL SVC3 (TP);
RTN: END /* ALLA */;

```

```

ALLO0560
ALLO0570
ALLO0580
ALLO0590
ALLO0600
ALLO0610
ALLO0620
ALLO0630
ALLO0640
ALLO0650
ALLO0660
ALLO0670
ALLO0680
ALLO0690

```

FILE: BUILD PLIDPT A1 VM/SP CONVERSATIONAL MONITOR SYSTEM

```

BUILD: PROC /* SUBROUTINE USED BY RETN TO BUILD RETURN DATA */
(CURS, PSETID, MP, K, LP, RTN_CODE);

/* MP POINTS TO THE FIRST TOKEN IN RETN TREE, K IS THE OCCUR
NUMBER OF ROOT NODE, LP POINTS TO THE LAST TOKEN IN THE
RETN_RTN LIST UP TO NOW */

DCL CURSOR (25) PTR CONNECTED; /* CURRENT PTR TO NODES */
DCL PSETID(25) BIT(32) ALIGNED CONNECTED; /* PSETID OF RETN TREE NDS */
DCL (P, MP, RTP, TP, RP, LP) PTR, (NULL, UNSPEC) BUILTIN, TP1 PTR;
DCL (RNODE, RTN_CODE) FIXED BIN;
DCL (K, NODE) FIXED BIN;

XINCLUDE DBEU, ARETN, ESUB1;

/* BEGIN PROCESSING */
RNODE = MP -> RETN_ARG.NODE; /* RNODE CONTAINS ROOT NODE NUMBER */
P = MP;

/* ALLOCATE ROOT NODE */
ALLOCATE RETN_RTN1 SET (TP);
TP -> RETN_RTN1.NODE = RETN_ARG.NODE;
TP -> RETN_RTN1.INDEX = K;
TP1 = CURSOR (RNODE);
IF TP1 = NULL THEN DO;
  RTN_CODE = 1; /* BAD ID IN CURSOR TREE */
  FREE TP -> RETN_RTN1;
  RETURN;
END;

TP -> RETN_RTN1.DLEN = TP1 -> BEU.DLEN/8;
TP -> RETN_RTN1.DATA = TP1 -> BEU.DATA;

  LP -> RETN_RTN1.PTR = TP;
  LP = TP;

/* ALLOCATE THE REST */
DO P = RETN_ARG.PTR REPEAT RETN_ARG.PTR WHILE (P ^ NULL);
IF RETN_ARG.GET ^ RETN_GET_NO & CURSOR(RETN_ARG.NODE) ^
  NULL THEN DO /* DO ONLY IF NECESSARY */
  ALLOCATE RETN_RTN1 SET (TP);
  LP -> RETN_RTN1.PTR = TP;
  LP = TP;
  TP -> RETN_RTN1.NODE = RETN_ARG.NODE;
  TP -> RETN_RTN1.INDEX = 1;
  TP1 = CURSOR(RETN_ARG.NODE);
  IF TP1 = NULL THEN DO;
    RTN_CODE = 1; /* IS ILLEGAL CURSOR */
    RETURN;
  END;
  TP -> RETN_RTN1.DLEN = TP1 -> BEU.DLEN/8;
  TP -> RETN_RTN1.DATA = TP1 -> BEU.DATA;
  END /* IF GET */;
END /* DO P = ... */;

```

FILE: BUILD PLIOPT A1

VM/SP CONVERSATIONAL MONITOR SYSTEM

PAGE 002

END /*BUILD*/:

BUI00560

```

CRT:PROC;
/* A PROCEDURE AT THE MEMORY MGMT LEVEL TO CREATE A SEU */
/* FOR INTEGRATED VERSION: MAKE USE OF SUBROUTINES GMEM (GET VIRTUAL */
/* MEMORY) AND PSEU (PUT SEU). */
%INCLUDE DSEU1.ACRT.SERVICE.FOEBUG.EPSEU.EGMEN;
DCL P PTR. (NULL,UNSPEC) BUILTIN;
DCL (I,RTN_CODE) FIXED BIN, (ID,ID1) BIT(32);

CALL TBEG(PROC_ADDR,P);
I=CRT_ARG.N/8;          %INCLUDE HEX;
ALLOCATE SEU;
UNSPEC(SEU.DATA) = CRT_ARG.DATA;

CALL GMEM (I+5,ID);      /* 5: 3 FOR INVAL & N, 2 FOR SAFETY */
ID1=ID;
CALL PSEU (ID1,SEUPTR,RTN_CODE); /* STORE IT */

/*RETURNS*/
ALLOCATE CRT RTN; CRT_RTN.RTN_CODE=0;
CRT_RTN.ID=ID;
CALL TRTN (PROC_ADDR,P);
RETURN;
END/*CRT*/;

```

```

CRT00010
CRT00020
CRT00030
CRT00040
CRT00050
CRT00060
CRT00070
CRT00080
CRT00090
CRT00100
CRT00110
CRT00120
CRT00130
CRT00140
CRT00150
CRT00160
CRT00170
CRT00180
CRT00190
CRT00200
CRT00210
CRT00220
CRT00230
CRT00240
CRT00250

```

FILE: CRTB PLIOPT A1 VM/SP CONVERSATIONAL MONITOR SYSTEM

```

CRTB: PROC (PARENT,BSETID, ID1,DATA,MODE,RTN_CODE);
%INCLUDE DBCAT, DBEU;
%INCLUDE EBCT,ESRCH,ECRTP,ESUB1;

DCL (PARENT, BSETID, ID1,ID2) BIT (32), DATA BIT (*);
DCL RTN_CODE FIXED BIN, MODE CHAR (1), P_PTR, NULL BUILTIN;

CALL GBCT (BSETID, BINFO, RTN_CODE);

SELECT (MODE);

  WHEN ('N') DO; /* MODE MODE OF CREATION OF BINARY ASSOC */

    IF BINFO.FUNC='S' THEN /*CREATE IT*/
      CALL CRTP (BINFO.PSETID(2), DATA, ID2,RTN_CODE);
    ELSE DO; /*SEE IF IT ALREADY EXISTS*/
      CALL SRCH(BINFO.PSETID(2),DATA,ID2,RTN_CODE);
      IF RTN_CODE=1 /*NOT FOUND*/
        THEN CALL CRTP(BINFO.PSETID(2),DATA,ID2,RTN_CODE);
      END;
    END /*WHEN N*/;

  WHEN ('A') DO; /*ARC MODE OF CREATION*/

    CALL SRCH (BINFO.PSETID(2), DATA,ID2, RTN_CODE);
    IF RTN_CODE=1 THEN RETURN; /*RTNCODE 1 IS TARGET OF ARC NOT FND*/
    END /*WHEN 'A'*/;

  END /*SLECT*/;

/*ENTER ID2 INTO ID1 ID ARRAY*/
CALL RET1 (ID1,P,RTN_CODE);
BEU.ID_ARRAY(BINFO.POS)=ID2;
CALL REP1 (ID1,P,RTN_CODE);
RTN_CODE=0;
END /*CRTB*/;

```

VM/SP CONVERSATIONAL MONITOR SYSTEM

```

FILE: CM/N      PLIOPT  A1
CRTN: PROC(MP,OFF_MODE,RTN_CODE);
/*A S-PROC AT THE N-ARY LEVEL WHICH IS CALLED AS A SUBROUTINE BY THE
* UPDN MODULE WHEN ROOT NODE OP IS CRT*/
XINCLUDE AUPDN;
XINCLUDE ECRIB,ECRTP;

DCL (MP,P)PTR,OFF_MODE(25) BIT (1) CONNECTED, (RTN_CODE,N) FIXED BIN;
DCL DATA BIT (320) VARYING, (SUBSTR,NULL) BUILTIN,
(P,BS,ID) BIT (32), MODE CHAR (1);

/*CREATE ROOT NODE*/
P=MP;
N=UPDN_ARG.DLEN;
DATA=SUBSTR(UPDN_ARG.DATA,1,N*8);
PS=UPDN_ARG.PSETID;
CALL CRTP(P,DATA,ID,RTN_CODE);
IF RTN_CODE^=0 THEN RETURN; /*0+ IS RTN CODE FROM CRTP*****/

/*FOLLOW THE TREE*/
DO P=UPDN_ARG.PTR REPEAT UPDN_ARG.PTR WHILE (P^=NULL);
IF OFF_MODE(UPDN_ARG.MODE)=0'B THEN DO;
  BS=UPDN_ARG.BSETID;
  DATA=SUBSTR(UPDN_ARG.DATA,1,UPDN_ARG.DLEN*8);
  SELECT (UPDN_ARG.OP);

    WHEN (UPDNOP.IST) MODE='A';
    WHEN (UPDNOP.CI) MODE='N';
    OTHERWISE DO;
      RTN_CODE=10; /*10 IS ILLEGAL OP*****/
      RETURN; END;
    END /*SELECT*/;
    CALL CRTB(P,BS,ID,DATA,MODE,RTN_CODE);
    END /*DO IF*/;
    END /*DO WHILE*/;
  IF RTN_CODE^=0 THEN
    RTN_CODE=RTN_CODE+20; /*20+ IS FROM SUBROUTINE CALL*****/
  END /*CRTN*/;

```

FILE: CRTIP PLI0PT A1 VM/SP CONVERSATIONAL MONITOR SYSTEM

```

CRTIP: PROC(PSETID,DATA,ID,RTN_CODE);
/* A S-PROC AT THE N-ARY LEVEL TO CREATE A PRIMITIVE ELEMENT WITHIN
* A PSET DESIGNATED BY PSETID*** */
XINCLUDE DPCAT,DBEU;
XINCLUDE EPC,ESUB1;
DCL (PSETID, ID, ID1) BIT (32), (P,P1) PTR, RTN_CODE FIXED BIN;
DCL (I,J) FIXED BIN, (L_POS,L_POS2) FIXED BIN;
DCL (NULL,UNSPEC) BUILTIN;
DCL DATA BIT (*) ;

CALL GPCT (PSETID,PINFO,RTN_CODE); /*OBTAIN PINFO OF PSETID*/
ID1=PINFO.L_ID; /*ID1 POINTS TO THE FIRST ELEMENT */
L_POS=PINFO.L_POS;
L_POS2=PINFO.L_POS2;
I=PINFO.NUMPTR;
J=PINFO.PLEN*8; /*NO VARIABLE PLEN AVAILABLE IN THIS VERSION*/
ALLOCATE BEU;

BEU.ID ARRAY(L_POS)=ID1;
BEU.ID_ARRAY(L_POS2)=UNSPEC(NULL); /*BI-DIRECTIONAL CHAINING*/
BEU.DATA=DATA;

CALL CRT1 (ID,P,RTN_CODE);
IF RTN_CODE=0 THEN DO; /*1+ IS PROBLEM IN MM LEVEL*/
RTN_CODE=1+RTN_CODE;
RETURN;
END;

/*FIX UP 'X' TYPE PTYPE*/
IF PINFO.PTYPE='X' THEN DO;
BEU.DATA=ID;
CALL REP1(ID,P,RTN_CODE);
END;

FREE BEU;

/*UPDATE PCAT ENTRY*/
PINFO.L_ID=ID;
CALL UPCT (PSETID,PINFO,RTN_CODE);

IF ID1^=UNSPEC (NULL) THEN DO; /*IF ORIGINAL SET NOT EMPTY*/
CALL RET1(ID1,P1,RTN_CODE);
IF P1=NULL THEN DO;
RTN_CODE=20+RTN_CODE; /*20+ PROBLEM IN REPLACING CHAINING*/
RETURN;
END;

/*UPDATE BACK POINTER*/
P1->BEU.ID_ARRAY(L_POS2)=ID;
CALL REP1(ID1,P1,RTN_CODE);
IF RTN_CODE=0 THEN RTN_CODE=10+RTN_CODE;
FREE P1->BEU;
END /*IF ID1 NOT NULL*/;

```

PAGE 002

VM/SP CONVERSATIONAL MONITOR SYSTEM

FILE: CKIP PLIOPT A1

CRT00560
CRT00570
CRT00580

RETURN:
END /*CRTP*/;

FILE: CRT1 PLIOPT A1 VM/SP CONVERSATIONAL MONITOR SYSTEM

```

CRT1:PROC (ID,P,RTN_CODE);
    /*A S-PROC AT THE N-ARY LEVEL FOR COMMUNICATING DIRECTLY WITH THE
    ** CRT MODULE AT THE MM LEVEL*/
    XINCLUDE ACRT, DBEU, SERVICE;
    DCL ID BIT (32), (P,P1,BP) PTR,(RTN_CODE,I,J,K) FIXED BIN;
    DCL (NULL,UNSPEC.STRING,SUBSTR) BUILTIN;
    DCL BIT_STRING BIT(800) BASED;

    /*FILL UP CRT_ARG*/
    ALLOCATE CRT_ARG SET (P1);
    P1->CRT_ARG.N=32+NUMPTR*32*DLEN;
    P1->CRT_ARG.DATA=P->BIT_STRING;
    /*CALL UP CRT*/

    CALL TCALL ('CRT',1,P1,BP);
    RTN_CODE=BP->CRT_RTN_CODE;
    IF RTN_CODE=0 THEN ID=BP->CRT_RTN.ID;
    CALL SVC3(P1); CALL SVC3(BP);

    /******MEMORY REQUEST TRACE DUMP *****//
    IF FFF THEN CALL DUMPT('CREATED',ID,P):DCL FFF BIT(1) EXT; /**/
    /******//

    RETURN;
    END;

```

```

CRT00010
CRT00020
CRT00030
CRT00040
CRT00050
CRT00060
CRT00070
CRT00080
CRT00090
CRT00100
CRT00110
CRT00120
CRT00130
CRT00140
CRT00150
CRT00160
CRT00170
CRT00180
CRT00190
CRT00200
CRT00210
CRT00220
CRT00230
CRT00240
CRT00250
CRT00260
CRT00270
CRT00280

```

FILE: DBA PLOIPT A1 VM/SP CONVERSATIONAL MONITOR SYSTEM

```

DBA:PROC:
/*****
*
*      MODULE DESCRIPTION
*
*      PURPOSE:  FOR DBA TO SELECT AMONG THE FOLLOWING DBA
*      SUBSYSTEMS:  DDB (FOR DEFINING BASE DATA), DDV (FOR
*      DEFINING VIEW DATA) AND DMB (FOR MANIPULATING
*      BASE DATA).
*      PRESENTLY THE DDV SUBSYSTEM IS NOT AVAILABLE.
*
*      CALLS PROCEDURES:
*      INTER-LEVEL T-PROC:  NONE;
*      INTRA-LEVEL T-PROC:  NONE;
*      INTRA-LEVEL S-PROC:  DDB,DDV,DMB; */
XINCLUDE EDDB,EDMB;
/*****
*      CONTROL STRUCTURE SERVICES:  */
/*****
*****/

DCL NAME CHAR(25),
TRUE BIT(1) INIT('1'B);

/* BEGIN SESSION */
PUT SKIP(2) LIST ('--- DATABASE ADMINISTRATOR (DBA) SESSION ---');

/* GET COMMAND */
DO WHILE(TRUE);
  PUT SKIP(2) LIST
  ('DBA: DATA DEFINITION (DD) OR DATA MANIPULATION (DM) OR DATA DEFINITION');
  GET EDIT (NAME) (A(25));
  IF NAME = (25) , THEN LEAVE; /*OUT OF DBA SUBSYSTEM*/

/* SELECT COMMAND */
SELECT(NAME);

/* DATA DEFINITION */
WHEN('DD','DATA DEFINITION')DO WHILE ('1'B);

/*SELECT AMONG BASE DATA DEF MODULE AND VIEW DATA DEF*/
PUT SKIP LIST ('DD: BASE DATA (BASE) OR VIEW DATA (VIEW)');
GET EDIT (NAME) (A(25));
IF NAME = (25) , THEN LEAVE; /*OUT OF DD MODE*/

SELECT (NAME);
  WHEN ('BASE DATA','BASE') CALL DDB;
  WHEN ('VIEW DATA','VIEW') CALL DDV;
  OTHERWISE PUT SKIP LIST (NAME,' IS NOT A VALID CMD. ');
END;

END /*DD WITHIN DD*/;

```

FILE: DBA PLIOPT A1 VM/SP CONVERSATIONAL MONITOR SYSTEM

```

/* QUERIES */
WHEN('DATA MANIPULATION','DM') CALL DMB;
WHEN('DATA DEFINITION QUERY','DDQ') CALL DDQ;

/* INVALID OPERATORS */
OTHERWISE
PUT SKIP(0) EDIT (NAME,' IS AN INVALID COMMAND') (A,A);
END;
END;

/* END SESSION */
PUT SKIP(2) LIST ('-- END OF DBA SESSION --');

/*****
DDV: PROC;
PUT SKIP LIST ('DATABASE VIEW DEFINITION NOT AVAILABLE. ');
RETURN;
END;
/*****/
END /*DBA*/;

```

DBA00560
DBA00570
DBA00580
DBA00590
DBA00600
DBA00610
DBA00620
DBA00630
DBA00640
DBA00650
DBA00660
DBA00670
DBA00680
DBA00690
DBA00700
DBA00710
DBA00720
DBA00730
DBA00740
DBA00750
DBA00760
DBA00770

FILE: DDB PLOPT A1 VN/SP CONVERSATIONAL MONITOR SYSTEM

```

STRING ENTITY 'SETS.'):
DO WHILE ('1'B): /*LOOP FOR ENTITY SET DEFINITIONS*/
PUT SKIP LIST ('NEW ENTITY SET (NEW) OR EXISTING ENTITY SET (OLD)?'):
GET EDIT (CMD) (A(B)):
IF CMD='(B)' THEN LEAVE: /* GET OUT OF DATA DEFINITION */
SELECT (CMD):
WHEN ('NEW') DO:
    ALLOCATE DEFE ARG:
    TP,RP=P: /*SET TEMP & ROOT PTRS*/
    PNAME='PANND': CURSOR=1: LD = ADDR (DEFE ARG.NAME):
    PDISPLAY PNAME(PNAME) RSTATUS (RSTATUS) ULDLIST(LDLIST):
    IF DEFE ARG.NAME='(B)' THEN GO TO RECYCLE_DEFE: /*NO MORE NEW ESETS */
    ENAME1=DEFE_ARG.NAME:/*SAVE ENTITY NAME*/
    /*SET UP DEFE CALL*/
    CALL TCALL ('DEFE',1,RP,TP):
    CALL SVC3 (RP): /*FREE UP CALLING ARGUMENTS*/
    /*CHECK RETURN CODE*/
    IF TP->DEFE_RTN_CODE ^=0 THEN DO:
        CALL DEFMSG (TP->DEFE_RTN_CODE):
        CALL SVC3(TP): /*FREE UP RTN MSG*/
        GO TO RECYCLE_DEFE:
        END:/* ILLEGAL ENTITY SET DEF */
    CALL SVC3 (TP): /*FREE UP RTN MSG*/
    CALL CONFIRM (1,ENAME1):
    GO TO ATTR: /* START DEFINE ATTRIBUTE LOOP */
    END /* WHEN NEW */:
WHEN ('OLD') DO:/* OLD ENTITY SET */
    PNAME='PANND': CURSOR=1: LD=ADDR(ENAME1):
    PDISPLAY PNAME (PNAME) RSTATUS (RSTATUS) ULDLIST (LDLIST):
    IF ENAME1='(B)' THEN GO TO RECYCLE_DEFE: /* GET OUT */
    GO TO ATTR:
    END /* WHEN OLD */:
OTHERWISE DO:
    PUT SKIP LIST ('ILLEGAL. PLEASE RE-ENTER'):
    GO TO RECYCLE_DEFE:
    END:
END /* SELECT */:
ATTR: DO WHILE ('1'B): /*ATTRIBUTE DEF LOOP */
    STRING(TEMP)=REPEAT(' ',100): /*INITIALIZE UNLOAD VAR TO BLANKS*/
    ALLOCATE DEFA_ARG:
    RP=P:

```

```

UNLOAD (1)=ADDR(RP->DEFA_ARG.NAME);
UNLOAD (2) =ADDR (DEFA_ARG.NAME);
UNLOAD(3)= ADDR (TEMP_FUNC);
UNLOAD (4) = ADDR (DEFA_ARG.TYPE);
UNLOAD(5) = ADDR (DEFA_ARG.ENAME);
UNLOAD (6) = ADDR (TEMP_MLEN);
UNLOAD(7)=ADDR(DEFA_ARG.VTYPE);
UNLOAD(8) = ADDR (TEMP_MAX);
UNLOAD(9) = ADDR (TEMP_MIN);
PNAME='PANASD'; CURSOR=2;
PDISPLAY PNAME(PNAME) RSTATUS (RSTATUS) ULDLIST (UNLDLIST)
CURSOR (CURSOR);/* GET DATA FROM PANEL */

IF DEFA_ARG.NAME = (8) ' ' THEN LEAVE; /*NO MORE ATTRIBUTE*/
DEFA_ARG.ENAME1=ENAME1; /*GIVE THE PARENT ENTITY NAME*/

/*PERFORM CONVERSION*/
SELECT (DEFA_ARG.TYPE);
WHEN ('V') DO;
IF TEMP_MLEN=(2) ' ' THEN TEMP_MLEN='20';
IF VERIFY(TEMP_MLEN,'0123456789 ') ^= 0
THEN DO;
CALL DEFMSG (-1);
GO TO RECYCLE_DEFA;
END /* ILLEGAL MLEN */;
DEFA_ARG.MLEN=TEMP_MLEN; /* CONVERSION */

IF DEFA_ARG.VTYPE=' ' THEN DEFA_ARG.VTYPE='C';
IF VERIFY(DEFA_ARG.VTYPE,'NC') ^= 0
THEN DO;
CALL DEFMSG (-2); /* ILLEGAL VTYPE */
GO TO RECYCLE_DEFA;
END; /* ILLEGAL VTYPE */

IF DEFA_ARG.VTYPE='N' THEN DO; /* CHECK MAX, MIN */
IF VERIFY (TEMP_MAX,'0123456789 ') ^= 0 ;
VERIFY (TEMP_MIN,'0123456789 ') ^= 0
THEN DO;
CALL DEFMSG (-3); /* ILLEGAL NUMERIC RANGE */
GO TO RECYCLE_DEFA;
END;
IF TEMP_MAX=(6) ' ' THEN TEMP_MAX='99999';
IF TEMP_MIN=(6) ' ' THEN TEMP_MIN='-99999';
DEFA_ARG.MAX=TEMP_MAX; /* CONVERSION */
DEFA_ARG.MIN=TEMP_MIN;
END;
END /* 'V' */;

WHEN ('E');
OTHERWISE DO;
CALL DEFMSG (-4); /* ILLEGAL ATTRIBUTE TYPE */
GO TO RECYCLE_DEFA;
END /* OTHERWISE */;

```

DDBO1110
 DDBO1120
 DDBO1130
 DDBO1140
 DDBO1150
 DDBO1160
 DDBO1170
 DDBO1180
 DDBO1190
 DDBO1200
 DDBO1210
 DDBO1220
 DDBO1230
 DDBO1240
 DDBO1250
 DDBO1260
 DDBO1270
 DDBO1280
 DDBO1290
 DDBO1300
 DDBO1310
 DDBO1320
 DDBO1330
 DDBO1340
 DDBO1350
 DDBO1360
 DDBO1370
 DDBO1380
 DDBO1390
 DDBO1400
 DDBO1410
 DDBO1420
 DDBO1430
 DDBO1440
 DDBO1450
 DDBO1460
 DDBO1470
 DDBO1480
 DDBO1490
 DDBO1500
 DDBO1510
 DDBO1520
 DDBO1530
 DDBO1540
 DDBO1550
 DDBO1560
 DDBO1570
 DDBO1580
 DDBO1590
 DDBO1600
 DDBO1610
 DDBO1620
 DDBO1630
 DDBO1640
 DDBO1650

FILE: DDB PLOIPT A1 VM/SP CONVERSATIONAL MONITOR SYSTEM

```

END /* SELECT */;

SELECT (TEMP_FUNC); /* CHECK FUNCTION TYPE */
WHEN ('1:1') DEFA_ARG_FUNC='S';
WHEN ('M:1') DEFA_ARG_FUNC='M';
WHEN ('KEY') DEFA_ARG_FUNC='K';
WHEN (' ') DEFA_ARG_FUNC='M'; /* DEFAULT TO M */
OTHERWISE DO;
  CALL DEFMSG (-5); /* ILLEGAL FUNCTION TYPE */
  GO TO RECYCLE_DEFA;
END;

END; /* SELECT */

/*PASS MSG TO DEFA*/
CALL TCALL ('DEFA',1,RP,TP);
CALL SVC3 (RP); /*FREE CALL ARG LIST*/

RTN_CODE=TP->DEFA_RTN_CODE;
CALL SVC3 (TP);

IF RTN_CODE ^= 0 THEN DO;
  CALL DEFMSG (RTN_CODE);
  GO TO RECYCLE_DEFA;
END; /*RTN_CODE IF*/

ELSE CALL CONFIRM (2,DEFA_ARG_NAME);

RECYCLE_DEFA;
END /*ATTR DEF LOOP DO WHILE '1'B*/;

RECYCLE_DEFE;
END /*ENTITY SET DEF LOOP*/;

/***** SUBROUTINE FOR CONFIRM MESSAGE AND ERROR MESSAGE *****/
CONFIRM: PROC (CODE,NAME);
  DCL CODE FIXED BIN (31);
  DCL NAME CHAR(8);
  SELECT (CODE);
    WHEN (1) PUT SKIP EDIT ('ENTITY SET 'NAME','DEFINED') (A,A,A);
    WHEN (2) PUT SKIP EDIT ('ATTRIBUTE 'NAME','DEFINED') (A,A,A);
    OTHERWISE;
  END /* SELECT */;
  CALL PROMPT;
  END /*CONFIRM */;

/***** SUBROUTINE FOR CONFIRM MESSAGE AND ERROR MESSAGE *****/
DEFMSG: PROC (CODE);
  DCL CODE FIXED BIN (31);
  PUT SKIP LIST ('ILLEGAL ENTITY SET NAME: DEFINITION IGNORED. ');
  CALL PROMPT;
  END /* DEFMSG */;

/***** SUBROUTINE FOR CONFIRM MESSAGE AND ERROR MESSAGE *****/

```

FILE: L-J PLIOT A1 VM/SP CONVERSATIONAL MONITOR SYSTEM

```

DEFAMSG: PROC (CODE);
DCL CODE FIXED BIN (31);
SELECT (CODE);
  WHEN (-1) PUT SKIP LIST ('ILLEGAL MAXIMUM LENGTH. ');
  WHEN (-2) PUT SKIP LIST ('ILLEGAL VALUE TYPE. ');
  WHEN (-3) PUT SKIP LIST ('ILLEGAL NUMERIC RANGE. ');
  WHEN (-4) PUT SKIP LIST ('ILLEGAL ATTRIBUTE TYPE. ');
  WHEN (-5) PUT SKIP LIST ('ILLEGAL FUNCTION TYPE. ');
  WHEN (1) PUT SKIP LIST ('ILLEGAL ENTITY SET NAME. ');
  WHEN (7) PUT SKIP LIST ('DUPLICATE ATTRIBUTE NAME. ');
  OTHERWISE:
    END /* SELECT */;
PUT SKIP LIST ('ILLEGAL ATTRIBUTE DEFINITION: DEFINITION IGNORED. ');
CALL PROMPT;
END /*DEFAMSG */;

/*****
PROMPT: PROC;
PUT SKIP LIST ('PRESS ENTER TO CONTINUE');
GET LIST (CMD);
END;
/*****/

END /*DDB*/;

```

DDB02210
 DDB02220
 DDB02230
 DDB02240
 DDB02250
 DDB02260
 DDB02270
 DDB02280
 DDB02290
 DDB02300
 DDB02310
 DDB02320
 DDB02330
 DDB02340
 DDB02350
 DDB02360
 DDB02370
 DDB02380
 DDB02390
 DDB02400
 DDB02410
 DDB02420
 DDB02430
 DDB02440
 DDB02450

```

DDQ: PROC;
/*****
*
*      MODULE      DESCRIPTION
*
* ****
* **** PURPOSE:
* **** S-PROC AT THE USER INTERFACE LEVEL, CALLED AS A SUBROUTINE BY THE
* **** DBA MODULE, TO PROCESS DBA'S REQUEST TO QUERY DATA DEFINITIONS.
* **** THIS MODULE MAKES USE OF THE T-PROC AT THE NEXT LEVEL, SHWE, TO
* **** OBTAIN INFORMATION ABOUT ENTITY SETS AND ATTRIBUTES AND DISPLAYS IT.
* ****
* ****
XINCLUDE ASHWE.SERVICE.ELEX;
DCL NAME CHAR (8);
DCL (NUM_TOKEN,1) FIXED BIN;
DCL RTN_CODE FIXED BIN;
DCL 1_TOKEN(25). /* USED TO COMM. WITH LEX */
      2 D CHAR (40).
      2 L FIXED BIN;
DCL NO_BLANKS FIXED BIN INIT (1);
DCL FUNC1 CHAR (3);
DCL NULL_BUILTIN, P PTR;

/* BEGIN */
PUT SKIP LIST ('--- DATA DEFINITION QUERY SESSION ---');

DO WHILE ('1'B);
  PUT SKIP LIST ('ENTITY SET NAME? USE * IF ALL ENTITY SET NAMES ARE
  ESIED. ');
  CALL LEX (NUM_TOKEN,TOKEN,NO_BLANKS,RTN_CODE);
  IF NUM_TOKEN=0 THEN LEAVE; /* GET OUT */
  ENAME = TOKEN D(1);
  ALLOCATE SHWE_ARG;
  SHWE_ARG.NAME1=ENAME;
  IF ENAME = '*' THEN GO TO SEND; /* WANT ALL */

  LOOP:
    PUT SKIP LIST ('ATTRIBUTE NAMES? SEPERATE BY COMMAS. USE * IF ALL
    TRIBUTES ARE DESIRED');
    CALL LEX (NUM_TOKEN,TOKEN,NO_BLANKS,RTN_CODE); /* CALL LEX TO GET
    LINE */

    IF RTN_CODE ^= 0 THEN DO;
      PUT SKIP LIST ('ILLEGAL SYNTAX');
      GO TO LOOP; /* TRY AGAIN */
    END;

    IF NUM_TOKEN=0 THEN LEAVE; /*GET OUT*/

    IF SUBSTR(TOKEN,D(1),1,1)='*'
    THEN SHWE_ARG.NUM_NAME2=999;
    ELSE DO;
      SHWE_ARG.NUM_NAME2=NUM_TOKEN;
      DO I = 1 TO NUM_TOKEN;
DDQ00010
DDQ00020
DDQ00030
DDQ00040
DDQ00050
DDQ00060
DDQ00070
DDQ00080
DDQ00090
DDQ00100
DDQ00110
DDQ00120
DDQ00130
DDQ00140
DDQ00150
DDQ00160
DDQ00170
DDQ00180
DDQ00190
DDQ00200
DDQ00210
DDQ00220
DDQ00230
DDQ00240
DDQ00250
DDQ00260
DDQ00270
DDQ00280
DDQ00290
DDQ00300
DDQ00310
DDQ00320
DDQ00330
DDQ00340
DDQ00350
DDQ00360
DDQ00370
DDQ00380
DDQ00390
DDQ00400
DDQ00410
DDQ00420
DDQ00430
DDQ00440
DDQ00450
DDQ00460
DDQ00470
DDQ00480
DDQ00490
DDQ00500
DDQ00510
DDQ00520
DDQ00530
DDQ00540
DDQ00550

```

FILE: DDQ PLOIPT A1 VM/SP CONVERSATIONAL MONITOR SYSTEM

```

SHWE_ARG.NAME2(I) = TOKEN.D(I);
END;
END;

SEND;
PUT LIST ('*** DDO TO CALL ICALL SHWE');
CALL TCALL ('SHWE', 1, SHWE_ARG_P, SHWE_RTN_P);
CALL SVC3 (SHWE_ARG_P);

/* EXAMINE RETURN CODE */
I = SHWE_RTN.RTN_CODE;
IF I = 0 THEN DO;
  CALL SHWMSG (I);
  CALL SVC3 (SHWE_RTN_P);
  GO TO END; /* TRY AGAIN */
END;

IF ENAME='*' THEN DO;
  P=SHWE_RTN_P; /* SAVE IT */
  SHWE_RTN_P=SHWE_RTN.PTR; /* ->SHWE_RTN2 */
  PUT SKIP LIST ('NUMBER OF ENTITY SETS DEFINED', NUM_ENAME);
  DO I = 1 TO NUM_ENAME;
    IF MOD(I,8) = 1 THEN PUT SKIP EDIT (SHWE_RTN2.ENAME(I))(A(8));
    ELSE PUT EDIT (SHWE_RTN2.ENAME(I)) (COL((I-1)*9+1),A(8));
  END;
  CALL SVC3 (P);
  END /* IF ENAME = * */;

ELSE DO;
  PUT SKIP LIST ('ENTITY SET NAME', ENAME);
  PUT SKIP EDIT ('ATTRIBUTE NAME FUNCTION TYPE ENAME VTYPE
MAX LEN MAX VALUE MIN VALUE') (A);
  P=SHWE_RTN_P; /* SAVE IT */
  DO SHWE_RTN_P = SHWE_RTN.PTR REPEAT SHWE_RTN1.PTR WHILE
    (SHWE_RTN_P ^= NULL);
    IF TYPE ^= 'E' THEN ENAME = (8) ' ' ; ELSE VTYPE = ' ';
    SELECT (FUNC);
      WHEN ('S') FUNC1='1:1';
      WHEN ('K') FUNC1='KEY';
      WHEN ('M') FUNC1='M:1';
    OTHERWISE;
    END /* SELECT */;
    PUT SKIP EDIT (ANAME,FUNC1,TYPE,SHWE_RTN1.DATA.ENAME,
VTYPE) (X(16),A(8),X(7),A(3),
X(5),A(1),X(2),A(8),X(6),A(1));
    IF TYPE = 'V' THEN DO;
      PUT EDIT (MLEN) (COL(52),F(5));
      IF VTYPE = 'N'
      THEN PUT EDIT (MAX,MIN)(COL(62),F(6),X(5),F(6));
    END;

END /* DO P */;
CALL SVC3 (P);
END /* ELSE */;

```

FILE: DDQ PLOPT AI VM/SP CONVERSATIONAL MONITOR SYSTEM

```

END:
END: /* DO WHILE */

/*****
SHWMSG: PROC(CODE);
DCL CODE FIXED BIN;

/* INTERNAL MSG SUBROUTINE */

IF CODE < 0
THEN DO;
  SELECT (CODE);
    WHEN (-1) PUT SKIP LIST ('ILLEGAL ENTITY NAME');
    WHEN (-2) PUT SKIP LIST ('NO ENTITY SETS DEFINED. ');
    WHEN (-3) PUT SKIP EDIT ('NO ATTRIBUTES DEFINED FOR ENTITY SET',
      ENAME)(A,A);
    OTHERWISE:
      END /*SELECT */;
      END /* CODE < 0 */;

  ELSE IF CODE <= NUM_TOKEN
  THEN PUT SKIP LIST
    ('ILLEGAL ATTRIBUTE NAME', TOKEN.D(1));

  PUT SKIP LIST ('ILLEGAL DD QUERY STATEMENT');
  PUT SKIP LIST ('PLEASE RE-ENTER');
  END /* SHWMSG */;
  /*****/

END /* DDQ */;

```

FILE: Defa PLTOPT A1 VM/SP CONVERSATIONAL MONITOR SYSTEM

```

DEFA:PROC;
/* A T-PROC AT THE ENTITY LEVEL FOR DEFINING ATTRIBUTES*/
%INCLUDE ADEFA, ARETE, AUPDE, SERVICE, FDEBUG, ADEFE;
%INCLUDE DECAT; /*CATALOGUE TEMPLATES*/
%INCLUDE EGACT, EGECT, EDFP1, EDFB1;

DCL (P, MP, TP, RP, RTP, LP) PTR, NULL BUILTIN;
DCL (ENAME, ANAME) CHAR (8), RTN_CODE FIXED BIN;
DCL (PSETID1, PSETID2, PSETID) BIT (32);
DCL BIT_STRING BIT (232) BASED; /*AINFO IS 19 BYTES LONG
      ENCODED STRING*/
DCL (STRING, UNSPEC) BUILTIN;

/*BEGIN*/
CALL TBEG (PROC_ADDR, MP);
ALLOCATE DEFA_RTN SET (RTP);

AINFO=MP->DEFA_ARG, BY NAME;
ENAME=MP->DEFA_ARG.ENAME1; /*PARENT ENTITY NAME*/
ANAME=MP->DEFA_ARG.NAME; /*ATTRIBUTE NAME*/

/*CHECK PARENT ENTITY SET*/
CALL GECT (ENAME, EINFO, RTN_CODE);
IF RTN_CODE=0 THEN DO;
  RTP->DEFA_RTN_CODE=5; /*5 IS PROBLEM WITH PARENT ENTITY*/
  CALL TRTN (PROC_ADDR, RTP);
  RETURN; /*RETURNS*/
END;
PSETID1=EINFO;

/*CHECK IF ATTRIBUTE NAME DUPLICATED*/
CALL GACT (ANAME, ENAME, AINFO, RTN_CODE);
IF RTN_CODE=2 THEN DO; /*2 FROM GACT IS NO SUCH ATTRIBUTE*/
  RTP->DEFA_RTN_CODE=1; /*1 IS DUPLICATE ATTRIBUTE NAME*/
  CALL TRTN (PROC_ADDR, RTP); /*RETURNS*/
  RETURN;
END;

/*HANDLING TARGET PSET*/
IF AINFO.VTYPE='V' THEN DO; /*MUST CREATE TARGET PSET*/
  IF AINFO.VTYPE='N' THEN DO; /*NUMERIC*/
    /*IF FDEBUG.LEV1 THEN PUT SKIP LIST ('DEFA: CREATING V PSET'); */
    CALL DFPI ('N', 4, 'V', RTN_CODE, PSETID2);
  ELSE DO;
    ELSE CALL DFPI (AINFO.VTYPE, AINFO.MLEN, 'V', RTN_CODE, PSETID2);
  END;
END;

ELSE DO; /*ENTITY TYPE, CHECK TO SEE IF EXISTS ALREADY*/
  A: CALL GECT (AINFO.ENAME, EINFO, RTN_CODE);
  SELECT (RTN_CODE);
    WHEN (2) DO; /*MEANS TARGET ENTITY SET DOES NOT EXIST YET*/
      /* IN THIS VERSION THIS MEANS AN ERROR; FUTURE VERSIONS

```

FILE: DEFA PLIOPT A1 VM/SP CONVERSATIONAL MONITOR SYSTEM

```

        WILL OPTIONALLY DEFINES THE TARGET ENTITY SET */
        RTP->DEFA.RTN.CODE=7; /*** 7 IS UNKNOWN TARGET ESET *****/
        CALL TRTN(PROC_ADDR, RTP);
        RETURN;
    END /* WHEN 2*/;

    WHEN (0) /*GOT IT*/
        PSETID2=EINFO;

    OTHERWISE DO; /*ILLEGA TARGET ESET*/
        RTP->DEFA.RTN.CODE=3; /*3 IS PROBLEM IN TARGET ESET*/
        CALL TRTN(PROC_ADDR, RTP);
        RETURN;
    END;

    END /*SELECT*/;
    END /*ELSE DO ENTITY TYPE*/;

/*CREATE BSET*/
/*IF FDEBUG.LEV1 THEN PUT SKIP LIST ('DEFA: CREATING BSET'); */
CALL DFB1 (AINFO.FUNC, PSETID1,PSETID2, RTN_CODE,BSETID);

AINFO.BSETID=BSETID;
AINFO.PSETID=PSETID2;

/*GIVE IT TO UPDE**/
ALLOCATE UPDE_ARG; RP,LP=P; /*ASET NODE*/
UPDE_ARG.NAME='E*ASET';
UPDE_ARG.OP=UPDEOP.CRT;

ALLOCATE UPDE_ARG; /*ESET NODE*/
LP->UPDE_ARG.PTR=P; LP=P;
UPDE_ARG.NAME='A*ESET';
UPDE_ARG.OP=UPDEOP.ID;
UPDE_ARG.PARENT=1;
UPDE_ARG.NODE=2;

ALLOCATE UPDE_ARG; /*ENAME NODE*/
LP->UPDE_ARG.PTR=P; LP=P;
UPDE_ARG.NAME='A*ENAME';
UPDE_ARG.OP=UPDEOP.ID;
UPDE_ARG.PARENT=2;
UPDE_ARG.DLEN=8;
UPDE_ARG.COATA=ENAME;
UPDE_ARG.NODE = 3;

ALLOCATE UPDE_ARG; /*ANAME NODE*/
LP->UPDE_ARG.PTR=P; LP=P;
UPDE_ARG.NAME='A*ANAME';
UPDE_ARG.OP=UPDEOP.ID;
UPDE_ARG.PARENT=1;
UPDE_ARG.DLEN=8;
UPDE_ARG.COATA=ANAME;
UPDE_ARG.NODE = 4;

```

FILE: ULR PLIOPT A1 VM/SP CONVERSATIONAL MONITOR SYSTEM

```

ALLOCATE UPDE_ARG; /*AINFO NODE*/
LP->UPDE_ARG.PTR=P; LP=P;
UPDE_ARG_NAME='A*AINFO';
UPDE_ARG.OP=UPDEOP.1ST;
UPDE_ARG.NODE = 5;
UPDE_ARG.PARENT=1;
UPDE_ARG.DLEN=29;
UNISPEC(UPDE_ARG.CDATA)=ADDR(AINFO)->BIT_STRING;

/*CALL UP UPDE*/
CALL TCALL ('UPDE',1,RP.TP);

/*CHECK RETURN*/
IF TP->UPDE_RTN.RTN_CODE=0 THEN
  RTP->DEFA_RTN.RTN_CODE=6; /*6 IS PROBLEM INUPDATING ENTITY CAT.*/
/*RETURNS*/
CALL SVC3 (RP);
CALL SVC3 (TP);
CALL TRTN( PROC_ADDR, RTP);
RETURN;
END /*DEFA*/ ;

```

```

DEFO1110
DEFO1120
DEFO1130
DEFO1140
DEFO1150
DEFO1160
DEFO1170
DEFO1180
DEFO1190
DEFO1200
DEFO1210
DEFO1220
DEFO1230
DEFO1240
DEFO1250
DEFO1260
DEFO1270
DEFO1280
DEFO1290
DEFO1300
DEFO1310
DEFO1320

```

FILE: DEFB PLIOPT A1 VM/SP CONVERSATIONAL MONITOR SYSTEM

```

DEFB:PROC;
DEF00010
DEF00020
DEF00030
DEF00040
DEF00050
DEF00060
DEF00070
DEF00080
DEF00090
DEF00100
DEF00110
DEF00120
DEF00130
DEF00140
DEF00150
DEF00160
DEF00170
DEF00180
DEF00190
DEF00200
DEF00210
DEF00220
DEF00230
DEF00240
DEF00250
DEF00260
DEF00270
DEF00280
DEF00290
DEF00300
DEF00310
DEF00320
DEF00330
DEF00340
DEF00350
DEF00360
DEF00370
DEF00380
DEF00390
DEF00400
DEF00410
DEF00420
DEF00430

/* A T-PROC AT THE N-ARY LEVEL FOR DEFINING BINARY ASSOCIATION SET */
%INCLUDE SERVICE, ADEFB, DPCAT, DBEU, DNKEY, DBCAT;
%INCLUDE EPCT, EMPST, ECRT;

DCL MAP BIT (32), (P, MP, RTP, TP) PTR, (NULL, UNSPEC, STRING, ADDR) BUILTIN;
DCL (POS, RTN_CODE) FIXED BIN;
DCL 1 BIT_STRING BIT (88) BASED; /* BIT TEMPLATE OF BINFO */
DCL ID BIT (32);

/* BEGIN PROCESSING */
CALL TBEG (PROC_ADDR, MP);
ALLOCATE DEFB_RTN SET (RTP); RTP->DEFB_RTN.RTN_CODE=0;
BINFO=MP->DEFB_ARG, BY NAME;

/* ASSIGN ID ARRAY IN PSETID(1) */
CALL GPCT(BINFO.PSETID(1), PINFO, RTN_CODE); /* OBTAIN CAT ENTRY */
MAP=PINFO.MAP;
CALL MPST (MAP, POS);
IF POS=0 THEN DO;
  RTP->DEFB_RTN.RTN_CODE=1; /* 1 IS ID ARRAY OVERFLOW */
  CALL TRTN(PROC_ADDR, RTP);
  RETURN;
END;

BINFO.POS=POS;

/* UPDATE PSETCAT ENTRY TO REFLECT USAGE OF PTR IN ID_ARRAY */
PINFO.MAP=MAP;
CALL UPCT(BINFO.PSETID(1), PINFO, RTN_CODE);

CALL CRTP(BCATID, ADDR(BINFO)->BIT_STRING, ID, RTN_CODE);
IF RTN_CODE=0 THEN
  RTP->DEFB_RTN.RTN_CODE=1+RTN_CODE; /* 1+ IS FROM CRTP */
ELSE RTP->DEFB_RTN.BSETID=ID;

CALL TRTN (PROC_ADDR, RTP);
RETURN;
END;

```

156

VM/SP CONVERSATIONAL MONITOR SYSTEM

FILE: DEFE PLIOPT A1

```

GO TO RTN;
END;

/*CALL SUBROUTINE TO CALL DEFP*/
CALL DFP1 ('X', O, 'E', RTN_CODE, PSETID);
IF RTN_CODE ^=0 THEN DO;
  IF FDEBUG.LEV1 THEN PUT SKIP LIST ('JUST RETURNED FROM
    DFP1 CALL, RTN CODE SET TO', RTN_CODE);
  RTP->DEFE_RTN.RTN_CODE=2; /*ILLEGAL DEFE*/
  GO TO RTN;
END;

/*CALL UPDE TO CREATE ECAT ENTRY FOR THIS ENTITY SET */
/* BY CREATING AN ENTITY IN THE E*SET ENTITY SET */

ALLOCATE UPDE_ARG; RP=P; /*ROOT PTR*/
UPDE_ARG.NAME='E*ESET';
UPDE_ARG.OP=UPDEOP.CRT;
ALLOCATE UPDE_ARG; /*2ND TOKEN*/
RP->UPDE_ARG.PTR=P; TP=P;
UPDE_ARG.NAME='A*EINFO';
UPDE_ARG.PARENT=1;
UPDE_ARG.NODE=2;
UPDE_ARG.DLEN=4;
UNSPEC(UPDE_ARG.CDATA)=PSETID; UPDE_ARG.OP=UPDEOP.1ST;
ALLOCATE UPDE_ARG; /*3RD TOKEN*/
TP->UPDE_ARG.PTR=P;
UPDE_ARG.NAME='A*ENAME';
UPDE_ARG.PARENT=1;
UPDE_ARG.NODE=3;
UPDE_ARG.DLEN=8;
UPDE_ARG.CDATA=NAME;
UPDE_ARG.OP=UPDEOP.1ST;

CALL TCALL ('UPDE', 1, RP, TP);

CALL SVC3 (RP); /*FREE CALLING ARG LIST*/
/*CHECK RTN_CODE*/
IF TP->UPDE_RTN.RTN_CODE ^=0 THEN DO;
  IF FDEBUG.LEV1 THEN PUT SKIP LIST ('2ND TIME ROUND FROM UPE1,
    RTN_CODE SET TO', TP->UPDE_RTN.RTN_CODE);
  RTP->DEFE_RTN.RTN_CODE =3; /*ILLEGAL DEFE*/
  END;
CALL SVC3 (TP); /*FREE UP RTN MSG*/

/*SET UP DEFE RTN*/
RTN:CALL TRTN(PROC.ADDR, RTP);
/*IF FDEBUG.LEV1 THEN PUT SKIP LIST ('DEFE RETURNS'); */
END /*DEFE*/;

```

```

DEFO0560
DEFO0570
DEFO0580
DEFO0590
DEFO0600
DEFO0610
DEFO0620
DEFO0630
DEFO0640
DEFO0650
DEFO0660
DEFO0670
DEFO0680
DEFO0690
DEFO0700
DEFO0710
DEFO0720
DEFO0730
DEFO0740
DEFO0750
DEFO0760
DEFO0770
DEFO0780
DEFO0790
DEFO0800
DEFO0810
DEFO0820
DEFO0830
DEFO0840
DEFO0850
DEFO0860
DEFO0870
DEFO0880
DEFO0890
DEFO0900
DEFO0910
DEFO0920
DEFO0930
DEFO0940
DEFO0950
DEFO0960
DEFO0970
DEFO0980
DEFO0990
DEFO1000
DEFO1010
DEFO1020
DEFO1030
DEFO1040
DEFO1050

```

FILE: DEFP PLIOPT A1 VM/SP CONVERSATIONAL MONITOR SYSTEM

```

DEFP PROC;
/*T-PROC AT THE N-ARY LEVEL FOR DEFINING PRIMITIVE SETS*/
XINCLUDE SERVICE.ADEFP,DPCAT,DNKEY;
XINCLUDE EMPST.ECRTP;
DCL (P.AP,RTP) PTR. (RTN_CODE.1) FIXED BIN, POS FIXED BIN;
DCL MAP BIT(32) INIT ((32)'0'B).(NULL,UNSPEC.STRING,ADDR)BUILTIN;
DCL ID BIT (32);
DCL BIT_STRING BIT (144) BASED; /*A PSET CAT ENTRY IS 144 BYTES*/
/*BEGIN PROCESSING**/
CALL TBEG (PROC.ADDR,AP);
ALLOCATE DEFP_RTN SET (RTP);
RTP->DEFP_RTN.RTN_CODE=0;

PINFO=AP->DEFP_ARG. BY NAME; /*COPY INFO FROM DEFP_ARG TO PINFO*/
IF PINFO.PTYPE='X' THEN PINFO.PLEN=4; /*FIX UP 'X' TYPE*/
PINFO.NUMPTR=AP->DEFP_ARG.IMP.NUMPTR;
PINFO.LTYPE=PINFOLTYPE.LL; /*ONLY LINK LIST IS AVAILABLE IN THIS VER*/
PINFO.L_ID=UNSPEC(NULL());

DO I=PINFO.NUMPTR+1 TO 32; /*INIT MAP*/
  CALL MPST(MAP.POS);
END;

CALL MPST(MAP.POS); /*FIND FORWARD LINK POS*/
PINFO.L_POS=POS;
CALL MPST(MAP.POS); /*FIND BACKWARD LINK POS*/
PINFO.L_POS2=POS;
PINFO.MAP=MAP;

CALL CRTP(PCATID,ADDR(PINFO)->BIT_STRING.ID,RTN_CODE);
IF RTN_CODE=0 THEN RTP->DEFP_RTN.RTN_CODE=RTN_CODE;
ELSE RTP->DEFP_RTN.PSETID=ID;

CALL TRTN(PROC.ADDR, RTP);
RETURN;
END /*DEFP*/;

```

```

DEFO0010
DEFO0020
DEFO0030
DEFO0040
DEFO0050
DEFO0060
DEFO0070
DEFO0080
DEFO0090
DEFO0100
DEFO0110
DEFO0120
DEFO0130
DEFO0140
DEFO0150
DEFO0160
DEFO0170
DEFO0180
DEFO0190
DEFO0200
DEFO0210
DEFO0220
DEFO0230
DEFO0240
DEFO0250
DEFO0260
DEFO0270
DEFO0280
DEFO0290
DEFO0300
DEFO0310
DEFO0320
DEFO0330
DEFO0340
DEFO0350
DEFO0360
DEFO0370
DEFO0380
DEFO0390
DEFO0400

```

FILE: DEL PL10PT A1 VM/SP CONVERSATIONAL MONITOR SYSTEM

```

DEL:PROC;
DELO0010
DELO0020
DELO0030
DELO0040
DELO0050
DELO0060
DELO0070
DELO0080
DELO0090
DELO0100
DELO0110
DELO0120
DELO0130
DELO0140
DELO0150
DELO0160
DELO0170
DELO0180
DELO0190
DELO0200
DELO0210
DELO0220
DELO0230

/*T-PROC AT THE MM LEVEL FOR DELETING A SEU FROM SH*****//
/* WILL SET INVAL AT ID*/
/* FOR INTEGRATED VERSION: MAKE USE OF SUBROUTINES PSEU, GSEU */

%INCLUDE ADEL.DSEU1.SERVICE.EGSEU.EPSEU;
DCL (P,RTP) PTR, (NULL,UNSPEC) BUILTIN, RTN_CODE FIXED BIN;

CALL TBEG (PROC_ADDR,P);
ALLOCATE DEL_RTN SET (RTP); RTP->DEL_RTN.RTN_CODE=0;
CALL GSEU(DEL_ARG.ID, SEUPTR,RTN_CODE); /* GET SEU TO BE DELETED */
IF RTN_CODE = 1 /* INVALID */
THEN RTP->DEL_RTN.RTN_CODE=1;
ELSE DO;
  SEU.INVAL = 'Y';
  CALL PSEU (DEL_ARG.ID, SEUPTR, RTN_CODE); /* SET SEU INVAL FLAG */
END;

/*RETURNS*/
CALL TRTN(PROC_ADDR, RTP);
RETURN;
END /*DEL*/;

```

FILE: DELB PLIOPT A1 VM/SP CONVERSATIONAL MONITOR SYSTEM

```

DELB: PROC (PARENT, BSETID, ID1, MODE, RTN_CODE);
/*A S-PROC AT THE N-ARY LEVEL TO DELETE A BINARY ASSOCIATION*/
XINCLUDE DBEU, DBCAT;
XINCLUDE EBCT, ESUB1, EDELP;

DCL (PARENT, BSETID, ID1, ID2) BIT (32), P PTR, MODE CHAR (1);
DCL (NULL, UNSPEC) BUILTIN;
DCL RTN_CODE FIXED BIN;

CALL GBCT (BSETID, BINFO, RTN_CODE);
IF RTN_CODE ^= 0 THEN RETURN; /* 0+ IS FROM GBCT */
CALL RET1 (ID1, P, RTN_CODE);
IF P=NULL THEN DO;
    RTN_CODE=10; /*+10 IS BAD SOURCE ELEM ++++++*/
    RETURN; END;

ID2=BEU.ID_ARRAY(BINFO.POS);
IF ID2 ^= UNSPEC (NULL) THEN DO; /* IF THERE EXISTS A BIN ASSOC ELEM */
    BEU.ID_ARRAY(BINFO.POS)=UNSPEC (NULL); /*DELETE ARC*/
    SELECT (MODE);
        WHEN ('N') /*MODE DELETES TARGET NODE TOO*/
            IF BINFO.FUNC='S' THEN CALL DELP(BINFO.PSETID(2), ID2, RTN_CODE);
        OTHERWISE;
            END /*SELECT*/;
        IF RTN_CODE ^= 0 THEN RTN_CODE = 20 + RTN_CODE; /* 20 + IS RTNCODE
            FROM DELP ++++++*/
        END /* IF THERE EXISTS... */;
    RETURN;
    END /*DELB*/;
DELO0010
DELO0020
DELO0030
DELO0040
DELO0050
DELO0060
DELO0070
DELO0080
DELO0090
DELO0100
DELO0110
DELO0120
DELO0130
DELO0140
DELO0150
DELO0160
DELO0170
DELO0180
DELO0190
DELO0200
DELO0210
DELO0220
DELO0230
DELO0240
DELO0250
DELO0260
DELO0270
DELO0280
DELO0290
DELO0300
DELO0310
DELO0320
DELO0330
DELO0340
DELO0350
DELO0360

```

FILE: DELN PLIOPT A1 VM/SP CONVERSATIONAL MONITOR SYSTEM

```

DELN: PROC(MP,OFF_NODE,RTN_CODE);
/*A S-PROC AT THE N-ARY LEVEL USED AS A SUBROUTINE CALLED BY THE
*UPDN PROGRAM TO PERFORM DEE OP AT ROOT NODE*/
XINCLUDE AUPDN;
XINCLUDE EDELB,ESRCH,EDELP;

DCL (MP,P) PTR, (N,RTN_CODE) FIXED BIN,
     OFF_NODE(25) BIT (1) CONNECTED,
     DATA_BIT (320) VARYING, (PS,ID,BS) BIT (32), (SUBSTR,NULL)BUILTIN;

/*DO P=MP REPEAT UPDN_ARG.PTR WHILE(P^=NULL);
  PUT SKIP LIST('DELN ARG LIST',UPDN_ARG); END: */

P=MP;
/*LOCATE ROOT NODE ID*/
PS=UPDN_ARG.PSETID;
DATA=SUBSTR(UPDN_ARG.DATA,1,UPDN_ARG.DLEN * 8);
CALL SRCH(PS,DATA,ID,RTN_CODE);

/*FOLLOW THE TREE*/
DO P=UPDN_ARG.PTR REPEAT UPDN_ARG.PTR WHILE (P^=NULL);
  IF OFF_NODE(UPDN_ARG.NODE)='O'B THEN DO;
    BS=UPDN_ARG.BSETID;
    SELECT (UPDN_ARG.OP);

    WHEN (UPDNOP.DEL) CALL DELB (PS,BS,ID,'A',RTN_CODE);
    WHEN (UPDNOP.DD) CALL DELB (PS,BS,ID,'N',RTN_CODE);

    END /*SELECT*/;
    END /*DO IF*/;
    END /*DO P LOOP */;

  IF RTN_CODE ^= 0 THEN RETURN; /* RTN_CODE SET TO RTN FROM SBR CALLS */

  /* NOW DELETE THE ROOT NODE */
  CALL DELP (PS,ID,RTN_CODE);
  IF RTN_CODE ^= 0 THEN RTN_CODE = 50 + RTN_CODE; /* 50+ FROM DELP */
  END /*DELN*/;

```

```

DELOO010
DELOO020
DELOO030
DELOO040
DELOO050
DELOO060
DELOO070
DELOO080
DELOO090
DELOO100
DELOO110
DELOO120
DELOO130
DELOO140
DELOO150
DELOO160
DELOO170
DELOO180
DELOO190
DELOO200
DELOO210
DELOO220
DELOO230
DELOO240
DELOO250
DELOO260
DELOO270
DELOO280
DELOO290
DELOO300
DELOO310
DELOO320
DELOO330
DELOO340
DELOO350
DELOO360
DELOO370
DELOO380
DELOO390
DELOO400
DELOO410

```

```

DELP: PROC (PSETID, ID, RTN_CODE);
/* A S-PROC AT THE N-ARY LEVEL WHICH DELETES A P ELEMENT GIVEN ITS
* ID WITHIN A PSETID */
XINCLUDE DPCAT, DBEU;
XINCLUDE EPCT, ESUB1;
DCL (PSETID, ID) BIT (32), RTN_CODE FIXED BIN, P_PTR;
DCL (NULL, UNSPEC) BUILTIN, (PREV_ID, POST_ID) BIT (32);
CALL GPCT (PSETID, PINFO, RTN_CODE); /* FIRST OBTAINS PSET CAT */
CALL RET1 (ID, P, RTN_CODE); /* OBTAIN ELEM TO BE DELETED */
IF P=0 THEN DO;
RTN_CODE=1; /* BAD ELEM, MIGHT ALREADY BEEN DELETED */
RETURN; END;
PREV_ID=BEU.ID_ARRAY(PINFO.L_POS2); /* REMEMBER NEIGHBORS IN CHAIN */
POST_ID=BEU.ID_ARRAY(PINFO.L_POS);
CALL DEL1 (ID, RTN_CODE); /* DELETE IT */
FREE BEU;
/* MODIFY PTR CHAIN WITHIN THE PSET */
IF PREV_ID ^= UNSPEC(0) THEN DO;
CALL RET1 (PREV_ID, P, RTN_CODE);
BEU.ID_ARRAY(PINFO.L_POS)=POST_ID;
CALL REP1 (PREV_ID, P, RTN_CODE);
FREE BEU;
END;
ELSE DO; /* IF PREV_ID IS NULL THEN ELEM BEING DELETED IS FIRST IN
PSET CHAIN, THEREFORE NEEDS TO UPDATE PCAT ENTRY */
PINFO.L_ID = POST_ID;
CALL UPCT (PSETID, PINFO, RTN_CODE);
END;
IF POST_ID ^= UNSPEC(0) THEN DO;
CALL RET1 (POST_ID, P, RTN_CODE);
BEU.ID_ARRAY(PINFO.L_POS2)=PREV_ID;
CALL REP1 (POST_ID, P, RTN_CODE);
FREE BEU;
END;
RTN_CODE=0; RETURN;
END /* DELP */;

```

DELO0010
 DELO0020
 DELO0030
 DELO0040
 DELO0050
 DELO0060
 DELO0070
 DELO0080
 DELO0090
 DELO0100
 DELO0110
 DELO0120
 DELO0130
 DELO0140
 DELO0150
 DELO0160
 DELO0170
 DELO0180
 DELO0190
 DELO0200
 DELO0210
 DELO0220
 DELO0230
 DELO0240
 DELO0250
 DELO0260
 DELO0270
 DELO0280
 DELO0290
 DELO0300
 DELO0310
 DELO0320
 DELO0330
 DELO0340
 DELO0350
 DELO0360
 DELO0370
 DELO0380
 DELO0390
 DELO0400
 DELO0410
 DELO0420
 DELO0430
 DELO0440
 DELO0450
 DELO0460

FILE: DEL1 PL1OPT A1 VM/SP CONVERSATIONAL MONITOR SYSTEM

```

DEL1: PROC (ID,RTN_CODE);
/*A S-PROC AT THE LOWEST LAYER OF N-ARY LEVEL TO SET UP CALL TO
* DEL MODULE AT THE MM LEVEL TO DELETE BEU DESIGNATED BY ID*/
%INCLUDE SERVICE.ADEL.FDEBUG;
DCL (P,TP) PTR, NULL BUILTIN, RTN_CODE FIXED BIN, ID BIT (32);

ALLOCATE DEL_ARG;
DEL_ARG.ID=ID;
CALL TCALL ('DEL',1,P,TP);

/***** MEMORY REQUEST TRACE DUMP *****/
IF FFF THEN CALL DUMPIT ('DELETED',ID,NULL);
DCL FFF BIT (1) EXT;
/****/

RTN_CODE=TP->DEL_RTN_CODE;
CALL SVC3(P);
CALL SVC3(TP);
RETURN;
END /*DEL1*/;
DUM00010
DUM00020
DUM00030
DUM00040
DUM00050
DUM00060
DUM00070
DUM00080
DUM00090
DUM00100
DUM00110
DUM00120
DUM00130
DUM00140
DUM00150
DUM00160
DUM00170
DUM00180
DUM00190
DUM00200
DUM00210
DUM00220

```

FILE: L, B1 PLIDPT A1 VM/SP CONVERSATIONAL MONITOR SYSTEM

```

DFB1: PROC(FUNC, PSETID1, PSETID2, RTN_CODE, BSETID);
/*A S-PROC AT THE ENTITY LEVEL FOR SETTING UP CALLS TO DEFB*/
XINCLUDE SERVICE, ADEFB, FDEBUG;

DCL NULL BUILTIN, (P, TP)PTR;
DCL FUNC CHAR (1), (PSETID1, PSETID2)BIT(32);
DCL RTN_CODE FIXED BIN, BSETID BIT(32);

ALLOCATE DEFB_ARG;
DEFB_ARG.PSETID(1)=PSETID1;
DEFB_ARG.PSETID(2)=PSETID2;
DEFB_ARG.FUNC=FUNC;
IF FUNC='K' THEN DEFB_ARG.FUNC='S';

CALL TCALL ('DEFB', 1, P, TP);
RTN_CODE=TP->DEFB_RTN_CODE;
IF RTN_CODE=0 THEN BSETID=TP->DEFB_RTN_BSETID;

CALL SVC3 (P);
CALL SVC3 (TP);
RETURN;
END /*DFB1*/;

```

```

DFB00010
DFB00020
DFB00030
DFB00040
DFB00050
DFB00060
DFB00070
DFB00080
DFB00090
DFB00100
DFB00110
DFB00120
DFB00130
DFB00140
DFB00150
DFB00160
DFB00170
DFB00180
DFB00190
DFB00200
DFB00210
DFB00220
DFB00230

```

```

DFP00010 DFP1:PROC ( PTYPE,PLEN,TYPE,RTN_CODE,PSETID);
/*****
*          MODULE      DESCRIPTION
*          ****
DFP00030 *****
DFP00040 *****
DFP00050 *****
DFP00060 *****
DFP00070 *****
DFP00080 *****
DFP00090 *****
DFP00100 *****
DFP00110 *****
DFP00120 *****
DFP00130 *****
DFP00140 *****
DFP00150 *****
DFP00160 *****
DFP00170 *****
DFP00180 *****
DFP00190 *****
DFP00200 *****
DFP00210 *****
DFP00220 *****
DFP00230 *****
DFP00240 *****
DFP00250 *****
DFP00260 *****
DFP00270 *****
DFP00280 *****
DFP00290 *****
DFP00300 *****
DFP00310 *****
DFP00320 *****
DFP00330 *****
DFP00340 *****
DFP00350 *****
DFP00360 *****
DFP00370 *****
DFP00380 *****
DFP00390 *****
DFP00400 *****
DFP00410 *****
DFP00420 *****
DFP00430 *****
DFP00440 *****
DFP00450 *****
DFP00460 *****
DFP00470 *****
DFP00480 *****
DFP00490 *****
DFP00500 *****
DFP00510 *****
DFP00520 *****
DFP00530 *****
DFP00540 *****
DFP00550 *****

*****/
DCL (P.TP.RP)PTR, NULL BUILTIN;
DCL RTN_CODE FIXED BIN;
DCL PTYPE CHAR(1), PLEN FIXED BIN, TYPE CHAR(1);
DCL PSETID BIT(32);
/*BEGIN PROCESSING*/

    ALLOCATE DEFP_ARG;   RP=P; /*ROOT PTR*/
    DEFP_ARG.PTYPE=PTYPE;
    DEFP_ARG.PLEN=PLEN;
    CALL IMPP; /*IMPP COMPLETES THE IMP PORTION OF DEFP_ARG*/

    /*SET UP MSG TO CALL DEFP*/
    CALL TCALL ('DEFP',1,RP,TP);

    CALL SVC3(RP); /*FREE CALLING ARG LIST*/
    /*EXTRACT RTN_CODE AND PSETID*/
    P=TP;
    RTN_CODE=TP->DEFP_RTN_CODE;
    IF FDEBUG.LEV1 THEN DO;
        IF DEFP_RTN_CODE=-O THEN DO;
            PUT SKIP LIST ('DFP1: DEFP RTN CODE NOT O, IS',DEFP_RTN_CODE);
            RTN_CODE=O;
        END;
    END;
    PSETID=TP->DEFP_RTN_PSETID;
    CALL SVC3 (TP); /*FREE UP RTN MSG LIST*/
    RETURN;
IMPP:PROC; /*INTERNAL SUBROUTINE FOR ASSIGNING IMP PORTION OF DEFP*/

```

FILE: Drr1 PLIOPT A1 VM/SP CONVERSATIONAL MONITOR SYSTEM

```

/*NOTE THAT THIS ROUTINE INTERFACES WITH IMPLEMENTATION
OF THE N-ARY LEVEL, THEREFORE SUBJECT TO CHANGE IF
DESIGN OF N-ARY LEVEL CHANGES*/
IF TYPE='E' THEN DO;
  DEFP_ARG.IMP.HOW=DEFPHOW.SA; /*STAND ALONE*/
  DEFP_ARG.IMP.NUMPTR=12; /*DEFAULT TO 12*/
END;
ELSE DO;
  DEFP_ARG.IMP.HOW=DEFPHOW.SA; /*ALSO STAND ALONE*/
  DEFP_ARG.IMP.NUMPTR=4; /*DEFAULT TO 4*/
END;
END /*IMPP*/;
END /*DFP1*/;

```

```

DFP00560
DFP00570
DFP00580
DFP00590
DFP00600
DFP00610
DFP00620
DFP00630
DFP00640
DFP00650
DFP00660
DFP00670
DFP00680

```

FILE: DMB PLIOPT A1 VM/SP CONVERSATIONAL MONITOR SYSTEM

```

DMB: PROC;
DCL CMD CHAR(7);

PUT SKIP LIST('--- DATA MANIPULATION SESSION ---');
DO WHILE ('1'B); /*DO FOREVER*/
  PUT SKIP LIST('DM: ENTER MANIPULATION COMMAND:');
  PUT SKIP LIST('CREATE(CRT), MODIFY(MOD), DELETE(DEL), QUERY(QUE)');
  GET EDIT (CMD) (A(8));
  SELECT (CMD);
  WHEN ('CREATE','CRT','C') CALL DMC;
  WHEN ('MODIFY','MOD','M') CALL DMH;
  WHEN ('DELETE','DEL','D') CALL DMH;
  WHEN ('QUERY','QUE','Q') CALL DMQ;
  WHEN ((8),) LEAVE;
  OTHERWISE PUT SKIP LIST('ILLEGAL COMMAND,(PLEASE USE UPPER CASE)');
END /*SELECT*/;
END /*DO WHILE*/;

END /*DM*/;

```

DMB00010
DMB00020
DMB00030
DMB00040
DMB00050
DMB00060
DMB00070
DMB00080
DMB00090
DMB00100
DMB00110
DMB00120
DMB00130
DMB00140
DMB00150
DMB00160
DMB00170
DMB00180
DMB00190

FILE: C:\C PLTOPT A1 VM/SP CONVERSATIONAL MONITOR SYSTEM

```

DMC: PROC; /*PROCESSING CREATE COMMANDS*/
%INCLUDE ELEX,ELEXT,EVNM1,EUPE1;
DCL 1 T. /*DATA STRUCTURE OF NAMES */
2 ENAME CHAR(8).
2 TN FIXED BIN. /*NUMBER OF ATTR*/
2 T1 (25).
3 PARENT FIXED BIN.
3 ANAME CHAR (8);
DCL RTN CODE FIXED BIN;
DCL (OP(25), OPE) CHAR(1);
DCL N FIXED BIN; /*THE FOLLOWING USED TO CALL LEX*/
DCL 1 TOKEN(25).
2 D CHAR(40).
2 L FIXED BIN;
2 L1 FIXED BIN;
DCL MODE FIXED BIN;
DCL N1 FIXED BIN; /*USED TO COMM. WITH VNM1*/
DCL 1 PRED(25)./* STRUCTURE TO COMMUNICATE WITH LEXT*/
2 OP CHAR (1). /* =,>,<, */
2 VALUE CHAR (40).
2 VLEN FIXED BIN;

DO WHILE ('1'B); /*DO FOREVER*/
A1:PUT SKIP LIST ('CREATE: ENTER ENTITY SET NAME');
CALL LEX (N,TOKEN,MODE,RTN_CODE);
IF N = 0 THEN LEAVE; /*GET OUT OF DMC*/
T.ENAME = TOKEN.D(1);

A:PUT SKIP LIST ('ENTER NAMES OF ATTRIBUTES SEPARATED BY COMMA');
MODE=1; /*SUPPRESS BLANKS*/
CALL LEX (N, TOKEN, MODE,RTN_CODE);/*GET INPUT LINE THRU LEX*/
IF RTN_CODE=0 THEN DO;
PUT SKIP LIST ('IMPROPER SYNTAX (LEX). PLEASE RE-ENTER');
GO TO A;
END;

IF N=0 THEN LEAVE; /*GET OUT OF DMC*/

CALL LEXT (N,TOKEN,T1,T.TN,PRED,OP,RTN_CODE);/*BUILD T STRUCTURE*/
IF RTN_CODE=0 THEN DO;
PUT SKIP LIST ('IMPROPER SYNTAX. PLEASE RE-ENTER');
GO TO A;
END;

/*CALL UP VNAME TO CHECK FOR LEGALITY OF NAMES*/
CALL VNM1(T,RTN_CODE,N1); /* VNM1 WILL PRINT ERR MSG IF ANY*/
IF N1=-1 THEN GO TO A1; /* ESET NAME ERROR */
ELSE IF N1 ^= 0 THEN GO TO A; /* ATTR NAME ERROR */

/*START ASKING FOR DATA THROUGH SUBROUTINE UPE1*/
OPE='C';
DO I=1 TO 25; OP(I)='N'; /*INITIALIZE OPS TO BE 'IST' */ END;
PUT SKIP LIST ('ENTER DATA:');

CALL UPE1 (OPE,T,OP,RTN_CODE); /*UPE1 WILL PRINT ERR MSG IF ANY */
END /*DO WHILE*/;

```

FILE: DMC

PLIOP T A1

VM/SP CONVERSATIONAL MONITOR SYSTEM

PAGE 002

RETURN;
END /*DMC*/;

DMC00560
DMC00570

FILE: DMD PLODPT A1 VM/SP CONVERSATIONAL MONITOR SYSTEM

```

DMD: PROC; /*PROCESSING DELETE COMMANDS*/
XINCLUDE ELEX,ELEXT,EVNM1,EUPE1;
DCL 1 T, /*DATA STRUCTURE USED TO COMMUNICATE WITH LEXT*/
2 ENAME CHAR(8),
2 TN FIXED BIN, /*NUMBER OF ATTR*/
2 T1 (25),
3 PARENT FIXED BIN,
3 ANAME CHAR (8);
DCL RTN CODE FIXED BIN;
DCL (OP(25), OPE) CHAR(1);
DCL N FIXED BIN; /*THE FOLLOWING USED TO CALL LEX*/
DCL 1 TOKEN(25),
2 D CHAR(40),
2 L FIXED BIN;
DCL MODE FIXED BIN; /*USED TO COMM. WITH VNM1*/
DCL N1 FIXED BIN; /*STRUCTURE TO COMMUNICATE WITH LEXT*/
DCL 1 PRED(25), /*
2 OP CHAR (1), /* =,>,<, */
2 VALUE CHAR (40),
2 VLEN FIXED BIN;
DCL CMD CHAR (8);

PUT SKIP LIST ('TYPE YES IF NEED HELP');
GET EDIT (CMD) (A(8)); IF CMD='YES' THEN CALL HELP;

DO WHILE ('1'B); /*DO FOREVER*/
A1:PUT SKIP LIST ('DELETE: ENTER ENTITY SET NAME');
CALL LEX (N, TOKEN, 1, RTN_CODE);
IF N = 0 THEN LEAVE; /*GET OUT OF DMD*/
T.ENAME = TOKEN.D (1);

A:PUT SKIP LIST ('ENTER IDENTIFIER ATTRIBUTE NAMES. SEPARATE BY COMMA
);
MODE=1; /*SUPPRESS BLANKS*/
CALL LEX (N, TOKEN, MODE,RTN_CODE);/*GET INPUT LINE THRU LEX*/
IF RTN_CODE^=0 THEN DO;
PUT SKIP LIST ('IMPROPER SYNTAX (LEX), PLEASE RE-ENTER');
GO TO A;
END;

IF N=0 THEN LEAVE; /*GET OUT OF DMD*/

CALL LEXT (N,TOKEN,T1,T.TN,PRED,OP,RTN_CODE);/*BUILD T STRUCTURE*/
IF RTN_CODE^=0 THEN DO;
PUT SKIP LIST ('IMPROPER SYNTAX, PLEASE RE-ENTER');
GO TO A;
END;

/*CALL UP VNME TO CHECK FOR LEGALITY OF NAMES*/
CALL VNM1(T,RTN_CODE,N1);
IF N1=-1 THEN GO TO A1; /* ESET NAME ERROR */
ELSE IF N1 ^= 0 THEN GO TO A; /* ATTR NAME ERROR */

/*START ASKING FOR DATA THROUGH SUBROUTINE UPE1*/
OPE='D';

```

FILE: DMD PLIOPT A1 VM/SP CONVERSATIONAL MONITOR SYSTEM

```

PUT SKIP LIST ('ENTER DATA:');
DO I = 1 TO 25; OP(I) = '1'; END; /*INITIALIZE ALL OPS TO ID */
CALL UPE1 (OPE,T,OP,RTN_CODE); /*UPE1 WILL PRINT ERR MSG IF ANY */
END /*DO WHILE*/;
RETURN;

/***** INTERNAL SUBROUTINE *****/
HELP: PROC; /*... BRIEF EXPLANATION ON HOW TO DO DELETE ***);DMD00670
PUT SKIP LIST ('AT THE PROMPT FOR IDENTIFIER ATTRIBUTE NAMES PLEASE'); DMD00680
PUT SKIP LIST ('ENTER ATTRIBUTE NAMES WHOSE VALUE WILL BE GIVEN TO'); DMD00690
PUT SKIP LIST ('IDENTIFY THE ENTITY TO BE DELETE. '); DMD00700
PUT SKIP LIST ('AN EXAMPLE FOLLOWS: '); DMD00710
PUT SKIP LIST('TO DELETE THE EMPLOYEE WITH A CERTAIN EMPLOYEE NUMBER');DMD00720
PUT SKIP LIST ('FIRST ISSUE THE FOLLOWING AT THE PROMPT FOR'); DMD00730
PUT SKIP LIST ('IDENTIFIER ATTRIBUTES: '); DMD00740
PUT SKIP LIST ('EMPNUM'); DMD00750
PUT SKIP LIST('AND LATER ENTER EMPNUM AT THE PROMT FOR DATA'); DMD00760
PUT SKIP LIST ('***** NOW TRY IT *****'); DMD00770
END /*HELP*/; DMD00780
/***** DMD00790
DMD00800
DMD00810

```

```

DMM: PROC; /*PROCESSING CREATE COMMANDS*/
/*-----*
*          MODULE      DESCRIPTION
*-----*
/*-----*/
PURPOSE:
THIS IS A S-PROC AT THE USER INTERFACE LEVEL, CALLED BY
MODULE DMB, TO PROCESS USER DATA MODIFICATION COMMAND
WITHIN A USER DATA MANIPULATION - MODIFICATION SESSION.
/*-----*/
METHOD:
1. INPUT ENTITY SET NAME IN T.ENAME.
2. CALL LEX TO INPUT AND PARSE ATTRIBUTE LIST INTO TOKENS.
3. CALL TEXT TO BUILD NAME TREE AND OP TABLE.
4. CALL VNM1 TO SET UP CALL TO VNME AT THE NEXT LEVEL
   TO VERIFY NAMES.
5. CALL UPE1 TO GET DATA FROM USER AND SET UP CALL TO UPDE.
/*-----*/
CALLS PROCEDURES:
INTER-LEVEL T-PROC: NONE
INTRA-LEVEL T-PROC: NONE
INTRA-LEVEL S-PROC: LEX.LEXT,VNM1,UPE1.
CONTROL STRUCTURE, PANEL MANAGER AND DEBUGGING FACILITIES:
NONE.
/*-----*/
%INCLUDE ELEX,ELEXT,EVNM1,EUPE1;
DCL 1 T, /*DATA STRUCTURE USED TO COMMUNICATE WITH LEXT*/
2 ENAME CHAR(8), /*NUMBER OF ATTR*/
2 TN FIXED BIN,
2 T1 (25),
3 PARENT FIXED BIN,
3 ANAME CHAR (8);
DCL RTN CODE FIXED BIN;
DCL (OP(25), OPE) CHAR(1);
DCL N FIXED BIN; /*THE FOLLOWING USED TO CALL LEX*/
DCL 1 TOKEN(25),
2 D CHAR(40),
2 L FIXED BIN;
DCL MODE FIXED BIN;
DCL N1 FIXED BIN; /*USED TO COMM. WITH VNM1*/
DCL 1 PRE(25),/* STRUCTURE TO COMMUNICATE WITH LEXT*/
2 OP CHAR (1), /* =,>,<, * /
2 VALUE CHAR (40),
2 VLEN FIXED BIN;
DCL CMD CHAR (8);

PUT SKIP LIST ('TYPE YES IF NEED HELP');
GET EDIT (CMD) (A(8)); IF CMD='YES' THEN CALL HELP;

DO WHILE ('1'B); /*DO FOREVER*/
A1:PUT SKIP LIST ('MODIFY: ENTER ENTITY SET NAME');
CALL LEX (N.TOKEN,1,RTN.CODE);
IF N = 0 THEN LEAVE; /*GET OUT OF DMC*/
T.ENAME = TOKEN.D(1);

```

FILE: DMM PLOIOT A1 VM/SP CONVERSATIONAL MONITOR SYSTEM

```

A:PUT SKIP LIST ('ENTER MODIFICATION OPERATORS AND NAMES OF ATTRIBUTED
S SEPARATED BY COMMAS');
MODE=1; /*SUPPRESS BLANKS*/
CALL LEX (N, TOKEN, MODE, RTN_CODE); /*GET INPUT LINE THRU LEX*/
IF RTN_CODE=0 THEN DO;
  PUT SKIP LIST ('IMPROPER SYNTAX (LEX). PLEASE RE-ENTER');
  GO TO A;
END;

IF N=0 THEN LEAVE; /*GET OUT OF DMC*/

CALL LEX (N, TOKEN, T1, T, IN, PRED, OP, RTN_CODE); /*BUILD T STRUCTURE*/
IF RTN_CODE=0 THEN DO;
  PUT SKIP LIST ('IMPROPER SYNTAX, PLEASE RE-ENTER');
  GO TO A;
END;

/*CALL UP VNAME TO CHECK FOR LEGALITY OF NAMES*/
CALL VNM1(T, RTN_CODE, N1); /* VNM1 WILL PRINT ERR MSG IF ANY */
IF N1 = -1 THEN GO TO A1; /* ESET NAME ERROR */
ELSE IF N1 ^= 0 THEN GO TO A; /* ATTR NAME ERROR */

/*START ASKING FOR DATA THROUGH SUBROUTINE UPE1*/
OPE='M';
PUT SKIP LIST ('ENTER DATA:');

CALL UPE1 (OPE, T, OP, RTN_CODE); /* UPE1 WILL PRINT ERR MSG */
END /*DO WHILE*/;
RETURN;

/***** INTERNAL SUBROUTINE *****/
HELP: PROC; /****** BRIEF EXPLANATION ON HOW TO DO MODIFY *****
*****);
PUT SKIP LIST ('AT THE PROMPT FOR MODIFICATION OPERATORS AND ATTRI-');
PUT SKIP LIST ('BUTE NAMES PLEASE ENTER THE OPERATOR (INSERT,DELETE,');
PUT SKIP LIST ('REPLACE) PREFIXED WITH "- ", AND NAMES OF ATTRIBUTES');
PUT SKIP LIST ('THAT WILL BE THUS MODIFIED. YOU SHOULD ALSO PROVIDE');
PUT SKIP LIST ('A SPECIAL OPERATOR (ID) FOR NAMES OF ATTRIBUTES USED');
PUT SKIP LIST ('TO IDENTIFY THE ENTITY TO BE MODIFIED. AN EXAMPLES');
PUT SKIP LIST ('FOLLOWS:');
PUT SKIP LIST ('TO CHANGE THE DEPARTMENT AN EMPLOYEE (WHOSE');
PUT SKIP LIST ('EMPLOYEE NUMBER WILL BE GIVEN) WORKS IN TO ANOTHER');
PUT SKIP LIST ('DEPARTMENT (WHOSE DEPARTMENT NUMBER WILL BE GIVEN)');
PUT SKIP LIST ('ISSUE THE FOLLOWING MODIFICATION STATEMENT AT THE');
PUT SKIP LIST ('PROMPT FOR ATTRIBUTE NAMES AND MODIFICATION');
PUT SKIP LIST ('OPERATORS:');
PUT SKIP LIST ('-ID:EMPNUM,-REP:WORKS_IN(DEPTNUM)');
PUT SKIP LIST ('AND LATER ENTER DATA FOR EMPNUM AND DEPTNUM. ');
PUT SKIP LIST ('***** NOW TRY IT *****');
END /*HELP*/;
END /*DMM*/;

```

```
DUM00010 *****  
DUM00020 /******  
*  
*          MODULE      DESCRIPTION  
*          *  
*          *  
/******  
DUM00030 *****  
DUM00040 *****  
DUM00050 *****  
DUM00060 *****/  
DUM00070 *****  
DUM00080 *****  
DUM00090 ***** S-PROC AT THE USER INTERFACE LEVEL. CALLED BY THE DMB MODULE, TO  
DUM00100 ***** PROCESS QUERY REQUESTS ISSUED BY THE USERS WITHIN A DATA  
DUM00110 ***** MANIPULATION - QUERY SESSION.  
DUM00120 ***** METHOD:  
DUM00130 ***** 1. CALLS LEX TO PARSE INPUT STRING INTO TOKENS.  
DUM00140 ***** 2. CALLS TEXT TO BUILD NAME TREE AND PREDICATE TABLE.  
DUM00150 ***** 3. SET UP RETE CALL TO THE NEXT LEVEL.  
DUM00160 ***** 4. CALL PRINT TO PRINT RESULT.  
DUM00170 *****  
DUM00180 *****  
DUM00190 ***** CALLS PROCEDURES:  
DUM00200 ***** INTER-LEVEL T-PROC: RETE.  
DUM00210 ***** INTRA-LEVEL T-PROC: NONE.  
DUM00220 ***** INTRA-LEVEL S-PROC: LEX,TEXT,PRINT  
DUM00230 ***** CONTROL STRUCTURE, PANEL MANAGER AND DEBUGGING FACILITIES:  
DUM00240 ***** TCALL,SVC3  
DUM00250 *****  
DUM00260 *****/  
DUM00270 *****  
DUM00280 *****%INCLUDE ELEX,ELEXT,SERVICE,ARETE,EPRNT;  
DUM00290 ***** DCL 1 T. /*DATA STRUCTURE USED TO COMMUNICATE WITH LEXT*/  
DUM00300 ***** 2 ENAME CHAR(8),  
DUM00310 ***** 2 TN FIXED BIN. /*NUMBER OF ATTR*/  
DUM00320 ***** 2 T1 (25).  
DUM00330 ***** 3 PARENT FIXED BIN,  
DUM00340 ***** 3 ANAME CHAR (8);  
DUM00350 ***** DCL RTN CODE FIXED BIN;  
DUM00360 ***** DCL N FIXED BIN; /*THE FOLLOWING USED TO CALL LEX*/  
DUM00370 ***** DCL 1 TOKEN(25),  
DUM00380 ***** 2 D CHAR(40).  
DUM00390 ***** 2 L FIXED BIN;  
DUM00400 ***** DCL MODE FIXED BIN;  
DUM00410 ***** DCL N1 FIXED BIN;  
DUM00420 ***** DCL 1 PRED(25). /* PRED AND MOP COMMUNICATE WITH LEXT */  
DUM00430 ***** 2 OP CHAR (1), /* =,>,<, +/  
DUM00440 ***** 2 VALUE CHAR (40),  
DUM00450 ***** 2 VLEN FIXED BIN;  
DUM00460 ***** DCL MOP (25) CHAR (1); /* NOT USED IN DMQ */  
DUM00470 ***** DCL CMD CHAR (8);  
DUM00480 ***** DCL (TP,P.RP,LP) PTR. NULL BUILTN ;  
DUM00490 ***** DCL 1 FIXED BIN (31);  
DUM00500 *****  
DUM00510 ***** /*BEGIN PROCESSING*/  
DUM00520 *****  
DUM00530 ***** PUT SKIP LIST ('--QUERY SESSION--');  
DUM00540 ***** PUT SKIP LIST ('TYPE YES IF NEED HELP');  
DUM00550 ***** GET EDIT (CMD) (A(B));
```

FILE: DMQ PLIOPT A1 VM/SP CONVERSATIONAL MONITOR SYSTEM

```

IF CMD='YES' THEN CALL HELP;

/* MAIN COURSE */
DO WHILE ('1'B); /*DO FOREVER*/
A1:PUT SKIP LIST ('ENTER ENTITY SET NAME');
CALL LEX (N.TOKEN,1,RTN_CODE);
IF N = 0 THEN LEAVE; /*GET OUT OF DMC*/
T.ENAME = TOKEN.D (1);

A:PUT SKIP LIST ('ENTER ATTRIBUTE NAMES AND PREDICATE, SEPARATED BY
COMMAS');
MODE=1; /*SUPPRESS BLANKS*/
CALL LEX (N.TOKEN, MODE,RTN_CODE);/*GET INPUT LINE THRU LEX*/
IF RTN_CODE=0 THEN DO;
PUT SKIP LIST ('IMPROPER SYNTAX (LEX). PLEASE RE-ENTER');
GO TO A;
END;

IF N=0 THEN LEAVE; /*GET OUT OF DMC*/

CALL LEX (N.TOKEN,T1,T.TN,PRED,MOP,RTN_CODE); /*BUILD T STRUC*/
IF RTN_CODE=0 THEN DO;
PUT SKIP LIST ('IMPROPER SYNTAX. PLEASE RE-ENTER');
GO TO A;
END;

/* SET UP CALL TO RETE */

ALLOCATE RETE_ARG SET (RP); /*ROOT NODE */
LP=RP;
RP->RETE_ARG.NAME=T.ENAME;
RP->RETE_ARG.GET =RETEGET.ALL;
RP->RETE_ARG.N =0; /*NO PREDICATE */

DO I=1 TO T.TN; /* ATTRIBUTE NODES */
ALLOCATE RETE_ARG; /* P->RETE_ARG */
LP->RETE_ARG.PTR=P; LP=P;
RETE_ARG.NAME=T.ANAME(I);
RETE_ARG.PARENT=T.PARENT(I)+1;
RETE_ARG.NODE=I+1;
RETE_ARG.GET= RETEGET.ANY;

/* PREDICATE */
IF PRED.VLEN(I)^=0 THEN DO;
RETE_ARG.N = 1;
SELECT (PRED.OP(I));
WHEN ('>') RETE_ARG.OP = RETEOP.GT;
WHEN ('<') RETE_ARG.OP = RETEOP.LT;
WHEN ('=') RETE_ARG.OP = RETEOP.EQ;
OTHERWISE;
END /* SELECT */;
RETE_ARG.CDATA = PRED.VALUE(I);
RETE_ARG.DLEN = PRED.VLEN(I);
END /* PREDICATE */;
END /*DO I */;

```

FILE: L...Q PLIOPT A1 VM/SP CONVERSATIONAL MONITOR SYSTEM

```

CALL TCALL ('RETE', 1, RP, TP); /* TP-> RETE RTN */
IF TP->RETE_RTN_CODE ^= 0 /* CHECK RETE RETURN CODE */
THEN DO;
  CALL RETMSG (TP->RETE_RTN_CODE);
  CALL SVC3 (RP); CALL SVC3 (TP);
  GO TO A1;
END;

IF TP->RETE_RTN_N = 0 THEN DO;
  PUT SKIP LIST ('NO DATA FOUND');
  CALL SVC3 (RP); CALL SVC3 (TP);
  GO TO A1;
END;

CALL PRNT (RP, TP, RTN_CODE);
CALL SVC3 (RP); CALL SVC3 (TP);
END /*DO WHILE */;

/***** INTERNAL SUBROUTINES *****/
HELP: PROC; /*INTERNAL SUBROUTINE */
PUT SKIP LIST ('***** BRIEF EXPLANATION ON HOW TO USE QUERY *****
');
PUT SKIP LIST ('THE SYSTEM WILL ASK YOU TO PROVIDE THE NAMES');
PUT SKIP LIST ('OF THE ENTITY SET YOU ARE TO QUERY ABOUT. ');
PUT SKIP LIST ('AND THE LIST OF ATTRIBUTES OF THIS ENTITY SET');
PUT SKIP LIST ('OF INTEREST TO YOU. YOU MAY ALSO PROVIDE PREDICATES');
PUT SKIP LIST ('ALONG WITH THE LIST OF ATTRIBUTES. ATTRIBUTES IN');
PUT SKIP LIST ('YOUR ATTRIBUTE LIST SHOULD BE SEPARATED BY COMMAS. ');
PUT SKIP LIST ('THE FOLLOWING IS AN EXAMPLE OF AN ATTRIBUTE LIST');
PUT SKIP LIST ('PROVIDED BY A USER QUERYING ABOUT AN ENTITY SET');
PUT SKIP LIST ('EMPLOYEE. THIS USER IS INTERESTED IN THE EMPNUM');
PUT SKIP LIST ('AND EMPNAME OF EMPLOYEES OLDER THAN 56 AND WORK IN');
PUT SKIP LIST ('DEPT WITH DEPTNUM 15. ');
PUT SKIP LIST ('EMPNUM,EMPNAME,AGE>56,DEPT(DEPTNUM=15)');
PUT SKIP LIST ('***** NOW TRY IT *****');
END /* HELP */;

/***** INTERNAL SUBROUTINE FOR RETE ERROR */
REMSG: PROC (CODE);
DCL CODE FIXED BIN;
IF CODE = 100 THEN PUT SKIP LIST ('NO SUCH ENTITY SET. ');

IF CODE > 100 & CODE < 200
THEN PUT SKIP EDIT ('ILLEGAL ATTRIBUTE ', T.ANAME(CODE-101))(A,A);

IF CODE < 100
THEN PUT SKIP EDIT ('ILLEGAL PREDICATE DATA FOR ', T.ANAME(CODE-101))
(A,A);

IF CODE > 200 THEN PUT SKIP EDIT ('RETN RETURN CODE ', CODE)(A,F(3));
PUT SKIP LIST ('PROBLEM IN QUERY STATEMENT: PLEASE REISSUE');
RETURN;
END;

```

FILE DMQ PLIORT A1 VM/SP CONVERSATIONAL MONITOR SYSTEM

...../ DUMO1660
DUMO1670
DUMO1680

END /* DMQ */;

178

```

SELECT(NAME);
WHEN ( 'F' , 'FILE', 'f', 'file' ) DO;
  OUT='1'B;
  PUT SKIP LIST ('FILE NAME?');
  GET EDIT (FNAME) (A(8));
  CALL READ (FNAME); /*READ IS INT SBR TO READ FILE INTO SH*/
  CALL NEXT1; /*INT SUBR TO SET UP CALL FOR FILE INIT AT NEXT LV*/
END;

WHEN ( 'N' , 'NEW', 'n', 'new' ) DO; /* NEW INIT */
  ALLOCATE PACKET IN (SH);
  OUT='1'B;
  CALL RINIT; /*REL VIEW PROCESSOR INIT*/
  CALL CINIT; /*BN VIEW PROCESSOR INIT*/
  FNAME=(8) , ' ';
  CALL NEXT1; /*INT SBR TO SET UP CALL FOR NEW INIT IN NEXT LV*/
END;

OTHERWISE PUT SKIP EDIT(NAME, ' IS NOT A COMMAND')(A,A);
END;
IF OUT THEN LEAVE;

END;
/*AFTER INITIALIZATION, CONSOLE MAY CHOOSE TO ENTER USER MODE*/
CALL TCALL ('USER', 1, TP, P);

/*CONSOLE LOG OUT AND SAVE FILE*/
PUT SKIP LIST ('SAVE FILE: FILE NAME?');
GET EDIT (FNAME) (A(8));
IF FNAME ^= '' THEN CALL SAVE(FNAME);
/*SAVE IS INT SBR TO SET UP CALL TO NEXT LEV FOR SAVING DATA*/

/***** TIMING REPORT OPTION *****/
ON ZERO DIVIDE;
DCL ((ETIME, ELO, ECOUNT)(50) FIXED BIN(31)) EXT;
DCL FTIME BIT(1) EXT;
XINCLUDE FSTRUC;
IF FTIME THEN DO;
  PUT EDIT ('LEVEL PROGRAMME COUNT TOT ELAPSED TIME TOT RUN TIME RUNDUM00960
TIME/INVOCATION')(SKIP, A);
  PUT EDIT ('-----') (SKIP, A);
  PUT EDIT (((ARCH.LEVEL(1), ARCH.PROCNAME(1), ECOUNT(1), ETIME(1),
(ETIME(1)-ELO(1)),
(ETIME(1)-ELO(1))/ECOUNT(1))
DO I=1 TO 19))
(SKIP, F(3), X(4), A, F(7), F(17), F(13), F(21));
END;
/*****
/*
CALL FINISH; /*SUICIDE****/
FREE SH;

```

DUM00560
 DUM00570
 DUM00580
 DUM00590
 DUM00600
 DUM00610
 DUM00620
 DUM00630
 DUM00640
 DUM00650
 DUM00660
 DUM00670
 DUM00680
 DUM00690
 DUM00700
 DUM00710
 DUM00720
 DUM00730
 DUM00740
 DUM00750
 DUM00760
 DUM00770
 DUM00780
 DUM00790
 DUM00800
 DUM00810
 DUM00820
 DUM00830
 DUM00840
 DUM00850
 DUM00860
 DUM00870
 DUM00880
 DUM00890
 DUM00900
 DUM00910
 DUM00920
 DUM00930
 DUM00940
 DUM00950
 DUM00960
 DUM00970
 DUM00980
 DUM00990
 DUM01000
 DUM01010
 DUM01020
 DUM01030
 DUM01040
 DUM01050
 DUM01060
 DUM01070
 DUM01080
 DUM01090
 DUM01100

FILE: DUMFSTV PLIOPT A1 VM/SP CONVERSATIONAL MONITOR SYSTEM

```

/*****INTERNAL SUBROUTINES*****/
NEXTI: PROC: /* INT SBR FOR PASSING INIT DOWN NEXT LEVEL*/
  ALLOCATE VINIT_ARG; /*SET UP ARG LIST*/
  VINIT_ARG.PTR=NULL;
  VINIT_ARG.FNAME=FNAME;
  IF FDEBUG.LEVO THEN PUT SKIP LIST ('FSTV CALLS VINIT');
  CALL TCALL ('VINIT', 1.P.TP);
  IF FDEBUG.LEVO THEN PUT SKIP LIST ('FSTV: VINIT RETURNS');
  CALL SVC3 (P);
  IF TP->VINIT.RTN.CODE ^=0 THEN DO;
    PUT SKIP LIST ('ERROR IN INITIALIZATION, RESTART');
    OUT='O'B; /*REST INIT LOOP*/
  END;
  CALL SVC3 (TP);
  END /*NEXTI*/;

/*****PROC(FNAME): /* SAVE EVERYTHING IN STORAGE HIERARCHY*/
  DCL FNAME CHAR(8);
  OPEN FILE(SAVESH) TITLE(FNAME) OUTPUT;
  WRITE FILE(SAVESH) FROM(SH);
  WRITE FILE(SAVESH) FROM(PP); /* OFFSET OF PACKET IN SH */
  CLOSE FILE(SAVESH);
  RETURN;
  END /* SAVE */;

  READ: PROC(FNAME);/*SBR TO READ FILE INTO STORAGE HIERARCHY*/
  DCL FNAME CHAR(8);
  OPEN FILE(SAVESH) TITLE(FNAME) INPUT;
  READ FILE(SAVESH) INTO(SH);
  READ FILE(SAVESH) INTO(PP);
  CLOSE FILE(SAVESH);
  RETURN;
  END;

/*****CINIT: PROC;
  /*IN THE FUTURE, THIS MODULE INITIALIZES THE BASE DATA MODEL VIEW
  SUBSYSTEM; PRESENTLY IT IS NOT AVAILABLE*/
  RETURN;
  END;

  RINIT: PROC;
  /*IN THE FUTURE, THIS MODULE INITIALIZES THE RELATIONAL VIEW
  SUBSYSTEM; PRESENTLY IT IS NOT AVAILABLE*/
  RETURN;
  END;

  END /*FSTV*/;

```

DUM01110
 DUM01120
 DUM01130
 DUM01140
 DUM01150
 DUM01160
 DUM01170
 DUM01180
 DUM01190
 DUM01200
 DUM01210
 DUM01220
 DUM01230
 DUM01240
 DUM01250
 DUM01260
 DUM01270
 DUM01280
 DUM01290
 DUM01300
 DUM01310
 DUM01320
 DUM01330
 DUM01340
 DUM01350
 DUM01360
 DUM01370
 DUM01380
 DUM01390
 DUM01400
 DUM01410
 DUM01420
 DUM01430
 DUM01440
 DUM01450
 DUM01460
 DUM01470
 DUM01480
 DUM01490
 DUM01500
 DUM01510
 DUM01520
 DUM01530
 DUM01540
 DUM01550
 DUM01560
 DUM01570
 DUM01580
 DUM01590
 DUM01600

FILE: DUM1 PLIOPT A1

VM/SP CONVERSATIONAL MONITOR SYSTEM

PAGE 001

L1:PROC(ARGPTR); /* A DUMMY ROUTINE TO SIMULATE ENTIRE SH-STV */
/* USED WHEN RUN WITHOUT CONTROL STRUCTURE AND DSH */

%INCLUDE ARG.SH;

DCL INDX FIXED BIN(31);

INDX = ARG.VIRTUAL_ADDRESS/8 + 1;
IF ARG.REQ_TYPE = 'READ'
THEN ARG.DATA = PACKET(INDX);
ELSE PACKET(INDX) = ARG.DATA;
RETURN;
END;

DUM00010
DUM00020
DUM00030
DUM00040
DUM00050
DUM00060
DUM00070
DUM00080
DUM00090
DUM00100
DUM00110

FILE: DUMPT P1DPT A1 VM/SP CONVERSATIONAL MONITOR SYSTEM

```

DUMPT:PROC (STRING, ID, P);
XINCLUDE DBED;
DCL STRING CHAR(*);
DCL ID BIT(32); DCL (P, BP BASED) PTR;
DCL I FIXED BIN;
DCL PAA(40) PTR BASED;
DCL NULL BUILTIN;
XINCLUDE HEX;

PUT SKIP EDIT('***MEMMGT: ', STRING, ' ID=', HEX(ADDR(ID)->BP))(A, A, A, A);
IF P = NULL THEN GO TO END;

PUT EDIT('NUMPTR=', NUMPTR)(X(1), A, F(2));
DO I = 1 TO NUMPTR;
  PUT EDIT(HEX(ADDR(ID, ARRAY(I)) -> BP))(X(1), A); END;
  PUT EDIT('DLEN=', DLEN)(X(1), A, F(3));
  DO J = 1 TO (DLEN+31)/32; PUT EDIT(HEX4(ADDR(DATA) -> PAA(J)))(A); END;
END;
END;

```

```

DUM00010
DUM00020
DUM00030
DUM00040
DUM00050
DUM00060
DUM00070
DUM00080
DUM00090
DUM00100
DUM00110
DUM00120
DUM00130
DUM00140
DUM00150
DUM00160
DUM00170
DUM00180
DUM00190

```

FILE: DUMTBEG PLIOPI A1

VM/SP CONVERSATIONAL MONITOR SYSTEM

PAGE 001

TBEG: PROC (PROC_ADDR,P):

/**** A SERVICE ROUTINE TO STRIP OFF MSG TOKEN FROM A CALLING ***
*** ARGUMENT LIST; IT IS USED BY A T-PROC WHEN IT IS INVOKED ***/

/**** PROC_ADDR CONTAINS THE PROC_ADDR OF THE CALLING PROCEDURE. ***
*** P WILL POINT TO THE FIRST TOKEN OF THE CALLING ARGUMENT LIST. */
DCL STAT_P1 PTR EXTERNAL STATIC;
DCL P PTR;

P= STAT_P1;
RETURN;
END;

DUM00010
DUM00020
DUM00030
DUM00040
DUM00050
DUM00060
DUM00070
DUM00080
DUM00090
DUM00100
DUM00110
DUM00120
DUM00130
DUM00140

FILE: DUMTCALL PLIOPT A1 VM/SP CONVERSATIONAL MONITOR SYSTEM

```

(SUBRG);
TCALL:PROC (PROCNAME,I.P1,P2) RECURSIVE :
/*THIS IS A DUMMY TCALL TO HANDLE PROBLEM OF CONTROL STRUCTURE*/
/* IT IS USED TO INSULATE FH MODULES FROM CONTROL STRUCTURE */

DCL FTCALL(4) BIT(1) STATIC EXT; /* FTCALL IS DEBUG TRACE BIT */
DCL FTIME BIT(1) EXT; /* FTIME IS TIMING TRACE BIT */
XINCLUDE FSTRUC; /* FSTRUC IS STRUCTURE OF FH */
DCL PROCNAME CHAR(7) ;
DCL (VINIT,DEFE,RETE,UPDE,DEFB,DEFB,RETN) ENTRY;
DCL (UPDN,DEFA,NINIT,VNME,SHWE) ENTRY;
DCL (MINIT,RET,REP,DEL,CRT,FSTV,USER) ENTRY;
DCL I FIXED BIN, (STAT P1,STAT P2) PTR EXTERNAL STATIC;
DCL (P1,P2)PTR; DCL (BEGTIME,ENUM,LOTIME CTL) FIXED BIN(31);
DCL RTIMER ENTRY RETURNS(FIXED BIN(31));
DCL (ECOUNT(50),ELO(50),ETIME(50)) FIXED BIN(31) EXT INIT((50)0);
STAT P1=P1;

DO ENJM = 1 TO 18;
IF ARCH.PROCNAME(ENUM) = PROCNAME THEN LEAVE;
END;
I=ARCH.LEVEL(ENUM);

/***** TCALL TRACE *****/
IF FTCALL(1) /* IF TCALL TRACE BIT FOR THAT LEVEL IS ON */
THEN PUT SKIP EDIT ('TCALL: ',PROCNAME)(A,A);
/*****

/***** TIMING REPORT *****/
IF FTIME THEN DO;
ALLOCATE LOTIME; LOTIME=0;
BEGTIME-RTIMER.END;
/*****
SELECT (PROCNAME);
  WHEN ( 'FSTV ' ) CALL FSTV;
  WHEN ( 'VINIT ' ) CALL VINIT;
  WHEN ( 'VNME ' ) CALL VNME;
  WHEN ( 'DEFE ' ) CALL DEFE;
  WHEN ( 'RETE ' ) CALL RETE;
  WHEN ( 'UPDE ' ) CALL UPDE;
  WHEN ( 'DEEP ' ) CALL DEFP;
  WHEN ( 'SHWE ' ) CALL SHWE;
  WHEN ( 'DEFB ' ) CALL DEFB;
  WHEN ( 'RETN ' ) CALL RETN;
  WHEN ( 'UPDN ' ) CALL UPDN;
  WHEN ( 'DEFA ' ) CALL DEFA;
  WHEN ( 'NINIT ' ) CALL NINIT;
  WHEN ( 'RET ' ) CALL RET;
  WHEN ( 'DEL ' ) CALL DEL;
  WHEN ( 'CRT ' ) CALL CRT;
  WHEN ( 'REP ' ) CALL REP;
  WHEN ( 'MINIT ' ) CALL MINIT;
  WHEN ( 'USER ' ) CALL USER;

```

FILE: DUMICALL PLIOPT A1 VM/SP CONVERSATIONAL MONITOR SYSTEM

```

OTHERWISE DO:PUT SKIP EDIT (' *** NAME NOT FOUND') (A);
      SIGNAL ERROR; END;
END;

/*WHEN RETURNED*/
/****** TIMING REPORT *****/
IF FTIME THEN DO:
  BEGTIME-RTIME-BEGTIME;
  ETIME(ENUM)=ETIME(ENUM)+BEGTIME;
  ECOUNT(ENUM)=ECOUNT(ENUM)+1;
  ELO(ENUM)=ELO(ENUM)+LOTIME;
  FREE LOTIME; IF ALLOCATION(LOTIME)^= 0 THEN LOTIME=LOTIME+BEGTIME;
END;
/******

/****** TCALL TRACE *****/
IF FTCALL(1) THEN
  PUT EDIT(' ***TCALL: END ',PROCNAME)(A,A);
/******
P2=STAT_P2;

END /* TCALL*/;

```

DUM00560
 DUM00570
 DUM00580
 DUM00590
 DUM00600
 DUM00610
 DUM00620
 DUM00630
 DUM00640
 DUM00650
 DUM00660
 DUM00670
 DUM00680
 DUM00690
 DUM00700
 DUM00710
 DUM00720
 DUM00730
 DUM00740
 DUM00750
 DUM00760
 DUM00770

FILE: DUMTRN PLIOPT A1 VM/SP CONVERSATIONAL MONITOR SYSTEM

```

TRTN: PROC(PROC_ADDR, RTP);
/**** A SERVICE ROUTINE USED BY A T-PROC WHEN IT FINISHES PROCESSING
**** IT ADDS A MSG TOKEN TO A RETURN ARGUMENT LIST AND CALLS SVCS2
*** AND THEN FINISHES ITSELF ****/
/*** PROC_ADDR CONTAINS THE ADDR OF THE CALLING PROCEDURE TO RETURN
*** THE RETURN MSG TO; RTP POINTS TO THE RETURN ARGUMENT LIST ***/

XINCLUDE FDEBUG;
DCL (P,RTP) PTR, RTN_CODE FIXED BIN;
DCL 1 T BASED (P),
    2 LEN FIXED BIN (15);
DCL STAT_P2_P2 PTR EXTERNAL STATIC;
STAT_P2=RTP;

END /*TRTN*/;
DUM00010
DUM00020
DUM00030
DUM00040
DUM00050
DUM00060
DUM00070
DUM00080
DUM00090
DUM00100
DUM00110
DUM00120
DUM00130
DUM00140
DUM00150
DUM00160
DUM00170
DUM00180
DUM00190

```

```

EBOT:PROC:
/*A S-PROC AT THE ENTITY LEVEL USED BY VINIT MODULE TO BOOTSTRAP
** ENTITY CATALOGUE ENTRIES *****
XINCLUDE AUPON.SERVICE, DKEY, FDEBUG;
DCL (STRING, UNSPEC) BUILTIN;
DCL (I,K,J) FIXED BIN, (LP,RP,TP,P)PTR, NULL BUILTIN;
DCL 1 MAINFO (5), /*AN ARRAY OF TEMPLATE AINFO*/
2 BSETID BIT (32), /*BSETID OF THIS ATTRIBUTE**/
2 PSETID BIT (32), /*PSETID OF TARGET NODE**/
2 ENAME CHAR (8) INIT (' ',' ','E','E','S','E','T'),
2 MAX FIXED BIN (31),
2 MIN FIXED BIN (31),
2 MLEN FIXED BIN INIT (8, 4,0,8,29),
2 FUNC CHAR(1) INIT ('K','S','M','S','S'),
2 TYPE CHAR (1) INIT('V','V','E','V','V'),
2 VTYPE CHAR (1) INIT ('C','B',' ','C','B');
DCL BIT_STRING BIT(232) BASED; /*BIT STRING FORM OF AINFO*/
DCL BAINFO(5) BIT(232);
DCL SEQ(5) FIXED BIN INIT (2,3,1,5,6);
DCL ADDR BUILTIN;

/*BEGIN PROCESSING*/
/*SET UP BAINFO*/
DO I=1 TO 5;
  MAINFO.BSETID(I)=KEY(I+6);
  MAINFO.PSETID(I)=KEY(SEQ(I));
  BAINFO(I)=ADDR(MAINFO(I))-BIT_STRING;
END;

DO J=1 TO 2; /*FOR TWO ENTITY SETS ESET AND ASET*/
  ALLOCATE UPDN_ARG; LP,RP=P;
  UPDN_ARG.PSETID=KEY(KY.PESET);
  UPDN_ARG.OP=UPDNOP.CRT;
  UPDN_ARG.DLEN=0;

  DO I=1 TO 2;
    ALLOCATE UPDN_ARG;
    LP->UPDN_ARG.PTR=P; LP=P;
    UPDN_ARG.PARENT=1;
    UPDN_ARG.OP=UPDNOP.CI; /*CI IS CREATE AND INSERT*/
    UPDN_ARG.NODE=I+1;

  SELECT (I);
  WHEN (1) DO;
    UPDN_ARG.PSETID=KEY(KY.PENAME);
    UPDN_ARG.BSETID=KEY(KY.BENAME);
    UPDN_ARG.DLEN=8;
    IF J=1 THEN UPDN_ARG.DATA=UNSPEC ('E','E','S','E','T',' ');
    ELSE UPDN_ARG.DATA=UNSPEC ('E','A','S','E','T',' ');
  END /*WHEN 1*/;

  WHEN (2) DO;
    UPDN_ARG.PSETID=KEY(KY.PEINFO);
    UPDN_ARG.BSETID=KEY(KY.BEINFO);

```

FILE: L-LJT PLOIOT A1 VM/SP CONVERSATIONAL MONITOR SYSTEM

```

UPDN_ARG.DLEN=4;
IF J=1 THEN UPDN_ARG.DATA=KEY(KY.PESET);
ELSE UPDN_ARG.DATA=KEY(KY.PASET);
END /*WHEN 2*/;

END /*SELECT 1*/;

END /*DO 1*/;
CALL TCALL ('UPDN',1,RP,TP); /*CALL UP UPDN*/
END /*DO J*/;

DO J=1 TO 5; /*5 ATTRIBUTE SETS IN THE CATALOGUE*/
  ALLOCATE UPDN_ARG; RP,LP=P;
  UPDN_ARG.PSETID=KEY(KY.PASET);
  UPDN_ARG.OP=UPDNOP.CRT;
  UPDN_ARG.DLEN=0;

  DO I=1 TO 4;
    ALLOCATE UPDN_ARG;
    LP->UPDN_ARG.PTR=P; LP=P;
    UPDN_ARG.NODE=I+1; /*ASSIGN NODE NUMBER*/
    SELECT (I);

    WHEN (1) DO; /*ANAME NODE*/
      UPDN_ARG.PSETID=KEY(KY.PANAME);
      UPDN_ARG.PARENT=1;
      UPDN_ARG.BSETID=KEY(KY.BANAME);
      UPDN_ARG.OP=UPDNOP.C1;
      UPDN_ARG.DLEN=8;
      SELECT (J);
      WHEN (1) UPDN_ARG.DATA=UNSPEC('A*ANAME ');
      WHEN (2) UPDN_ARG.DATA=UNSPEC('A*EINFO ');
      WHEN (3) UPDN_ARG.DATA=UNSPEC('A*ESET ');
      WHEN (4) UPDN_ARG.DATA=UNSPEC('A*ANAME ');
      WHEN (5) UPDN_ARG.DATA=UNSPEC('A*AINFO ');
      END /*SELECT J*/;
      END /*WHEN 1*/;

    WHEN (2) DO; /*ESET NODE*/
      UPDN_ARG.PSETID=KEY(KY.PESET);
      UPDN_ARG.PARENT=1;
      UPDN_ARG.BSETID=KEY(KY.BESET);
      UPDN_ARG.OP=UPDNOP.IST;
      UPDN_ARG.DLEN=0;
      END /*WHEN 2*/;

    WHEN(3) DO; /*ENAME NODE*/
      UPDN_ARG.PSETID=KEY(KY.PENAME);
      UPDN_ARG.PARENT=3;
      UPDN_ARG.BSETID=KEY(KY.BENAME);
      UPDN_ARG.OP=UPDNOP.ID;
      UPDN_ARG.DLEN=8;
      IF J<3 THEN
        UPDN_ARG.DATA=UNSPEC('E*ESET ');
      ELSE UPDN_ARG.DATA=UNSPEC('E*ASET ');
    END /*WHEN 3*/;
  END /*DO J*/;

```

EB000560
EB000570
EB000580
EB000590
EB000600
EB000610
EB000620
EB000630
EB000640
EB000650
EB000660
EB000670
EB000680
EB000690
EB000700
EB000710
EB000720
EB000730
EB000740
EB000750
EB000760
EB000770
EB000780
EB000790
EB000800
EB000810
EB000820
EB000830
EB000840
EB000850
EB000860
EB000870
EB000880
EB000890
EB000900
EB000910
EB000920
EB000930
EB000940
EB000950
EB000960
EB000970
EB000980
EB000990
EB001000
EB001010
EB001020
EB001030
EB001040
EB001050
EB001060
EB001070
EB001080
EB001090
EB001100

FILE: EBOT PLIOPT A1 VM/SP CONVERSATIONAL MONITOR SYSTEM

```

END /*3*/;

WHEN (4) DO: /*AINFO NODE*/
  UPDN_ARG.PSETID=KEY(KY.PAINFO);
  UPDN_ARG.PARENT=1;
  UPDN_ARG.BSETID=KEY(KY.BAINFO);
  UPDN_ARG.OP=UPDNOP.CI;
  UPDN_ARG.DLEN=29;
  UPDN_ARG.DATA=STRING (BAINFO(J));
END /*4*/;

END /*SELECT I */;
END /*DO I */;

CALL TCALL ('UPDN',1, RP, TP);
END /*DO J */;
END /*EBOT*/;

```

```

EB001110
EB001120
EB001130
EB001140
EB001150
EB001160
EB001170
EB001180
EB001190
EB001200
EB001210
EB001220
EB001230
EB001240
EB001250
EB001260
EB001270

```

FILE: FHL1 PLI0PT A1 VM/SP CONVERSATIONAL MONITOR SYSTEM

L1:PROC: /* A DUMMY ROUTINE TO SIMULATE ENTIRE SH-STV */
 *INCLUDE USERS.VPX,ARG.SH;

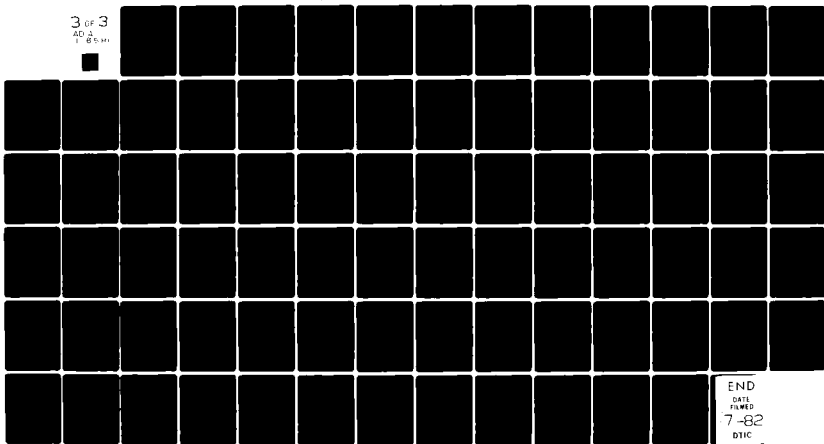
ARGPTR = VP.WAIT.MSG;
 INDX = ARG.VIRTUAL_ADDRESS/8 + 1;
 IF ARG.REQ_TYPE = 'READ'
 THEN ARG.DATA = PACKET(INDX);
 ELSE PACKET(INDX) = ARG.DATA;
 CALL SEND (0,ARG.VPID,ARG.BOXID,'S',20,ARGPTR);
 CALL FINISH;
 RETURN;
 END;

L1 00010
 L1 00020
 L1 00030
 L1 00040
 L1 00050
 L1 00060
 L1 00070
 L1 00080
 L1 00090
 L1 00100
 L1 00110
 L1 00120

AD-A116 591

ALFRED P SLOAN SCHOOL OF MANAGEMENT CAMBRIDGE MA CEN--ETC F/6 9/2
FSTV: THE SOFTWARE TEST VEHICLE FOR THE FUNCTIONAL HIERARCHY OF--ETC(U)
JAN 82 M HSU N00039-81-C-0663
UNCLASSIFIED CISR-M010-8201-09 NL

3 of 3
AD-A
1 86 H



END
DATE
FILMED
7-82
DTIC

FILE: FSTV PLI0PT A1 VM/SP CONVERSATIONAL MONITOR SYSTEM

```

FSTV: PROC;
/*****
*
*      MODULE      DESCRIPTION
*
*      PURPOSE:  A T-PROC FOR CONSOLE INITIALIZATION FROM
*      AN EXISTING FILE OR FROM SCRATCH, AND CONSOLE LOGOUT.
*
*      METHOD:
*      IT ASKS FOR INIT METHOD FROM THE CONSOLE USER.  IF FILE INIT,
*      IT ASKS FOR FILE NAME AND PERCOLATES THE FILE INIT DOWN.  IF
*      NEW INIT, IT CALLS SBRS TO INITIALIZE CAT FILES OF THIS LEVEL
*      AND PERCOLATES THE NEW INIT DOWN.
*
*      CALLS PROCEDURES:  RINIT, CINIT, VINIT, VSAVE, USER;
*      (INTRA-LEVEL S-PROCS:  RINIT, CINIT, USED FOR INITIALIZING THE
*      RELATIONAL VIEW PROCESSOR AND THE BN VIEW PROCESSOR, BOTH
*      ARE CURRENTLY INCLUDED IN THE TOP LEVEL BUT EVENTUALLY WILL
*      BE DEDICATED TO THE VTL, WHICH DOES NOT EXIST YET.
*      USER:  ENTER INTO USER MODE FROM CONSOLE.
*      INTER-LEVEL T-PROC:  VINIT, VSAVE:  USED FOR PERCOLATING INIT
*      AND SAVE COMMANDS TO THE NEXT LEVEL.)
*
*      ON ERROR SNAP SYSTEM;
*      %INCLUDE EUSER; /*INTRA-LEVEL S-PROC ENTRY DCLS*/
*      %INCLUDE AVINI; /*INTER-LEVEL T-PROC ARG LIST*/
*      %INCLUDE SERVICE,USERS,VPX /*CONTROL STRUCTURE SERVICES*/;
*      %INCLUDE FDEBUG;
*      DCL NULL BUILTIN;
*      DCL RTN CODE FIXED BIN(15);
*      DCL (TP_P) POINTER;
*      DCL (NAME, FNAME) CHAR(8);
*      DCL (SETH) ENTRY;
*      /* BEGIN SESSION */
*      PUT SKIP (2) LIST ('--FUNCTIONAL HIERARCHY STV CONSOLE PROGRAM--');
*      CALL SETH; /* SETTING FSTV TRACE OPTIONS */
*      CALL NEXT; /* INITIALIZE SYSTEM CATALOGUES OF FSTV */
*      CALL TCALL('USER',1,P,TP); /* THEN START USER PROCESSING */
*      /* P AND TP IRRELEVANT */
*
*      /* WHEN USER QUITS USER SESSION */
*      /***** TIMING REPORT OPTION *****/
*      ON ZERODIVIDE;
*      DCL ((TIME,ELC,ECOUNT)(50) FIXED BIN(31)) EXT;
*      DCL FTIME BIT(1) EXT;
*      %INCLUDE FSTRUC;
*      IF FTIME THEN DO;
*      PUT EDIT ('LEVEL    PROCNAME    COUNT    TOT ELAPSED TIME    TOT RUN TIME    RUNFSTO0520
*      TIME/INVOCATION')(SKIP,A);
*      PUT EDIT ('-----') (SKIP,A);
*      FSTO0550
*      FSTO0550

```

FILE: FSV PLIOPT A1 VM/SP CONVERSATIONAL MONITOR SYSTEM

```

PUT ED:(((ARCH.LEVEL(1),ARCH.PROCNAME(1),ECOUNT(1),ETIME(1),
(ETIME(1)-ELO(1)),
(ETIME(1)-ELO(1))/ECOUNT(1))
DO I=1 TO 19))
(SKIP.F(3),X(4),A.F(7),F(17),F(13),F(21));
END;
/*****
CALL FINISH; /* SUICIDE */

/****INTERNAL SUBROUTINES*****/
NEXTI: PROC; /* INT SBR FOR PASSING INIT DOWN NEXT LEVEL*/
GETI:
PUT LIST ('REGENERATE(NEW) OR OLD DATA BASE(FILE):');
GET EDIT (FNAME) (A(8));
SELECT (FNAME);
  WHEN ('N','NEW','n','new') FNAME = '';
  WHEN ('F','FILE','f','file') FNAME = 'F';
  OTHERWISE GO TO GETI; END /*SELECT */;
ALLOCATE VINIT ARG; /*SET UP ARG LIST*/
VINIT_ARG.PTR=NULL;
VINIT_ARG.FNAME=FNAME;
IF FDEBUG.LEVO THEN PUT SKIP LIST ('FSTV CALLS VINIT');
CALL TCALL ('VINIT',1,P,TP);
IF FDEBUG.LEVO THEN PUT SKIP LIST ('FSTV: VINIT RETURNS');
CALL SVC3 (P);
CALL SVC3 (TP);
END /*NEXTI*/;

/****
CINIT: PROC;
/*IN THE FUTURE, THIS MODULE INITIALIZES THE BASE DATA MODEL VIEW
SUBSYSTEM; PRESENTLY IT IS NOT AVAILABLE*/
RETURN;
END;

RINIT: PROC;
/*IN THE FUTURE, THIS MODULE INITIALIZES THE RELATIONAL VIEW
SUBSYSTEM; PRESENTLY IT IS NOT AVAILABLE*/
RETURN;
END;

END /*FSTV*/;

```

FST00560
FST00570
FST00580
FST00590
FST00600
FST00610
FST00620
FST00630
FST00640
FST00650
FST00660
FST00670
FST00680
FST00690
FST00700
FST00710
FST00720
FST00730
FST00740
FST00750
FST00760
FST00770
FST00780
FST00790
FST00800
FST00810
FST00820
FST00830
FST00840
FST00850
FST00860
FST00870
FST00880
FST00890
FST00900
FST00910
FST00920
FST00930
FST00940
FST00950
FST00960
FST00970
FST00980
FST00990
FST01000

```

GACT: PROC (ANAME, ENAME, AINFO, RTN_CODE);
/*... A S-PROC IN THE ENTITY LEVEL RESPONSIBLE FOR RETRIEVING ATTRI-
*** BUTE CAT ENTRY GIVEN ATTRIB NAME AND ENTITY NAME ******/
XINCLUDE DECAT, OKEY, ARETN, SERVICE, FDEBUG;

DCL (ANAME, ENAME) CHAR (8);
DCL (P, RP, LP, RTP, PTR, (NULL, ADDR) BUILTIN;
DCL BIT STRING BIT (232) BASED;
DCL RTN_CODE FIXED BIN, I FIXED BIN;
DCL UNSPEC BUILTIN, STRING BUILTIN;

/*BEGIN PROCESSING*/
RTN_CODE=0; /*INIT TO O.K.*/

/*RETN TREE ROOT NODE IS ASET*/
I=0; /*NO PREDICATE*/
ALLOCATE RETN_ARG; RP,LP=P;
RETN_ARG.PSETID=KEY(KY.PASET);
RETN_ARG.GET=RETNGET.ANY;
RETN_ARG.N=0; /*NO PREDICATE*/

/*ANAME NODE*/
I=1; /*1 PREDICATE*/
ALLOCATE RETN_ARG;
LP->RETN_ARG.PTR=P; LP=P;
RETN_ARG.PSETID=KEY(KY.PANAME);
RETN_ARG.NODE=2;
RETN_ARG.PARENT=1;
RETN_ARG.BSETID=KEY(KY.BANAME);
RETN_ARG.GET=RETNGET.NO;
RETN_ARG.N=1; /*1 PREDICATE*/
RETN_ARG.PRED.CN(1)=1;
RETN_ARG.PRED.OP(1)=RETNOP.EQ;
RETN_ARG.PRED.DLEN(1)=8;
RETN_ARG.PRED.DATA=UNSPEC (ANAME);

/*ESET NODE*/
I=0;
ALLOCATE RETN_ARG;
LP->RETN_ARG.PTR=P; LP=P;
RETN_ARG.PSETID=KEY(KY.PESET);
RETN_ARG.NODE=3;
RETN_ARG.PARENT=1;
RETN_ARG.BSETID=KEY(KY.BESET);
RETN_ARG.GET=RETNGET.NO;
RETN_ARG.N=0; /*NO PREDICATE*/

/*ENAME NODE*/
I=1; /*1 PREDICATE*/
ALLOCATE RETN_ARG;
LP->RETN_ARG.PTR=P; LP=P;
RETN_ARG.PSETID=KEY(KY.PENAME);
RETN_ARG.NODE=4;

```

```

RETN_ARG.PARENT=3;
RETN_ARG.BSETID=KEY(KY.BENAME);
RETN_ARG.GET=RETGET.NO;
RETN_ARG.N=1; /*1 PREDICATE*/
RETN_ARG.PRED.CN(1)=1;
RETN_ARG.PRED.OP(1)=RETNOP.EQ;
RETN_ARG.PRED.DLEN(1)=8;
RETN_ARG.PRED.DATA=UNSPEC (ENAME);

/*AINFO NODE*/
I=0; /*NO PREDICATE*/
ALLOCATE RETN_ARG;
LP->RETN_ARG.PTR=P; LP=P;
RETN_ARG.PSETID=KEY(KY.PAINFO);
RETN_ARG.NODE=5;
RETN_ARG.PARENT=1;
RETN_ARG.BSETID=KEY(KY.BAINFO);
RETN_ARG.GET=RETGET.ANY;
RETN_ARG.N=0; /*0 PREDICATE*/

/*CALL UP RETN*/
CALL TCALL ('RETN', 1, RP, TP);

/*DECODE RETURN*/
IF TP->RETN.RTN.CODE=0 THEN DO;
  RTN_CODE=1; /*PROBLEM IN RETRIEVAL*/
  CALL SVC3 (RP);
  CALL SVC3 (TP);
  RETURN;
END;

IF TP->RETN.RTN.N=1 THEN DO;
  RTN_CODE=2; /*NO SUCH ATTRIBUTE*/
  CALL SVC3 (RP);
  CALL SVC3 (TP);
  RETURN;
END;

/*GET DATA AND DECODE*/
P=TP->RETN.RTN.PTR->RETN.RTN1.PTR; /*THE 3RD TOKEN RETURNED*/
ADDR(AINFO)->BIT_STRING=RETN.RTN1.DATA;

RETURN: CALL SVC3 (RP); CALL SVC3 (TP);
RETURN: /*RETURNS*/
END /*GACT*/;

```

GAC00560
 GAC00570
 GAC00580
 GAC00590
 GAC00600
 GAC00610
 GAC00620
 GAC00630
 GAC00640
 GAC00650
 GAC00660
 GAC00670
 GAC00680
 GAC00690
 GAC00700
 GAC00710
 GAC00720
 GAC00730
 GAC00740
 GAC00750
 GAC00760
 GAC00770
 GAC00780
 GAC00790
 GAC00800
 GAC00810
 GAC00820
 GAC00830
 GAC00840
 GAC00850
 GAC00860
 GAC00870
 GAC00880
 GAC00890
 GAC00900
 GAC00910
 GAC00920
 GAC00930
 GAC00940
 GAC00950
 GAC00960
 GAC00970
 GAC00980
 GAC00990
 GAC01000

FILE: GBCT PLIOPT A1

VM/SP CONVERSATIONAL MONITOR SYSTEM

PAGE 001

```
GBCT: PROC(BSETID,BINFO,RTN_CODE);
/*A SUBROUTINE WHICH ACCEPTS THE BSETID AND RETURNS A DECODED BINFO
*** TO THE CALLER*****/
XINCLUDE DBCAT,FDEBUG,DBEU;
XINCLUDE ESUB1;
DCL P PTR. (NULL,ADDR) BUILTIN, RTN_CODE FIXED BIN;
DCL BSETID BIT (32);
DCL BIT_STRING BIT (88) BASED;

CALL RET1 (BSETID,P,RTN_CODE);
IF P=NULL THEN DO;
RTN_CODE=1; /*ILLEGA BSETID --1*****/
RETURN;
END;

ADDR(BINFO)->BIT_STRING-BEU.DATA;
FREE BEU;
RTN_CODE=0;
RETURN;
END;
```

GBCO0010
GBCO0020
GBCO0030
GBCO0040
GBCO0050
GBCO0060
GBCO0070
GBCO0080
GBCO0090
GBCO0100
GBCO0110
GBCO0120
GBCO0130
GBCO0140
GBCO0150
GBCO0160
GBCO0170
GBCO0180
GBCO0190
GBCO0200
GBCO0210

FILE: GECT PLIOPT A1 VM/SP CONVERSATIONAL MONITOR SYSTEM

```

GECT :PROC (ENAME, EINFO, RTN_CODE);
/***** A S-PROC IN THE ENTITY LEVEL TO OBTAIN THE ENTITY CAT. ENTRY
***** OF ENTITY SET WHOSE NAME IS SPECIFIED IN ENAME; RETURNS EINFO*/
XINCLUDE DECAT, ARETN, DKEY, SERVICE, FDEBUG;

DCL RTN_CODE FIXED BIN, ENAME CHAR (8);
DCL (P_TP, RP, LP) PIR, I FIXED BIN, NULL BUILTIN;
DCL UNSPEC BUILTIN;

/*BEGIN PROCESSING*/
RTN_CODE=0; /*INIT TO O.K.*/
/*ALLOCATE ROOT RETN TREE - ESET*/
I=0; /*NO PREDICATE*/
ALLOCATE RETN_ARG; LP, RP=P;
RETN_ARG.PSETID=KEY(KY.PESET);
RETN_ARG.GET=RETNGET.ANY;
RETN_ARG.N=0; /*O PREDICATE*/

/*NEXT NODE -- ENAME*/
I=1; /*NO PREDICATE*/
ALLOCATE RETN_ARG;
LP->RETN_ARG.PTR=P; LP=P;
RETN_ARG.PSETID=KEY(KY.PENAME);
RETN_ARG.GET=RETNGET.NO;
RETN_ARG.NODE=2;
RETN_ARG.BSETID=KEY(KY.BENAME);
RETN_ARG.PARENT=1;
RETN_ARG.CN(1)=1;
RETN_ARG.OP(1)=RETNOP.EQ;
RETN_ARG.DLEN(1)=8;
RETN_ARG.DATA(1)=UNSPEC (ENAME);

/*NEXT NODE -- EINFO */
I=0; /*NO PREDICATE*/
ALLOCATE RETN_ARG;
LP->RETN_ARG.PTR=P; LP=P;
RETN_ARG.PSETID=KEY(KY.PEINFO);
RETN_ARG.GET=RETNGET.ANY;
RETN_ARG.NODE=3;
RETN_ARG.BSETID=KEY(KY.BEINFO);
RETN_ARG.PARENT=1;
RETN_ARG.N=0; /*O PREDICATE*/

/*CALL UP RETN*/
CALL TCALL ('RETN', 1, RP, TP);

/*CHECK RETURNS*/
IF TP->RETN_RTN_CODE^=0 THEN DO;
  RTN_CODE=1; /*PROBLEM IN RETRIEVAL*/
  CALL SVC3 (RP);
  CALL SVC3 (TP);
  RETURN;

```

FILE: GECT PLIOPT A1 VM/SP CONVERSATIONAL MONITOR SYSTEM

```

END;

IF TP->RETN_RTN.N^=1 THEN DO;
  RTN_CODE=2; /*NO SUCH ENTITY*/
/* IF FDEBUG.LEV1 THEN PUT SKIP LIST ('GECT: NO SUCH ENTITY'); */
  CALL SVC3 (RP);
  CALL SVC3 (TP);
  RETURN;
END;

/*GET DATA AND DECODE*/
P=TP->RETN_RTN.PTR->RETN_RTN1.PTR; /*THE 3RD TOKEN RETURNED*/
EINFO=P->RETN_RTN1.DATA;
CALL SVC3 (TP);
CALL SVC3 (RP);
RETURN;
END /*GECT*/;

GEC00560
GEC00570
GEC00580
GEC00590
GEC00600
GEC00610
GEC00620
GEC00630
GEC00640
GEC00650
GEC00660
GEC00670
GEC00680
GEC00690
GEC00700
GEC00710
GEC00720

```

VM/SP CONVERSATIONAL MONITOR SYSTEM

```

GMEM: PROC(NUM_BYTES, ID);

/* THIS IS A SUBROUTINE AT THE MM LEVEL WHICH RESPOND TO A 'GET MAIN'
   REQUEST BY LOOKING UP THE NEXT FREE AREA AND ASSIGN AN ID TO IT */

DCL NUM_BYTES FIXED BIN, ID BIT (32), DATA CHAR(8), DT4 CHAR(4);
DCL (AVAIL_LOC, I) FIXED BIN(31), (UNSPEC, SUBSTR) BUILDIN;
XINCLUDE EPKAT;
CALL PAKT (O, DATA, 'R'); /* GET FIRST PACKET */
UNSPEC(AVAIL_LOC) = UNSPEC(SUBSTR(DATA, 1, 4)); /* OBTAIN AVAIL_LOC */
ID = UNSPEC(AVAIL_LOC); /* ASSIGN ID */
I = (NUM_BYTES - 1) / 8 + 1; /* FORCE TO BE MULTIPLE OF 8 */
AVAIL_LOC = AVAIL_LOC + I * 8; /* UPDATE IT */
UNSPEC(DT4) = UNSPEC(AVAIL_LOC);
SUBSTR(DATA, 1, 4) = DT4;
CALL PAKT (O, DATA, 'W'); /* WRITE NEW AVAIL_LOC */
RETURN;
END;

```

FILE: GPCY PLIOPT A1 VM/SP CONVERSATIONAL MONITOR SYSTEM

```

GPCY: PROC(PSETID,PINFO,RTN_CODE);
/*A SUBROUTINE WHICH ACCEPTS THE PSETID AND RETURNS A DECODED PINFO
*** TO THE CALLER*****/
%INCLUDE DPCAT,FDEBUG,DBEU;
%INCLUDE ESUB1;
DCL P PTR, (NULL,ADDR) BUILTIN, RTN_CODE FIXED BIN;
DCL PSETID BIT (32);
DCL BIT_STRING BIT (144) BASED;

CALL RET1 (PSETID,P,RTN_CODE);
IF P=NULL THEN DO;
  RTN_CODE=1; /*ILLEGAL PSETID --1*****/
  RETURN;
END;

ADDR(PINFO)->BIT_STRING=BEU.DATA;
FREE BEU;
RTN_CODE=0;
RETURN;
END;

```

GPC00010
 GPC00020
 GPC00030
 GPC00040
 GPC00050
 GPC00060
 GPC00070
 GPC00080
 GPC00090
 GPC00100
 GPC00110
 GPC00120
 GPC00130
 GPC00140
 GPC00150
 GPC00160
 GPC00170
 GPC00180
 GPC00190
 GPC00200
 GPC00210

FILE: GSEU PLIOPT A1 VM/SP CONVERSATIONAL MONITOR SYSTEM

```

GSEU: PROC (ID,SEUPTR,RTN_CODE);
GSE00010
GSE00020
GSE00030
GSE00040
GSE00050
GSE00060
GSE00070
GSE00080
GSE00090
GSE00100
GSE00110
GSE00120
GSE00130
GSE00140
GSE00150
GSE00160
GSE00170
GSE00180
GSE00190
GSE00200
GSE00210
GSE00220
GSE00230
GSE00240
GSE00250
GSE00260
GSE00270
GSE00280
GSE00290
GSE00300
GSE00310
GSE00320
GSE00330
GSE00340
GSE00350
GSE00360
GSE00370
GSE00380
GSE00390

/** THIS IS A SUBROUTINE AT THE MM LEVEL WHICH RETURNS A PTR TO AN
*** SEU (SEUPTR) OR A RETURN CODE (RTN_CODE) GIVEN THIS SEU'S ID:
*** IT CALLS PAKT TO GET PACKETS FROM THE DSH *****/

XINCLUDE DSEU1.EPAKT;
DCL DATA CHAR(8), VA FIXED BIN(31), TP PTR, DT5 CHAR(5);
DCL (I,RTN_CODE,L1,L2) FIXED BIN, ID BIT (32);
DCL (NULL,ADDR,SUBSTR,UNSPEC,MIN) BUILTIN;

RTN_CODE = 0;
UNSPEC(VA) = ID;
CALL PAKT (VA,DATA,'R');
SEUPTR = ADDR (DATA);
IF SEU.INVAL ^= 'N' THEN DO;
    RTN_CODE = 1;
    RETURN;
END;

/* OBTAIN FIRST 8 BYTES OF THIS SEU */
/* IMPOSE SEU STRUCTURE ON DATA */
/* IF ATTEMPT TO RETRIEVE INVAL SEU */

I=SEU.N;
ALLOC SEU SET (TP);
TP->SEU.INVAL = SEU.INVAL;
SEUPTR = TP;
L = MIN (5,SEU.N);
DATA = SUBSTR(DATA,4,L);
DT5 = SUBSTR(DATA,1,L);
SUBSTR(SEU.DATA,1,L) = DT5;

/* ESTABLISH FIRST 8 BYTES OF SEU */

L = (SEU.N + 2)/8;
DO I = 1 TO L;
    CALL PAKT (VA+I*8,DATA,'R');
    L1 = I*8-2; L2 = MIN(8,SEU.N-L1+1);
    DATA = SUBSTR (DATA,1,L2);
    SUBSTR (SEU.DATA,L1,L2)= DATA;
END;

RETURN;
END;

/* ESTABLISH THE REST OF SEU */
/* L = # OF PACKETS IN THIS SEU - 1 */
/* GET ALL PACKETS

```

```

LEX: PROC (1, TOKEN, MODE, RTN_CODE);
//*****
*
*      MODULE      DESCRIPTION
*      *****
*      *****/
//*****
PURPOSE: THIS IS THE LEXICAL ANALYZER WHICH PARSES AN INPUT
LINE INTO TOKENS. TOKEN DELIMITERS ARE ',' AND ';;'.
DELIMITERS IN QUOTES ARE NOT TREATED AS EXCEPTIONS.
MORE THAN 1 LINE OF INPUT CAN BE STRING TOGETHER BY
ENDING PREVIOUS LINES IN BACK SLASH.
BLANKS CAN BE OPTIONALLY SUPPRESSED.  BLANKS ENCLOSED IN
QUOTES ARE NOT SUPPRESSED.
*****
//*****
INPUT PARAMETERS:
MODE: IF SET INDICATES THAT BLANKS ARE TO BE SUPPRESSED.
*****
//*****
OUTPUT PARAMETERS:
I: NUMBER OF TOKENS
TOKEN: AN ARRAY CONTAINING THE TOKENS (CHAR$ UP
      TO 40 CHAR'S) AND LENGTH OF TOKENS
RTN_CODE: 0 - O.K.
          2 - UNBALANCED QUOTES
          1 - EXCESSIVE SEPARATORS
*****
//*****
/* LEXICAL ANALYZER VARIABLES */
DCL I TOKEN (25),
    2 D CHAR(40),
    2 L FIXED BIN;
DCL RTN_CODE FIXED BIN;
DCL I FIXED BIN,
    LAST FIXED BIN(15), LAST1 FIXED BIN;
DCL MODE FIXED BIN; /*1-SUPPRESS ALL BLANKS*/
*****
/* INPUT STRING VARIABLES */
DCL (STR1,STR2,STR) CHAR(240) VARYING,
    LINE CHAR(80),
    POS FIXED BIN(15);
DCL (VERIFY,SUBSTR,INDEX,LENGTH) BUILTIN;
*****
/* START SUBROUTINE */
RTN_CODE=0;
*****
/* GET INPUT STRING */
STR = '';
LOOP: GET EDIT (LINE) (A(80));
POS = INDEX(LINE,';');
IF POS ^= 0
THEN DO;
    STR = STR || SUBSTR(LINE,1,POS-1);
    GO TO LOOP;
END;
STR = STR || LINE;
*****
LEX00010
LEX00020
LEX00030
LEX00040
LEX00050
LEX00060
LEX00070
LEX00080
LEX00090
LEX00100
LEX00110
LEX00120
LEX00130
LEX00140
LEX00150
LEX00160
LEX00170
LEX00180
LEX00190
LEX00200
LEX00210
LEX00220
LEX00230
LEX00240
LEX00250
LEX00260
LEX00270
LEX00280
LEX00290
LEX00300
LEX00310
LEX00320
LEX00330
LEX00340
LEX00350
LEX00360
LEX00370
LEX00380
LEX00390
LEX00400
LEX00410
LEX00420
LEX00430
LEX00440
LEX00450
LEX00460
LEX00470
LEX00480
LEX00490
LEX00500
LEX00510
LEX00520
LEX00530
LEX00540
LEX00550

```

```

/* FIX UP LINE */
DO I = LENGTH(STR) TO 1 BY -1 WHILE(SUBSTR(STR,I,1) = ' ');
END;
STR = SUBSTR(STR,1,1) || ' ';

/* SUPPRESS BLANKS */
IF MODE=1 THEN DO;
  DO I=1 TO LENGTH(STR);
    /* SKIP BLANKS IN QUOTES */
    IF SUBSTR(STR,I,1)=''' THEN DO WHILE (''B);
      I=I+1;
    IF I>LENGTH(STR) THEN DO;
      RTN_CODE=2; /* ILLEGAL SYNTAX */
      RETURN;
    END;
    IF SUBSTR(STR,I,1)=''' THEN LEAVE;
  END;
ELSE
  IF SUBSTR(STR,I,1)=' ' THEN DO;
    STR1=SUBSTR(STR,1,I-1);
    STR2=SUBSTR(STR,I+1);
    STR=STR1||STR2;
    I=I-1; /* FIX UP I */
  END;
END;

/* LEX STARTS */
I = 0;
DO WHILE(VERIFY(STR,'') ^ 0); /* DO TILL END OF STRING */
  LAST = INDEX(STR,'') - 1; /* LOOKING FOR DELIMITER */
  LAST1 = INDEX(STR,'') - 1;
  IF LAST1 > -1 & LAST1 < LAST THEN LAST = LAST1;
  ELSE IF LAST < 0 THEN LAST = LAST1;

  IF LAST = 0
  THEN DO; /* MISUSE OF DELIMITER */
    RTN_CODE = 1;
    RETURN;
  END;

  I = I + 1;
  TOKEN.D(I) = SUBSTR(STR,1,LAST);
  TOKEN.L(I) = LAST;
  STR = SUBSTR(STR,LAST+2);
END;
END /* LEX */;

```

LEX00560
 LEX00570
 LEX00580
 LEX00590
 LEX00600
 LEX00610
 LEX00620
 LEX00630
 LEX00640
 LEX00650
 LEX00660
 LEX00670
 LEX00680
 LEX00690
 LEX00700
 LEX00710
 LEX00720
 LEX00730
 LEX00740
 LEX00750
 LEX00760
 LEX00770
 LEX00780
 LEX00790
 LEX00800
 LEX00810
 LEX00820
 LEX00830
 LEX00840
 LEX00850
 LEX00860
 LEX00870
 LEX00880
 LEX00890
 LEX00900
 LEX00910
 LEX00920
 LEX00930
 LEX00940
 LEX00950
 LEX00960
 LEX00970
 LEX00980
 LEX00990
 LEX01000
 LEX01010
 LEX01020
 LEX01030
 LEX01040

```

LEX00010
LEX00020
LEX00030
LEX00040
LEX00050
LEX00060
LEX00070
LEX00080
LEX00090
LEX00100
LEX00110
LEX00120
LEX00130
LEX00140
LEX00150
LEX00160
LEX00170
LEX00180
LEX00190
LEX00200
LEX00210
LEX00220
LEX00230
LEX00240
LEX00250
LEX00260
LEX00270
LEX00280
LEX00290
LEX00300
LEX00310
LEX00320
LEX00330
LEX00340
LEX00350
LEX00360
LEX00370
LEX00380
LEX00390
LEX00400
LEX00410
LEX00420
LEX00430
LEX00440
LEX00450
LEX00460
LEX00470
LEX00480
LEX00490
LEX00500
LEX00510
LEX00520
LEX00530
LEX00540
LEX00550

LEXT: PROC(N,TOKEN,T1,TN,PRED,MOP,RTN_CODE);
/* ***** */
**          MODULE      DESCRIPTION
**          *****
**          *****/
***** PURPOSE:
***** S-PROC AT THE USER INTERFACE LEVEL WHICH ACCEPTS TOKENS
***** IN A MATRIX AND IDENTIFIES THE TREE STRUCTURE AND THE PREDICATE
***** VALUES IN THESE TOKENS; IT ALSO DISCERNES MODIFICATION OPERATORS
***** FROM REGULAR ATTRIBUTE NAMES AND STORE THEM IN STRUCTURE MOP */
*****
***** METHOD:
***** THE HIERARCHICAL STRUCTURE OF NAMES IS IDENTIFIED BY
***** NESTED PARANTHESES. FOR EXAMPLE, IF TOKEN(1) IS 'ABC(',
***** TOKEN(2) IS 'DEF' AND TOKEN(3) IS 'GHI)', THEN DEF AND GHI
***** HAVE ABC AS THEIR PARENT NAME.
***** THERE ARE TWO TYPES OF NON-NAME TOKENS: MODIFICATION OPERATOR
***** (MOP) AND PREDICATE TOKEN. MOP IS DISTINGUISHED FROM REGULAR
***** NAME TOKENS BY ITS SPECIAL STARTING CHAR '-'. TOKEN THAT
***** CONTAIN SPECIAL COPARISON OPERATORS ('=', '<', '>') IS TAKEN
***** AS PREDICATE TOKEN AND IS BROKEN INTO A NAME TOKEN AND A
***** VALUE TOKEN.
*****
***** INPUT PARAMETERS:
***** N: NUMBER OF TOKENS IN THE TOKEN ARRAY
***** TOKEN: TOKEN ARRAY
*****
***** OUTPUT PARAMETERS:
***** T1: AN ARRAY DESCRIBING THE HIERARCHICAL STRUCTURE OF
***** NAME TOKENS. ROOT NAMES HAVE PARENT = 0 WHILE
***** NON-ROOT NAMES HAVE A PARENT EQUAL TO THE INDEX
***** OF THE PARENT IN T1.
***** TN: NUMBER OF NAMES IN T1.
***** PRED: AN ARRAY USED TO STORE PREDICATE OP, VALUE AND VALUE
***** LENGTH IDENTIFIED FOR CORRESPONDING NAME IN T1.
***** MOP: AN ARRAY USED TO OMDOCATE MODIFICATION OP IDENTIFIED
***** FOR CORRESPONDING NAME IN T1.
***** RTN_CODE: 0 - O.K.
*****           1,2: MISMATCH OF PARANTHESES
*****           3: ILLEGAL MODIFICATION OPERATOR
*****
DCL (N,TN,RTN_CODE,I,J,K)FIXED BIN;

DCL I T1 (25), /* ENTITY AND ATTRIBUTE NAMES TREE STRUCTURE */
    2 PARENT FIXED BIN,
    2 ANAME CHAR(8);

DCL I PRED (25), /* PRED VALUE STRUCTURE */
    2 OP CHAR (1),
    2 VALUE CHAR (40),
    2 VLEN FIXED BIN;
```

```

DCL MOP (25) CHAR (1). CURRENT_MOP_CHAR (1) INIT ('X');
/* INIT TO 'DON'T KNOW' */

DCL 1 TOKEN (25).
  2 D CHAR(40).
  2 L FIXED BIN;
DCL ST(25) FIXED BIN, STP FIXED BIN;
DCL (STR,DATA) CHAR(40) VARYING;
DCL (SUBSTR,INDEX,LENGTH) BUILTIN;

/*BEGIN PROCESSING*/

RTN_CODE=0;
TN=0; /*INIT T1 COUNT TO 0*/
STP=1; ST(STP)=0; /*INIT STACK*/

DO I=1 TO N; /*N IS NUMBER OF TOKEN*/
  STR=SUBSTR(TOKEN.D(I),1,TOKEN.L(I)); /* EXTRACT TOKEN */
  LOOP: /* FOR EACH TOKEN PERFORM THIS LOOP */

  K=INDEX(STR,'(');
  IF K=0
  THEN DO;
    CALL TIP(K); /* PROCESS UP TO '(' */
    IF RTN_CODE ^= 0 THEN RETURN; /* SET BY TIP */
    STP=STP+1; /*PUSH STACK*/
    ST(STP)=TN;
    END /* '(' */;
  ELSE DO;
    K=INDEX(STR,')');
    IF K=0
    THEN DO;
      CALL TIP(K); /* PROCESS UP TO ')' */
      IF RTN_CODE ^= 0 THEN RETURN;
      STP=STP-1;
      IF STP=0 THEN DO;
        RTN_CODE=1;
        RETURN;
        END /* RTN_CODE */;
      END;
    ELSE CALL TIP(K); /* PROCESS ENTIRE STRING */
    IF RTN_CODE ^= 0 THEN RETURN;
    END;

    IF STR^='' THEN GO TO LOOP; /* RECYCLE IF STR NOT EMPTY */
  END;

  IF STP=1 THEN RTN_CODE=2; /* MISMATCH OF PARENTH */
  RETURN;
  /****** */

```

```

TIP: PROC(K); /*INTERNAL SUBROUTINE*/
DCL K FIXED BIN;
DCL OP(3) CHAR (1) INIT ('>','<','=');
DCL VALUE CHAR (40) VARYING; DCL CMD CHAR (8);
DCL END FIXED BIN;

IF K=1
THEN DO; /*NO DATA*/
STR=SUBSTR(STR,K+1); RETURN;
END;

IF K>1 THEN DO;
DATA=SUBSTR(STR,1,K-1);
STR=SUBSTR(STR,K+1);
END;

IF K=0 THEN DO;
DATA=STR;
STR='';
END;

/* SEE IF TOKEN IS MODIFICATION OPERATORS */
K=INDEX(DATA, '-');
IF K = 1 THEN DO ; /* IT IS A MOP */
DATA = SUBSTR (DATA,2);
CMD = DATA;

SELECT (CMD);
WHEN ('ID') CURRENT MOP='I';
WHEN ('INSERT','INS','IST') CURRENT MOP='N';
WHEN ('REPLACE','REP','R') CURRENT MOP='R';
WHEN ('DELETE','DEL','D') CURRENT MOP='D';
OTHERWISE RTN CODE = 3; /* ILLEGAL SYNTAX */
END /* SELECT */;

RETURN; /* FINISHED THIS TOKEN */
END /* MOP */;

/* PROCESS DATA - NOT A MOP */
TN=TN+1;
DO J=1 TO 3; /* CHECK FOR VALUE */
K=INDEX(DATA,OP(J));
IF K=0 THEN LEAVE; /* FOUND */
END;

IF K=0 THEN DO ;/* OP FOUND, PROCESS VALUE BY BREAKING TOKEN INTO
TWO: VALUE AND NAME DATA */
PRED.OP(TN)=SUBSTR(DATA,K,1);
VALUE=SUBSTR(DATA,K+1);
DATA=SUBSTR(DATA,1,K-1);

IF SUBSTR(VALUE,1,1)=''' THEN DO; /*TAKE QUOTES OUT */
END=LENGTH(VALUE)-2;
VALUE=SUBSTR(VALUE,2,END);
END;

```

205

VM/SP CONVERSATIONAL MONITOR SYSTEM

FILE: LEXT PLIOPT A1

```

END;
ELSE VALUE='';
PRED.VALUE(TN)=VALUE;
PRED.VLEN(TN)=LENGTH(VALUE);
/* REGISTER NAME*/
T1.ANAME(TN)=DATA;
T1.PARENT(TN)=ST(STP);
MOP(TN) = CURRENT_MOP;
RETURN;
END /*T1P*/;
/*****
END /*LEXT*/;
LEXO1660
LEXO1670
LEXO1680
LEXO1690
LEXO1700
LEXO1710
LEXO1720
LEXO1730
LEXO1740
LEXO1750
LEXO1760
LEXO1770
LEXO1780
LEXO1790
LEXO1800
LEXO1810
LEXO1820

```

```

MIN00010
MIN00020
MIN00030
MIN00040
MIN00050
MIN00060
MIN00070
MIN00080
MIN00090
MIN00100
MIN00110
MIN00120
MIN00130
MIN00140
MIN00150
MIN00160
MIN00170
MIN00180
MIN00190
MIN00200
MIN00210
MIN00220
MIN00230
MIN00240
MIN00250
MIN00260
MIN00270
MIN00280
MIN00290
MIN00300
MIN00310
MIN00320
MIN00330
MIN00340
MIN00350
MIN00360
MIN00370
MIN00380
MIN00390
MIN00400
MIN00410
MIN00420
MIN00430
MIN00440
MIN00450
MIN00460
MIN00470
MIN00480
MIN00490
MIN00500
MIN00510
MIN00520
MIN00530
MIN00540
MIN00550

INIT: PROC;
/.....
MODULE DESCRIPTION
*****
PURPOSE:
** A T-PROC AT THE MM LEVEL WHICH INITIALIZES THIS LEVEL AND OBTAINS
** THE KEY FOR THE UPPER LEVEL IF DATA ALREADY EXISTS IN SH.
** (FOR INTEGRATED VERSION: MAKE USE OF SUBROUTINE PAKT)
/.....
METHOD:
1. IF IT IS INIT REQUEST AND IT IS NEW INIT (I.E., SH DOES*
NOT CONTAIN ANY DATA YET), THEN AVAIL LOC WHICH INDICATES*
THE STARTING LOCATION OF FREE VIRTUAL MEMORY AND IT IS STORED AT THE FIRST PACKET OF SH IS INITIALIZED TO BE 16 BY CALLING SUBROUTINE PAKT.
2. IF IT IS INIT REQUEST AND IT IS FILE INIT (I.E., SH ALREADY EXISTS WITH DATA) THE KEY FOR THE UPPER LEVEL (4 BYTES) IS READ IN FROM VIRTUAL MEMORY LOCATION 5 AND THEN GIVEN TO THE UPPER LEVEL
3. IF IT IS STORE KEY REQUEST THEN THE KEY IS STORED AT VM LOC 5.
/.....
INPUT PARAMETERS: AS INDICATED BY AMINIT.
/.....
XINCLUDE DSEU1,EPAKT,SERVICE, FDEBUG,AMINI;
DCL KEY BIT (32);
DCL (P,RTP) PTR. (NAME,DATA) CHAR (8), OP CHAR (1), DT4 CHAR(4);
DCL (NULL,BIN,UNSPEC,SUBSTR) BUILTIN;

CALL TBEG ( PROC_ADDR, P);
ALLOCATE MINIT_RTN SET (RTP);
OP=MINIT_ARG.OP;
NAME=MINIT_ARG.FNAME;

SELECT (OP);
WHEN ('I') DO; /*INIT*/

IF NAME=(8)' ' THEN DO; /*NEW INIT*/
/*SET STARTING LOCATION OF AVAILABLE VIRTUAL MEMORY AT 16: 4 FOR KEY, 4 FOR AVAIL LOC, 8 FOR EXPANSION */
UNSPEC(DT4) = UNSPEC(BIN(16,31));
SUBSTR(DATA,1,4) = DT4;
CALL PAKT (O.DATA,'W');
GO TO RTN.
END;

ELSE DO;
/* FILE INIT */
CALL PAKT (O.DATA,'R'); /* GET FIRST PACKET IN THE SH */
DT4 = SUBSTR(DATA,5,4); /* GET THE 2ND HALF OF THE PACKET */

```

```
KEY = UNSPEC (DT4); /* GET KEY */
RIP->MINIT_RTN.KEY=KEY;
GO-TO-RTN
END;

END /*WHEN I*/;

WHEN ('S') DO: /*PROCESS STORE KEY REQUEST*/
CALL PAKT (O.DATA,'R'); /* GET 1ST PACKET IN THE SH */
UNSPEC(DT4) = MINIT_ARG.KEY;
SUBSTR(DATA,5,4) = DT4; /* UPDATE IT WITH KEY */
CALL PAKT (O.DATA,'W');
END /*WHEN S*/;

END /*SELECT */;
RTN: CALL TRTN( PROC_ADDR,RTP);

RETURN;
END /*MINIT*/;
```

```
MIN00560
MIN00570
MIN00580
MIN00590
MIN00600
MIN00610
MIN00620
MIN00630
MIN00640
MIN00650
MIN00660
MIN00670
MIN00680
MIN00690
MIN00700
MIN00710
MIN00720
MIN00730
MIN00740
```

FILE: MODN PLIOP T A1 VM/SP CONVERSATIONAL MONITOR SYSTEM

```

MODN: PROC(MP,OFF_NODE,RTN_CODE);
/*A S-PROC AT THE N-ARY LEVEL USED AS A SUBROUTINE CALLED BY THE
*UPDN PROGRAM TO PERFORM MOD OP AT ROOT NODE*/
XINCLUDE AUPDN;
XINCLUDE ECRTB,EDELB,EREPB,ESRCH;
DCL (MP,P) PTR, (N,RTN_CODE) FIXED BIN,
OFF_NODE(25) BIT (1) CONNECTED,
DATA BIT (320) VARYING, (PS,ID,BS) BIT (32), (SUBSTR,NULL)BUILTIN;
/*DO P=MP REPEAT UPDN_ARG.PTR WHILE(P^=NULL);
PUT SKIP LIST('MODN ARG LIST',UPDN_ARG); END: */

P=MP;
/*LOCATE ROOT NODE ID*/
PS=UPDN_ARG.PSETID;
DATA=SUBSTR(UPDN_ARG.DATA,1,UPDN_ARG.DLEN * 8);
CALL SRCH(PS,DATA,ID,RTN_CODE);

/*FOLLOW THE TREE*/
DO P=UPDN_ARG.PTR REPEAT UPDN_ARG.PTR WHILE (P^=NULL);
IF OFF_NODE(UPDN_ARG.NODE)='O'B THEN DO;
BS=UPDN_ARG.BSETID;
DATA=SUBSTR(UPDN_ARG.DATA,1,UPDN_ARG.DLEN * 8);
SELECT (UPDN_ARG.OP);

WHEN (UPDNOP.IST) CALL CRTB(PS,BS,ID,DATA,'A',RTN_CODE);
WHEN(UPDNOP.CI) CALL CRTB (PS,BS,ID,DATA,'N',RTN_CODE);
WHEN (UPDNOP.DEL) CALL DELB (PS,BS,ID,'A',RTN_CODE);
WHEN (UPDNOP.DD) CALL DELB (PS,BS,ID,'N',RTN_CODE);
WHEN (UPDNOP.REP)
CALL REPB (PS,BS,ID,DATA,'A',RTN_CODE);
WHEN (UPDNOP.REPP)
CALL REPB (PS,BS,ID,DATA,'N',RTN_CODE);

END /*SELECT*/;
END /*DO IF*/;
END /*DO*/;
END /*MODN*/; /*RTN_CODE IS SET TO RTN FROM SUBROUTINE CALLS*/

```

VM/SP CONVERSATIONAL MONITOR SYSTEM

FILE: MPST PLI0PT A1

```

MPST: PROC(MAP,POS);
/*S-PROC AT THE N-ARY LEVEL USED BY DEFP AND DEFB FOR ASSIGNING
*ID SLOTS IN THE ID ARRAY****/
DCL MAP BIT (32), (POS.1) FIXED BIN;
DCL SUBSTR BUILTIN;
DO I=32 TO 1 BY -1;
  IF SUBSTR(MAP,I,1)='O'B THEN DO;
    SUBSTR(MAP,I,1)='1'B;
    POS=I;
    RETURN;
  END;
END;
POS=0; /*NOT AVAILABLE*/
RETURN;
END /*MPST*/;
MP500010
MP500020
MP500030
MP500040
MP500050
MP500060
MP500070
MP500080
MP500090
MP500100
MP500110
MP500120
MP500130
MP500140
MP500150
MP500160
MP500170
MP500180

```

```

NINIT: PROC;
/* A T-PROC AT THE N-ARY LEVEL FOR INITIATING THE CATALOGUE STRUC
** AT THE N-ARY LEVEL *****/
XINCLUDE ANINI.SERVICE,FDEBUG.AMINI,DNKEY,DPCAT,DBEU,ADEFP;
XINCLUDE ESUB1.EMPST;
DCL (P,RP,TP,IP1,MP,RTP) PTR,(NULL,STRING,UNSPEC,ADDR) BUILTIN;
DCL AKEY BIT (32) STATIC , NKEY(3) BIT (32);
DCL MAP BIT (32), POS FIXED BIN, (I,J) FIXED BIN, ID BIT (32);
DCL RTN CODE FIXED BIN;
DCL BIT_STRING BIT (144) BASED;
/*BEGIN_PROCESSING*/
CALL TBEG (PROC ADDR, P); MP=P;
ALLOCATE NINIT_RTN SET (RTP);
RTP->NINIT_RTN_CODE=0;
SELECT (NINIT_ARG.OP);

WHEN ('I') DO: /*'I' TYPE NINIT COMMAND*****/
  ALLOCATE NINIT_ARG SET (RP);
  RP->NINIT_ARG.OP='I';
  RP->NINIT_ARG.FNAME=NINIT_ARG.FNAME;
  CALL TCALL ('NINIT',1,RP,P);
  CALL SVC3(RP);

/* PROCESS INIT AT THIS LEVEL****/
IF MP->NINIT_ARG.FNAME~(8), THEN DO: /*FILE INIT**/
  AKEY=NINIT_RTN.KEY; /*OBTAIN KEY FROM NINIT RETURN****/
  CALL SVC3 (P);
  CALL RET1(AKEY,P,RTN_CODE);/*CONTENT OF ANCHOR KEY*/
  STRING (NKEY)=BEU.DATA;
  FREE BEU;
  PCATID=NKEY(1); /*OBTAIN PCATID*****/
  BCATID=NKEY(2); /*OBTAIN BCATID*****/
  RTP->NINIT_RTN.KEY=NKEY(3); /*ALSO RETURN KEY TO UPPER LVL*/
  CALL TRTN(PROC_ADDR, RTP); /*RETURNS*/
  RETURN;
END; /*FILE INIT*/

IF MP->NINIT_ARG.FNAME=(8), THEN DO: /*NEW INIT*/
  /*INIT PCAT*/
  CALL SVC3(P);
  PINF0.NUMPTR=4;
  PINF0.PLEN=18; /*18 BYTES LONG*/
  PINF0.PTYPE='B';
  PINF0.LTYPE=PINF0.LTYPE; /*LINK LIST*/
  PINF0.L.ID=UNSPEC (NULL); /*INIT TO NULL*/
  /*CALL UP MPST TO OBTAIN L_POS'S*/
  MAP='00001111111111111111111111111111'B;
  CALL MPST (MAP,POS);
  PINF0.L.POS=POS;
  CALL MPST(MAP,POS);
  PINF0.L.POS2=POS;
  PINF0.MAP=MAP;

```

```

/*STREAMLINE PINFO INTO BIT STRING*/
/*CREATE THIS PCAT CAT ENTRY***/
I=4; /*4 IDS*/
J=18*8;
ALLOCATE BEU;
BEU.DATA=ADDR(PINFO)->BIT_STRING;
CALL CRT1 (ID.P, RTN_CODE);
PCATID=ID;
FREE BEU;

/*INITIALIZE BCAT AS A PSET*/
ALLOCATE DEFP ARG SET (TP);
TP->DEFP_ARG.PTYPE='B';
TP->DEFP_ARG.PLEN=11;
TP->DEFP_ARG.IMP.HOW=DEFPHOW.SA;
TP->DEFP_ARG.IMP.NUMPTR=4;

CALL TCALL ('DEFP',1, TP.TP1);
BCATID=TP1->DEFP_RTN.PSETID;
CALL SVC3 (TP);
CALL SVC3 (TP1);

NKEY(1)=PCATID;
NKEY(2)=BCATID;
NKEY(3)=UNSPEC (NULL); /*RESERVED*/
I=1; J=96; ALLOCATE BEU;
BEU.DATA=STRING(NKEY);
CALL CRT1(AKEY,P.RTN_CODE); /*OBTAINS ANCHOR KEY(AKEY)*/
FREE BEU;

/*STORE KEY BY CALLING MINIT*/
ALLOCATE MINIT ARG;
MINIT_ARG.OP='S'; MINIT_ARG.KEY=AKEY;
CALL TCALL ('MINIT',1,P.TP);

/*RETURNS**/
CALL SVC3 (P);
CALL SVC3 (TP);
CALL TRTN (PROC_ADDR, RTP);
RETURN;
END /*NEW INIT*/;

END /*WHEN 'I'*/;
WHEN ('S') DO: /*'S' OP FOR MINIT*/
  NKEY(1)=PCATID;
  NKEY(2)=BCATID;
  NKEY(3)=MINIT_ARG.KEY;
  I=1; J=96; ALLOCATE BEU; BEU.DATA=STRING(NKEY);
  IF AKEY=UNSPEC (NULL) THEN
    CALL REPT(AKEY,P.RTN_CODE);
  ELSE CALL CRT1(AKEY,P.RTN_CODE);
  FREE BEU;
  CALL TRTN(PROC_ADDR, RTP);
END /*WHEN 'S'*/;

```

FILE: NINIT PLIOPT A1

VM/SP CONVERSATIONAL MONITOR SYSTEM

PAGE 003

END /*SELECT*/;

END /*NINIT*/;

NINO1110
NINO1120
NINO1130
NINO1140
NINO1150

FILE: PAKT PLIDPT A1 VM/SP CONVERSATIONAL MONITOR SYSTEM

```

PAK00010
PAK00020
PAK00030
PAK00040
PAK00050
PAK00060
PAK00070
PAK00080
PAK00090
PAK00100
PAK00110
PAK00120
PAK00130
PAK00140
PAK00150
PAK00160
PAK00170
PAK00180
PAK00190
PAK00200
PAK00210
PAK00220
PAK00230
PAK00240
PAK00250
PAK00260
PAK00270
PAK00280
PAK00290
PAK00300
PAK00310
PAK00320
PAK00330
PAK00340
PAK00350
PAK00360
PAK00370
PAK00380
PAK00390
PAK00400
PAK00410
PAK00420
PAK00430
PAK00440
PAK00450
PAK00460
PAK00470
PAK00480
PAK00490
PAK00500
PAK00510
PAK00520
PAK00530
PAK00540
PAK00550

PAKT PROC(VA,DATA,RW);
/* THIS IS A SUBROUTINE AT THE MM LEVEL WHICH RETRIEVES AN 8-BYTE
*** PACKET (DATA) GIVEN ITS VIRTUAL ADDRESS (VA) IF RW IS 'R', OR
*** WRITES AN 8-BYTE PACKET GIVEN ITS VA IF RW IS 'W'. ****
*** IT INTERFACES DIRECTLY WITH THE DSH STV ***** */
/* DUE TO INCOMPATIBILITY OF FORMAT OF 'ARG' TO THE SET OF SERVICE
*** ROUTINES COMMONLY USED BY OTHER FH MODULES, CONTROL STRUCTURE
*** IS DIRECTLY ACCESSED FROM PAKT, AND TIMING CODE IS DUPLICATED
*** HERE TO ACCUMULATE STATISTICS OF MODULE L1 WHICH IS NORMALLY
*** PERFORMED BY SERVICE ROUTINE TCALL ***** */
XINCLUDE USERS.VPX.PFSVC.ARG;
DCL VA FIXED BIN(31), DATA CHAR(8), RW CHAR(1);

J=8; ALLOC ARG;
ARG.VIRTUAL_ADDRESS = VA;
ARG.VPID = THISVP->VP.VPID;
ARG.BOXID = 1;
IF RW = 'R' THEN ARG.REQ_TYPE = 'READ';
ELSE DO;
  ARG.REQ_TYPE = 'WRITE';
  ARG.DATA = DATA;
END;

/* PREPARE ARG TO COMM. WITH DSH */
/* PREPARE FOR SEND PROTOCOL */
ALLOC PF_SVC;
PF_SVC.SVC = 'L1';
PF_SVC.PTR = ARGPTR;

/* ***** TIMING REPORT OPTION ***** */
DCL FTIME BIT(1) EXT; /* FTIME IS TIMING REPORT OPTION */
DCL (BEGTIME,ENUM,LOTIME,CIL) FIXED BIN(31);
DCL RTIMER ENTRY RETURNS(FIXED BIN(31));
DCL (E/CUNT(50),ELO(50),ETIME(50)) FIXED BIN(31) EXT INIT((50)0);
IF FTIME THEN DO;
  ALLOCATE LOTIME; LOTIME=0;
  BEGTIME=RTIMER; END;
/* ***** TIMING REPORT OPTION ***** */

CALL SEND (1,0,1,'S',20,PT_SVC);
CALL WAIT(1);

/* ***** TIMING REPORT OPTION ***** */
IF FTIME THEN DO;
  BEGTIME=RTIMER-BEGTIME;
  ENUM = 19;
  ETIME(ENUM)=ETIME(ENUM)+BEGTIME; /* L1 IS THE 19TH PROC NAME TO BE TRACED */
  ECOUNT(ENUM)=ECOUNT(ENUM)+1; /* ACCUMULATE COUNT */
  ELO(ENUM)=ELO(ENUM)+LOTIME; /* ACCUMULATE LOWER LEVEL PROC TIME */
  FREE LOTIME; IF ALLOCATION(LOTIME)~= 0 THEN LOTIME=LOTIME+BEGTIME;
END;
/* ***** TIMING REPORT OPTION ***** */
ARGPTR = VP.WAIT.MSG; /* OBTAIN RETURN ARG */
IF RW = 'R' THEN DATA = ARG.DATA;
FREE ARG;

```

FILE: PAKT

PLIOP T A 1

VM/SP CONVERSATIONAL MONITOR SYSTEM

PAGE 002

RETURN;
END;

PAK00560
PAK00570

FILE: PRNF PLIOPT A1 VM/SP CONVERSATIONAL MONITOR SYSTEM

```

PRNT: PROC (P1,P2,RTN_CODE);
/* S-PROC AT THE USER INTERFACE LEVEL, CALLED BY DMO. THIS MODULE WILL
  FORMAT OUTPUT OF A RETE CALL. IT IS PASSED A POINTER POINTING TO THE
  RETE_ARG CHAIN AND A POINTER POINTING TO THE RETE_RTN CHAIN */
XINCLUDE ARETE;
DCL 1 T(25). /* TEMP STRUCTURE */
  2 NAME CHAR (8) INIT ((25) (8) ' '),
  2 PARENT FIXED BIN INIT((25)0),
  2 MLEN FIXED BIN,
  (2 POS, 2 LINE) FIXED BIN (31);

DCL HEADER(2) CHAR (80) VARYING;
DCL TOKEN CHAR (20) VARYING;
DCL STR (4) CHAR (80) VARYING;
DCL (I,J,K,L,CURRENT_PARENT, LAST.SUM, NN, I1, I2, I3) FIXED BIN (31);
DCL RTN_CODE FIXED BIN;
DCL (P1,P2,P) PTR. (NULL, SUBSTR, MAX) BUILTIN;
DCL NODE FIXED BIN;
DCL ST (10) FIXED BIN. STP FIXED BIN;

/* BEGIN */

PUT SKIP LIST ('ENTITY SET NAME', P1->RETE_ARG.NAME);

/*BUILD T STRUCTURE */
DO P=P1 REPEAT RETE_ARG.PTR WHILE (P^=NULL);
  NN=RETE_ARG.NODE; /* NN IS BOTH INDEX TO T AND NODE # */
  T.NAME(NN) = RETE_ARG.NAME;
  T.PARENT(NN) = RETE_ARG.PARENT;
END;

DO P=P2->RETE_RTN.PTR REPEAT RETE_RTN1.PTR WHILE (P^=NULL);
  NN=RETE_RTN1.NODE;
  T.MLEN(NN)=RETE_RTN1.DLEN; /*NOTE IN CURRENT VERSION, MLEN A
    OLEN OF A DATA ELEMENT IS THE SAME */
END;

/*HEADER */
HEADER(1)=''; HEADER(2) = ''; LAST=99; STP=1; ST(STP) =1; L=1;
DO J=2 TO 25;
  IF T.NAME(J) ^= (8) ' ' THEN DO;
    TOKEN = '';
    SELECT (T.PARENT(J));
      WHEN (LAST) DO;
        TOKEN = TOKEN || ' (' || T.NAME(J);
        STP = STP + 1;
        ST(STP) = T.PARENT(J);
      END;
    WHEN (ST(STP)) TOKEN = TOKEN || ' | ' || T.NAME(J);
    OTHERWISE DO;
      DO WHILE (ST(STP) ^= T.PARENT(J));
        STP=STP-1;
        TOKEN = TOKEN || ' ' ;
      END;
    END;
  END;

```

```

      TOKEN = TOKEN||' ' || T.NAME(J);
      END;
      END /* SELECT */;

      LAST=J;
      IF LENGTH(HEADER(L))+LENGTH(TOKEN) > 80
      THEN DO;
        L = L + 1;
        HEADER(L) = TOKEN;
      END;
      ELSE HEADER(L) = HEADER(L)||TOKEN;
      END /* IF */;
      END /*DO J */;
      DO WHILE (STP ^= 1); /* FIX UP LAST TOKEN */
        STP=STP-1;
        HEADER(L) = HEADER(L)||' ';
      END;
      HEADER(L) = HEADER(L)||' ';
      DO I = 1 TO L;
        PUT SKIP LIST (HEADER(I));
      END;
      PUT SKIP;

      /* DATA */
      SUM=3; L=1;
      DO J=2 TO 25;
        IF T.NAME(J)^(8)' ' THEN DO;
          T.POS(J)=SUM+1;
          SUM=SUM+T.MLEN(J)+3;
          IF SUM > 80 THEN DO;
            L=L+1;
            T.POS(J)=1;
            SUM=T.MLEN(J)+1;
          END;
          T.LINE(J)=L;
        END;
      END /*DO J */;

      /* PRINT DATA */
      DO I = 1 TO L; /* INIT STR */
        STR(I)=(100)' ';
      END;
      SUBSTR(STR(1),1,3)=' ' ' ';

      DO P=P2->RETE_RTN.PTR->RETE_RTN1.PTR REPEAT RETE_RTN1.PTR WHILE
        (P^=NULL);
      NODE=RETE_RTN1.NODE;
      IF NODE=1
      THEN DO;
        DO I = 1 TO L;
          PUT SKIP LIST (STR(I));
          STR(I)=(100)' ';
        END;
        SUBSTR(STR(1),1,3)=' ' ' ';
      END;

```

```

PRNO0560
PRNO0570
PRNO0580
PRNO0590
PRNO0600
PRNO0610
PRNO0620
PRNO0630
PRNO0640
PRNO0650
PRNO0660
PRNO0670
PRNO0680
PRNO0690
PRNO0700
PRNO0710
PRNO0720
PRNO0730
PRNO0740
PRNO0750
PRNO0760
PRNO0770
PRNO0780
PRNO0790
PRNO0800
PRNO0810
PRNO0820
PRNO0830
PRNO0840
PRNO0850
PRNO0860
PRNO0870
PRNO0880
PRNO0890
PRNO0900
PRNO0910
PRNO0920
PRNO0930
PRNO0940
PRNO0950
PRNO0960
PRNO0970
PRNO0980
PRNO0990
PRNO1000
PRNO1010
PRNO1020
PRNO1030
PRNO1040
PRNO1050
PRNO1060
PRNO1070
PRNO1080
PRNO1090
PRNO1100

```

FILE: PRNT PLIOPT A1 VM/SP CONVERSATIONAL MONITOR SYSTEM

```

ELSE IF RETE_RTN1.DLEN ^= 0
THEN DO:
  I1=T.LINE(NODE);
  I2=T.POS(NODE);
  I3=T.MLEN(NODE);
  SUBSTR(STR(I1),I2,I3)
  =SUBSTR(RETE_RTN1.CDATA,1,RETE_RTN1.DLEN);
  SUBSTR(STR(I1),I2+I3,3) = ' | ' ;
END;
END /* DO P */;
DO I= 1 TO L; /*WRITE OUT THE LAST OCCURRENCE */
  PUT SKIP LIST (STR(I));
END;
END /* PRINT */;
PRNO1110
PRNO1120
PRNO1130
PRNO1140
PRNO1150
PRNO1160
PRNO1170
PRNO1180
PRNO1190
PRNO1200
PRNO1210
PRNO1220
PRNO1230
PRNO1240
PRNO1250

```

FILE: PSEU PLLOPT A1 VM/SP CONVERSATIONAL MONITOR SYSTEM

```

PSEU: PROC (ID,SEUPTR,RTN_CODE);
  /... THIS IS A SUBROUTINE AT THE MM LEVEL WHICH PUTS AN SEU ACCORDING
  *** TO ITS PREASSIGNED ID INTO THE STORAGE AREA. IT DECOMPOSES AN
  *** SEU INTO 8-BYTE PACKTES AND CALLS PAKT TO WRITE THEM INTO DSH */
  %INCLUDE DSEU1,EPAKT;
  DCL DATA CHAR(8), VA FIXED BIN(31), (TP,BP) PTR;
  DCL (I,RTN_CODE) FIXED BIN, BASED_DATA CHAR (8) BASED (BP);
  DCL (NULL_ADDR,SUBSTR,MIN,UNSPEC) BUILTIN, (L,L1,L2) FIXED BIN(31);
  DCL ID BIT(32);

  RTN_CODE = 0;
  UNSPEC(VA) = ID;
  DATA = SEUPTR -> BASED_DATA;
  CALL PAKT (VA,DATA,'W');
  L = (SEU.N + 2)/8;
  DO I = 1 TO L;
    L1 = 1+8-2; L2 = MIN(8,SEU.N-L1+1);
    DATA = SUBSTR (SEU.DATA,L1,L2);
    CALL PAKT (VA+I*8,DATA,'W');
  END;

  RETURN;
END;

```

PSE00010
 PSE00020
 PSE00030
 PSE00040
 PSE00050
 PSE00060
 PSE00070
 PSE00080
 PSE00090
 PSE00100
 PSE00110
 PSE00120
 PSE00130
 PSE00140
 PSE00150
 PSE00160
 PSE00170
 PSE00180
 PSE00190
 PSE00200
 PSE00210
 PSE00220
 PSE00230
 PSE00240
 PSE00250
 PSE00260

FILE: REP PLOIPT A1 VM/SP CONVERSATIONAL MONITOR SYSTEM

```

REP:PROC;
/*T-PROC AT THE MM LEVEL TO REPLACE A SEU GIVEN ITS ID*****//
/* FOR INTEGRATED VERSION: MAKE USE OF SUBROUTINES GSEU & PSEU */

XINCLUDE SERVICE, AREP, DSEU1, EGSEU, EPSEU;
DCL (P,RTP)PTR;
DCL (I,RTN_CODE) FIXED BIN;
DCL (NULL, UNSPEC) BUILTIN;

/*BEGIN*/
CALL TBEG (PROC_ADDR, P);
ALLOCATE REP_RTN_SET (RTP);
RTP->REP_RTN_CODE = 0;

CALL GSEU (REP_ARG.ID, SEUPT, RTN_CODE); /*GET SEU DESIGNATED BY ID*/
IF RTN_CODE ^= 0
THEN RTP->REP_RTN_CODE = 1; /* ILLEGAL SEU */
ELSE DO;
  SEU.N=REP_ARG.N/8;
  UNSPEC(SEU.DATA) = REP_ARG.DATA;
  CALL PSEU (REP_ARG.ID, SEUPT, RTN_CODE);
END /*ELSE DO*/;

CALL TRTN(PROC_ADDR, RTP);
RETURN;
END /*REP*/;

```

```

REPO0010
REPO0020
REPO0030
REPO0040
REPO0050
REPO0060
REPO0070
REPO0080
REPO0090
REPO0100
REPO0110
REPO0120
REPO0130
REPO0140
REPO0150
REPO0160
REPO0170
REPO0180
REPO0190
REPO0200
REPO0210
REPO0220
REPO0230
REPO0240
REPO0250
REPO0260
REPO0270

```

FILE: REPB PLIOPT A1 VM/SP CONVERSATIONAL MONITOR SYSTEM

```

REPB: PROC (PARENT,BSETID, ID1,DATA,MODE,RTN_CODE);
  %INCLUDE DBCAT, DBEU;
  %INCLUDE EBCT,ESRCH,ECRTP,ESUB1;

  DCL (PARENT, BSETID, ID1,ID2) BIT (32), DATA BIT (*);
  DCL RTN_CODE FIXED BIN, MODE CHAR (1), P_PTR, NULL BUILTIN;

  CALL GBCT (BSETID, BINFO, RTN_CODE);
  CALL RET1 (ID1,P,RTN_CODE); /* GET SOURCE OUT */
  ID2 = BEU.ID_ARRAY (BINFO.POS); /* GET OLD TARGET */
  SELECT (MODE);

  WHEN ('N') DO; /* MODE MODE OF REPLACE OF BINARY ASSOC */
    IF BINFO.FUNC='S' THEN DO; /*REPLACE IT*/
      IF ID2 ^= UNSPEC (NULL) THEN
        CALL REPP (BINFO.PSETID(2), DATA, ID2, RTN_CODE);
      ELSE CALL CRTP (BINFO.PSETID(2), DATA, ID2, RTN_CODE);
    END /* IF 'S' */;

  ELSE DO; /* WHEN FUNC IS MULTIPLE SEE IF DATA ALREADY EXISTS */
    CALL SRCH(BINFO.PSETID(2),DATA,ID2,RTN_CODE);
    IF RTN_CODE=1 /*NOT FOUND*/
      THEN CALL CRTP(BINFO.PSETID(2),DATA,ID2,RTN_CODE);
    END;

  END /*WHEN N*/;

  WHEN ('A') DO; /*ARC MODE OF REPLACEMENT */
    CALL SRCH (BINFO.PSETID(2), DATA,ID2, RTN_CODE);
    IF RTN_CODE=1 THEN RETURN; /*RTNCODE 1 IS TARGET OF ARC NOT FND*/
    END /*WHEN 'A'*/;

  END /*SLECT*/;

  /*ENTER ID2 INTO ID1 ID ARRAY*/
  BEU.ID_ARRAY(BINFO.POS)=ID2;
  CALL REP1 (ID1,P,RTN_CODE);
  RTN_CODE=0;
  END /*REPB*/;

```

FILE: REP VM/SP CONVERSATIONAL MONITOR SYSTEM

FILE: REP PLIOPT AI

```

REPP: PROC (PSETID, DATA, ID, RTN_CODE);
/* A S-PROC AT THE N-ARY LEVEL USED TO REPLACE THE DATA OF A P-ELEMENT
   (DESIGNATED BY ID) IN A PSET (DESIGNATED BY PSETID) WITH NEW DATA
   (GIVEN AS DATA); IT IS AN ERROR IF ID IS NULL OR IS BAD */
%INCLUDE DBEU;
%INCLUDE ESUB1;
DCL (PSETID, ID) BIT (32), P PTR, RTN_CODE FIXED BIN;
DCL (NULL, UNSPEC) BUILTIN;
DCL DATA BIT (*);

CALL RET1 (ID, P, RTN_CODE); /* GET SOURCE */
IF P = NULL | RTN_CODE = 0 THEN DO; RTN_CODE = 1 + RTN_CODE;
RETURN; END; /*** 1+ IS BAD SOURCE */

BEU, DATA = DATA; /* REPLACE DATA */
CALL REP1 (ID, P, RTN_CODE);
END /*REPP */;

REPO0010
REPO0020
REPO0030
REPO0040
REPO0050
REPO0060
REPO0070
REPO0080
REPO0090
REPO0100
REPO0110
REPO0120
REPO0130
REPO0140
REPO0150
REPO0160
REPO0170
REPO0180
REPO0190

```

```
REP1:PROC (ID,P,RTN_CODE);
```

```
/*A S-PROC AT THE N-ARY LEVEL FOR COMMUNICATING DIRECTLY WITH THE
** REP MODULE AT THE MM LEVEL*/
```

```
%INCLUDE AREP.DBEU.SERVICE;
```

```
DCL ID BIT (32), (P,P1,BP) PTR (RTN_CODE,I,J,K) FIXED BIN;
```

```
DCL (NULL,UNSPEC,STRING,SUBSTR) BUILTIN;
```

```
DCL BIT_STRING BIT (800) BASED;
```

```
/*FILL UP REP_ARG*/
```

```
ALLOCATE REP_ARG SET (P1);
```

```
P1->REP_ARG.N=NUMPTR*32*DLEN+32;
```

```
P1->REP_ARG.DATA = P->BIT_STRING;
```

```
P1->REP_ARG.ID=ID;
```

```
/*CALL UP REP*/
```

```
CALL TCALL ('REP',1,P1,BP);
```

```
/****** MEMORY REQUEST TRACE *****//
IF FFF THEN CALL DUMPT('REPLCED',ID,P); DCL FFF BIT(1) EXT; /*//
/*****//
```

```
RTN_CODE=BP->REP_RTN_CODE;
```

```
CALL SVC3(P1); CALL SVC3(BP);
```

```
RETURN;
```

```
END;
```

```
DUM00010
DUM00020
DUM00030
DUM00040
DUM00050
DUM00060
DUM00070
DUM00080
DUM00090
DUM00100
DUM00110
DUM00120
DUM00130
DUM00140
DUM00150
DUM00160
DUM00170
DUM00180
DUM00190
DUM00200
DUM00210
DUM00220
DUM00230
DUM00240
DUM00250
DUM00260
```

```
RET: PROC;

/* A T-PROC AT THE MM LEVEL FOR RETRIEVAL OF BEU GIVEN ITS ID */
/* FOR INTEGRATED VERSION: MAKE USE OF SUBROUTINE GSEU (GET SEU) */

%INCLUDE ARET, SERVICE, DSEU, EGSEU;
DCL (P.RTP,TP) PTR;
DCL (UNSPEC,NULL) BUILTIN;
DCL (I,RTN_CODE) FIXED BIN;

CALL TBEG (PROC_ADDR,P);
ALLOCATE RET_RTN SET (RTP);

CALL GSEU (RET_ARG.ID,SEUPTR,RTN_CODE); /* GET SEU DESIGNATED BY ID */
IF RTN_CODE ^= 0 THEN RTP->RET_RTN_CODE = 1;
ELSE DO;
  RTP->RET_RTN.N = SEU.N*8;
  RTP->RET_RTN.DATA = UNSPEC(SEU.DATA);
END;

CALL TRTN (PROC_ADDR,RTP);
RETURN;
END;
```

RET00010
RET00020
RET00030
RET00040
RET00050
RET00060
RET00070
RET00080
RET00090
RET00100
RET00110
RET00120
RET00130
RET00140
RET00150
RET00160
RET00170
RET00180
RET00190
RET00200
RET00210
RET00220
RET00230

FILE: RETB PLIOPT A1 VM/SP CONVERSATIONAL MONITOR SYSTEM

```

RETB: PROC (PARENT,BINFO,PTR1,PTR2,RTN_CODE);
/* A S-PROC WITHIN N-ARY LEVEL WHICH OBTAINS THE ID OF THE BINARY
* ASSOCIATED ELEMENT IN BSETID OF ID1 IN PARENT PSET */
XINCLUDE OBCAT.ESUB1.DBEU;
XINCLUDE EBCT;
DCL PARENT BIT(32) ALIGNED;
(RTN_CODE,POS) FIXED BIN;
DCL (PTR1,PTR2) PTR.(NULL,UNSPEC) BUILTIN;
DCL ID2 BIT(32);
POS=BINFO.POS; /*GET PTR SLOT FOR BSET */

IF PTR1=NULL THEN DO; RTN_CODE=10+RTN_CODE; RETURN: END;

ID2=PTR1->BEU.ID_ARRAY(POS);

IF ID2=UNSPEC(NULL)
THEN DO;
  PTR2 = NULL;
  RTN_CODE = 0;
END;
ELSE CALL RET'(ID2,PTR2,RTN_CODE); /*CHECK IF ID2 LEGAL */
RETURN;
END /*RETB */;

```

```

RET00010
RET00020
RET00030
RET00040
RET00050
RET00060
RET00070
RET00080
RET00090
RET00100
RET00110
RET00120
RET00130
RET00140
RET00150
RET00160
RET00170
RET00180
RET00190
RET00200
RET00210
RET00220
RET00230
RET00240
RET00250

```

```

RETE: PROC
/*T-PROC TO PROCESS RETE **/
XINCLUDE ARETE.FDEBUG.SERVICE.ARETN;
XINCLUDE DECAT;
XINCLUDE EGECT.EGACT.EVTPE;

DCL (P,TP,RP,LP,TP1,RTN,PP) PTR, NULL BUILTIN;
DCL(RTN_CODE,1,J) FIXED BIN, NUM FIXED BIN (31);
DCL BDATA BIT (320), CDATA CHAR (40), OLEN FIXED BIN;
DCL TO_BIT FIXED BIN INIT (1), TO_CHAR FIXED BIN INIT (2);
DCL (ENAME,NAME,NAME1) CHAR (8);
DCL 1 MAINFO (25), /*AN ARRAY OF AINFO TO STORE RETRIEVED ACAT ENT*/
      2 BSETID BIT (32), /*BSETID OF THIS ATTRIBUTE**/
      2 PSETID BIT (32), /*PSETID OF TARGET NODE**/
      2 ENAME CHAR (8), /*IF TARGET IS A ENTITY SET*/
      2 MAX FIXED BIN (31),
      2 MIN FIXED BIN (31),
      2 MLEN FIXED BIN,
      2 FUNC CHAR(1),
      2 TYPE CHAR (1),
      2 VTYPE CHAR (1);
DCL NODE FIXED BIN;

/*BEGIN PROCESSING**/
CALL TBEG (PROC_ADDR,P);
ALLOCATE RETE_RTN SET (RTP);
RTP->RETE_RTN_CODE=0; /*INIT TO 0.K.*/
ENAME=RETE_ARG.NAME;
MAINFO.ENAME(1)=ENAME; /*INIT AINFO ARRAY'S FIRST TOKEN*/
/*ALLOCATE ROOT NODE**/
CALL GECT (ENAME,EINFO,RTN_CODE); /*GECT GETS ENTITY CAT ENTRY */
IF RTN_CODE=0 THEN DO;
  RTP->RETE_RTN_CODE=100; /*100 IS NO SUCH ENTITY**/
  CALL TRTN (PROC_ADDR,RTP); /*RETURNS**/
  RETURN;
END;

I=0; /*NO PRED*/
ALLOCATE RETN_ARG SET (TP);
RP,LP=TP;
I-TP->RETN_ARG.CBTP; J=TP->RETN_ARG.LEN; PP=TP->RETN_ARG.PTR;
TP->RETN_ARG=RETE_ARG, BY NAME;
TP->RETN_ARG.CBTP=I; TP->RETN_ARG.LEN=J; TP->RETN_ARG.PTR=PP;

TP->RETN_ARG.PSETID=EINFO;

/*FOLLOW RETE TREE **/
DO P=RETE_ARG.PTR REPEAT P->RETE_ARG.PTR WHILE (P^NULL);
  NODE=RETE_ARG.NODE;
  ENAME1=MAINFO.ENAME(RETE_ARG.PARENT);
  ANAME=RETE_ARG.NAME;
  CALL GACT (ANAME,ENAME1,AINFO,RTN_CODE); /*GACT GETS ATTR CAT ENT*/
  IF RTN_CODE=0 THEN DO;
    RTP->RETE_RTN_CODE=NODE+100; /*100+ IS NO SUCH ATTRIBUTE**/
    CALL SVC3 (RP); /*FREE UP RETN ALLOCATION UP TO NOW*/
    CALL TRTN (PROC_ADDR, RTP); /*RETURNS**/
  
```

```

RETO0010
RETO0020
RETO0030
RETO0040
RETO0050
RETO0060
RETO0070
RETO0080
RETO0090
RETO0100
RETO0110
RETO0120
RETO0130
RETO0140
RETO0150
RETO0160
RETO0170
RETO0180
RETO0190
RETO0200
RETO0210
RETO0220
RETO0230
RETO0240
RETO0250
RETO0260
RETO0270
RETO0280
RETO0290
RETO0300
RETO0310
RETO0320
RETO0330
RETO0340
RETO0350
RETO0360
RETO0370
RETO0380
RETO0390
RETO0400
RETO0410
RETO0420
RETO0430
RETO0440
RETO0450
RETO0460
RETO0470
RETO0480
RETO0490
RETO0500
RETO0510
RETO0520
RETO0530
RETO0540
RETO0550

```

VM/SP CONVERSATIONAL MONITOR SYSTEM

FILE: RETE PLOIPT A1

```

RETURN;
END;
MAINFO(NODE)=AINFO; /*ENTER INTO AINFO ARRAY*/

/*VERIFY PREDICATE DATA **/
IF RETE_ARG.N^=0 THEN DO;
  CDATA=RETE_ARG.PRED.CDATA(1);
  DLEN=RETE_ARG.PRED.DLEN(1);
  CALL VPIE (AINFO,TO_BIT,BDATA,CDATA,DLEN,RTN_CODE);
  IF RTN_CODE^=0 THEN DO; /*PROBLEMS**/
    RTP->RETE_RTN.RTN_CODE=NODE; /*1+ IS ILLEGAL NODE DATA*/
    CALL SVC3(RP); /*FREE UP RETN ALLOCATION UP TO NOW*/
    CALL TRTN (PROC_ADDR, RTP); /*RETURNS*/
    RETURN;
  END;
END;

/*REPLACE ANAME WITH PSETID AND BSETID AND FIX UP DATA**/
I=RETE_ARG.N;
ALLOCATE RETN_ARG SET (TP);
LP->RETN_ARG.PTR=TP; LP=TP;

I=TP->RETN_ARG.CBTP; J=TP->RETN_ARG.LEN; PP=TP->RETN_ARG.PTR;
TP->RETN_ARG=RETE_ARG, BY NAME;
TP->RETN_ARG.CBTP=I; TP->RETN_ARG.LEN=J; TP->RETN_ARG.PTR=PP;

TP->RETN_ARG.PSETID=AINFO.PSETID;
TP->RETN_ARG.BSETID=AINFO.BSETID;
IF RETE_ARG.N^=0 THEN DO;
  TP->RETN_ARG.PRED.DLEN(1)=DLEN;
  TP->RETN_ARG.PRED.DATA(1)=BDATA;
END;
END; /*DO P=RETE_ARG.PTR...*/

/*CALL UP RETN **/
CALL TCALL ('RETN',1,RP,TP);

/*CHECK RETURNS*/
I=TP->RETN_RTN_CODE;
IF I^=0 THEN DO;
  RTP->RETE_RTN.RTN_CODE=I+200; /*200+ IS RTN CODE FROM RETN**/
  CALL SVC3 (RP);
  CALL SVC3 (TP);
  CALL TRTN (PROC_ADDR, RTP);
  RETURN;
END;
RTP->RETE_RTN.N=TP->RETN_RTN.N; /*COPY*/
LP=RP;

/*COPY RETN_RTN1 INTO RETE_RTN1 WHILE FIXING UP DATA **/
DO P=TP->RETN_RTN.PTR REPEAT P->RETN_RTN1.PTR WHILE (P^=NULL);

/* INCLUDE HEX:*/
/* PUT SKIP LIST ('(RETE) RETN_RTN1', RETN_RTN1.HEX(RETE_RTN1.PTR));*/
ALLOCATE RETE_RTN1 SET (TP1);

```

FILE: N-SE PLIOPT A1 VM/SP CONVERSATIONAL MONITOR SYSTEM

```

LP->RETE_RTN1.PTR=TP1; LP=TP1;
I=TP1->RETE_RTN1.CBTP; J=TP1->RETE_RTN1.LEN; PP=TP1->RETE_RTN1.PTR;
TP1->RETE_RTN1.RETN_RTN1, BY NAME; /*COPY*/
TP1->RETE_RTN1.CBTP=I; TP1->RETE_RTN1.LEN=J; TP1->RETE_RTN1.PTR=PP;
/*FIX UP DATA */
NODE=RETN_RTN1.NODE;
IF NODE = 1 THEN DO; /*ROOT HAS NO DATA */
TP1->RETE_RTN1.DLEN=0; GO TO END; END;
AINFO=MAINFO(NODE);
SELECT(AINFO.TYPE);
WHEN ('V') DO;
    BDATA=RETN_RTN1.DATA;
    DLEN=RETN_RTN1.DLEN;
    CALL VTPE (AINFO.TO_CHAR, BDATA, CDATA, DLEN, RTN_CODE);
    TP1->RETE_RTN1.DLEN=DLEN;
    TP1->RETE_RTN1.CDATA=CDATA;
    END;
WHEN ('E')
    TP1->RETE_RTN1.DLEN=0; /*IGNORE DATA*/
    END /*SELECT*/; /*
PUT SKIP LIST(' (RETE)RETERTN', TP1->RETE_RTN1.HEX(TP1->RETE_RTN1.PTR));
*/ END; END;

CALL SVC3 (RP); /* FREE UP ARG AND RTN LIST FROM RETN*/
CALL SVC3 (TP);
CALL TRTN (PROC_ADDR, RTP); /*RETURNS WITH ARGUMENTS*/
DO P-RTP REPEAT RETE_RTN.PTR WHILE (P=NULL);
END;
END /*RETE*/;

```

RETO1110
RETO1120
RETO1130
RETO1140
RETO1150
RETO1160
RETO1170
RETO1180
RETO1190
RETO1200
RETO1210
RETO1220
RETO1230
RETO1240
RETO1250
RETO1260
RETO1270
RETO1280
RETO1290
RETO1300
RETO1310
RETO1320
RETO1330
RETO1340
RETO1350
RETO1360
RETO1370
RETO1380
RETO1390
RETO1400
RETO1410

```

RETN:PROC;
/*****
*
*      MODULE      DESCRIPTION
*
*****/
PURPOSE:
*****
THIS IS A T-PROC AT THE N-ARY LEVEL WHICH PROCESSES
RETRIEVAL REQUESTS EXPRESSED IN A RETN TREE AS
DEMONSTRATED BY THE RETN_ARG; IT RETURNS A LINK
LIST OF DATA FOR EACH NODE IN THE RETN TREE.
*****

METHOD:
*****
1. THIS MODULE USES A VERY SIMPLE METHOD TO TRAVERSE
THE DATA BASE; IT FIRST VISITS THE ROOT NODE AND
OBTAINS THE FIRST OCCURRENCE OF THE ROOT NODE; IT
THEN VERIFIES THE OCCURRENCE AGAINST THE PREDICATE
(IF ANY) TO DECIDE WHETHER THIS OCCURRENCE IS VALID;
IF YES, THEN CONTINUE; OTHERWISE DISCARD IT AND
GET THE NEXT ROOT OCCURRENCE UNTIL ALL ROOT OCCURRENCES
ARE EXAMINED.
*****
2. AFTER A ROOT OCCURRENCE IS ESTABLISHED, THIS ROUTINE
FOLLOWS THE RETN TREE TO ESTABLISH OCCURRENCE OF ITS
ASSOCIATED ELEMENTS AND VERIFY THEIR CONTENT AGAINST
THEIR PREDICATES. IF ANY MISMATCH IS ENCOUNTERED, THE
CURRENT ROOT NODE OCCURRENCE IS ABANDONED AND A NEW
ROOT NODE OCCURRENCE IS OBTAINED AS IN 1.
*****
3. AFTER ONE OCCURRENCE OF THE ENTIRE RETN TREE IS ESTABLISH
ED, A SUBROUTINE BUILD IS CALLED TO GENERATE THE DATA
FROM THE CURSOR IDS; AFTER WHICH THE NEXT ROOT NODE
IS OBTAINED AS IN 1.
*****
INPUT PARAMETERS:
*****
AS INDICATED IN RETN_ARG;
*****
OUTPUT PARAMETERS:
*****
AS INDICATED IN RETN_RTN;
*****
CALLING ARGUMENTS :
*****
%INCLUDE ARETN;
/****
CALLS PROCEDURES:
*****
INTER-LEVEL T-PROC: NONE
INTRA-LEVEL T-PROC: NONE
*****
INTRA-LEVEL S-PROC: BUILD,VERIF,RETB,RETP;
*****
%INCLUDE EBUIL,EVERT,ERETB,ERETP;
/****
CONTROL STRUCTURE, PANEL MANAGER AND DEBUGGING FACILITIES:
%INCLUDE SERVICE,FDEBUG;
/****

RETO0010
RETO0020
RETO0030
*
RETO0040
*
RETO0050
/
RETO0060
RETO0070
RETO0080
RETO0090
RETO0100
RETO0110
RETO0120
RETO0130
RETO0140
RETO0150
RETO0160
RETO0170
RETO0180
RETO0190
RETO0200
RETO0210
RETO0220
RETO0230
RETO0240
RETO0250
RETO0260
RETO0270
RETO0280
RETO0290
RETO0300
RETO0310
RETO0320
RETO0330
RETO0340
RETO0350
RETO0360
RETO0370
RETO0380
RETO0390
*
RETO0400
RETO0410
RETO0420
RETO0430
RETO0440
*
RETO0450
RETO0460
RETO0470
RETO0480
RETO0490
RETO0500
RETO0510
RETO0520
RETO0530
RETO0540
RETO0550
/
%INCLUDE EBCT,EPCT,DBEU;
DCL CURSOR (25) PTR INIT((25) NULL); /* CURRENT PTR TO NODES */
DCL PSET (25) BIT(32) ALIGNED; /*TEMP FOR PSETID OF RETN TREE NDS */
DCL (P.MP,RTB,TP,RP,LP,ROOT_PTR) PTR.(NULL,SUBSTR,UNSPEC) BUILTN;
DCL (PTR1,PTR2) PTR;

```

VM/SP CONVERSATIONAL MONITOR SYSTEM

FILE: REIN PLIOPT A1

```

DCL (RNODE,RTN_CODE) FIXED BIN;
DCL BDATA BIT(320) VARYING ALIGNED;
DCL (OP_BIT(8), YES_BIT(1)) ALIGNED;
DCL (PARENT,BSETID) BIT(32) ALIGNED;
DCL (K,MODE,1) FIXED BIN;
/*DCL OF BCAT TEMPLATE*/
DCL 1 BINFO (25),
2 PSETID(2) BIT (32),
2 POS FIXED BIN, /*POS OF POINTER ARRAY OF PSETID1 USED */
2 FUNC CHAR (1); /*'S' OR 'M'*/;
/*DCLs OF PSET CATALOGUE AT THE N-ARY LEVEL*****
DCL 1 PINFO (25),
3 NUMPTR FIXED BIN,
3 PLEN /*IN NUMBER OF BYTES*/ FIXED BIN,
3 PTYPE CHAR (1), /*'N' OR 'X' OR 'B' OR 'C'*/
3 LTYPE BIT (8), /*'00000001' IS LINK LIST*/
3 L_ID BIT (32), /*ID OF THE FIRST BEU IN THIS SET*/
3 L_POS FIXED BIN, /*POS IN ID_ARRAY USED FOR LINEAR CHAINING
OF BEU'S WITHIN THIS SET*/
3 L_POS2 FIXED BIN,
3 MAP BIT (32); /*UP TO 16 PTRS ARE ALLOWED FOR A BEU*/
/*BEGIN PROCESSING*/

CALL TBEG(PROC_ADDR,MP);
ALLOCATE RETN_RTN_SET (RTP);
RTP->RETN_RTN_CODE=0; /*INIT RTN CODE TO 0.K.*/
LP=RTP; /*LP INIT TO FIRST TOKEN TO RETURN*/ K=0;
P=MP;
RNODE=RETN_ARG.NODE; /*RNODE IS ROOT NODE# */
/*INIT PSET STRUCTURE*/

/***** RETN REQUEST TRACE *****/
DCL FFFN BIT (1) EXT;
IF FFFN THEN DO;
PUT SKIP EDIT('NODE *PSETID* PAR **GET** N *BSETID* ***OP*** DL DATA')RETN_ARG.DL_DATA;
(A);
DO P=MP REPEAT P->RETN_ARG.PTR WHILE (P^=NULL);

PUT SKIP EDIT(RETN_ARG.NODE,HEX4(ADDR(RETN_ARG.PSETID)->BP),
RETN_ARG.PARENT,RETN_ARG.GET,RETN_ARG.N,' ');
(F(3),X(2),A(8),F(3),X(2),B(8),F(2),A);
IF RETN_ARG.PARENT^=0 THEN PUT EDIT(
HEX4(ADDR(RETN_ARG.BSETID)->BP))(A(8));
ELSE PUT EDIT('')(A);
IF RETN_ARG.N^=0
THEN DO;
PUT EDIT(RETN_ARG.OP(1),RETN_ARG.DLEN(1),' ');
(X(1),B(8),F(3),A);
DO I=1 TO RETN_ARG.DLEN(1)/4;DCL I1 FIXED BIN;
PUT EDIT(HEX4(ADDR(RETN_ARG.DATA(1))->PAA(I1)))(A(8));
XINCLUDE HEX; DCL BP PTR BASED;
END;
DCL PAA(10) PTR BASED;
END;

```

```

RETO0560
RETO0570
RETO0580
RETO0590
RETO0600
RETO0610
RETO0620
RETO0630
RETO0640
RETO0650
RETO0660
RETO0670
RETO0680
RETO0690
RETO0700
RETO0710
RETO0720
RETO0730
RETO0740
RETO0750
RETO0760
RETO0770
RETO0780
RETO0790
RETO0800
RETO0810
RETO0820
RETO0830
RETO0840
RETO0850
RETO0860
RETO0870
RETO0880
RETO0890
RETO0900
RETO0910
RETO0920
RETO0930
RETO0940
RETO0950
RETO0960
RETO0970
RETO0980
RETO0990
RETO1000
RETO1010
RETO1020
RETO1030
RETO1040
RETO1050
RETO1060
RETO1070
RETO1080
RETO1090
RETO1100

```

FILE: RETN PLIOPT A1 VM/SP CONVERSATIONAL MONITOR SYSTEM

```

END:
END:
/*****
/****/RETO1110
/****/RETO1120
/****/RETO1130
/****/RETO1140
/****/RETO1150
/****/RETO1160
/****/RETO1170
/****/RETO1180
/****/RETO1190
/****/RETO1200
/****/RETO1210
/****/RETO1220
/****/RETO1230
/****/RETO1240
/****/RETO1250
/****/RETO1260
/****/RETO1270
/****/RETO1280
/****/RETO1290
/****/RETO1300
/****/RETO1310
/****/RETO1320
/****/RETO1330
/****/RETO1340
/****/RETO1350
/****/RETO1360
/****/RETO1370
/****/RETO1380
/****/RETO1390
/****/RETO1400
/****/RETO1410
/****/RETO1420
/****/RETO1430
/****/RETO1440
/****/RETO1450
/****/RETO1460
/****/RETO1470
/****/RETO1480
/****/RETO1490
/****/RETO1500
/****/RETO1510
/****/RETO1520
/****/RETO1530
/****/RETO1540
/****/RETO1550
/****/RETO1560
/****/RETO1570
/****/RETO1580
/****/RETO1590
/****/RETO1600
/****/RETO1610
/****/RETO1620
/****/RETO1630
/****/RETO1640
/****/RETO1650
*****/

DO P=MP REPEAT P->RETN_ARG.PTR WHILE (P^=NULL);
IF RETN_ARG.NODE>25
THEN DO;
RTP->RETN_ARG.NODE=1;
IF FDEBUG.LEV2 THEN PUT SKIP LIST ('(RETN) ERR RTN ');
'PSET DATA STRUCTURE OVER FLOW (LIMIT 25)';
GO TO RTN;
END;
CALL GPCT (RETN_ARG.PSETID, PINF(RTN_ARG.NODE), RTN_CODE);
IF RETN_ARG.NODE ^= RNODE
THEN CALL GBCT (RETN_ARG.BSETID, BINFO (RETN_ARG.NODE), RTN_CODE);
PSET (RETN_ARG.NODE)=RETN_ARG.PSETID;
END;
CALL RETP (PINF(RNODE), NULL, ROOT_PTR, RTN_CODE); /*1ST BEU*/
/*BEGIN TREE PROCESSING*/
OUTER_LOOP:
DO WHILE ('1'8);
/*ROOT NODE PROCESSING*/
P=MP;
IF ROOT_PTR = NULL
THEN LEAVE OUTER_LOOP;
ELSE CURSOR(RNODE) = ROOT_PTR;
/*ROOT ARG.N ^= 0
THEN DO; /*VERIFY NODE DATA AGAINST PREDICATE*/
OP=RETN_ARG.PRED.OP(1);
BDATA=SUBSTR(RETN_ARG.PRED.DATA(1),1,
RETN_ARG.PRED.DLEN(1)*8);
CALL VERIF (PINF(RNODE), ROOT_PTR.OP, BDATA, RTN_CODE, YES);
IF RTN_CODE ^= 0
THEN DO;
RTP->RETN_ARG.NODE=3; /****** RTN CODE 3 ******/
IF FDEBUG.LEV2 THEN PUT SKIP LIST ('(RETN) ERR RTN ');
/*AFTER CALLING VERIF WHICH GIVES RTN CODE', RTN_CODE);
GO TO RTN;
END;
IF ^YES THEN GO TO NEXT_ROOT; /*DISCARD THIS ROOT NODE*/
END;

/*NEXT RET TREE LEAF*/
DO P=RETN_ARG.PTR REPEAT P->RETN_ARG.PTR WHILE (P^=NULL);
NODE=RETN_ARG.NODE;
PARENT =PSET (RETN_ARG.PARENT);
BSETID=RETN_ARG.BSETID;
PTR1=CURSOR (RETN_ARG.PARENT);
IF PTR1 = NULL
THEN DO;
IF RETN_ARG.N ^= 0 THEN GO TO NEXT_ROOT;
CURSOR (NODE) = NULL;

```

```

END;
ELSE DO;
  CALL RETB(PARENT.BINFO(NODE),PTR1,PTR2,RTN_CODE);
  IF RTN_CODE ^= 0
  THEN DO;
    RTP->RETN_RTN_CODE=4; /****** RTN CODE 4 *****/
    IF FDEBUG.LEV2 THEN PUT SKIP LIST ('(RETN) ERR RTN ',
    'AFTER CALLING RETB WHICH GIVES RTN CODE',RTN_CODE);
    GO TO RTN;
  END;
  CURSOR(NODE)=PTR2;
  IF PTR2=NULL & RETN_ARG.N ^= 0 THEN GO TO NEXT_ROOT;
  /* IF PREDICATE ISSUED THEN DISCARD NULL ATTRIBUTE */
  IF PTR2 ^= NULL & RETN_ARG.N ^= 0
  THEN DO;
    OP=RETN_ARG.PRED.OP(1);
    BDATA = SUBSTR(RETN_ARG.PRED.DATA(1),1,
    RETN_ARG.PRED.DLEN(1)*8);
    CALL VERIF(PINFO(RNODE),PTR2.OP,BDATA,RTN_CODE.YES);
    IF RTN_CODE ^= 0
    THEN DO;
      RTP->RETN_RTN_CODE=5; /*** RTN CODE 5 ***/
      IF FDEBUG.LEV2 THEN PUT SKIP LIST ('(RETN) ERR RTN ',
      'AFTER CALLING VERIF WHICH GIVES RTN CODE',RTN_CODE);
      GO TO RTN;
    END;
    IF ^YES THEN GO TO NEXT_ROOT; /*DISCARD CURRENT ROOT*/
  END;
END;

/*NOW A COMPLETE SET OF CURSOR HAS BEEN SET UP*/
K=K+1; /*INDEX TO OCCUR OF ROOT NODE*/
CALL BUILD (CURSOR,PSET,MP,K,LP,RTN_CODE);
IF RTN_CODE ^= 0
THEN DO;
  RTP->RETN_RTN_CODE=6; /****** RTN CODE 6 *****/
  IF FDEBUG.LEV2 THEN PUT SKIP LIST ('(RETN) ERR RTN ',
  'AFTER CALLING BUILD WHICH GIVES RTN CODE',RTN_CODE);
  GO TO RTN;
END;
NEXT_ROOT:
CALL RETP(PINFO(RNODE),CURSOR(RNODE),ROOT_PTR,RTN_CODE);
DO I = 1 TO 25;
  IF CURSOR(I) ^= NULL
  THEN DO;
    FREE CURSOR(I)->BEU;
    CURSOR(I) = NULL;
  END;
END;
IF RTN_CODE ^= 0
THEN DO;
  RTP->RETN_RTN_CODE=2; /******RTN CODE 2 *****/

```

RETO1660
 RETO1670
 RETO1680
 RETO1690
 RETO1700
 RETO1710
 RETO1720
 RETO1730
 RETO1740
 RETO1750
 RETO1760
 RETO1770
 RETO1780
 RETO1790
 RETO1800
 RETO1810
 RETO1820
 RETO1830
 RETO1840
 RETO1850
 RETO1860
 RETO1870
 RETO1880
 RETO1890
 RETO1900
 RETO1910
 RETO1920
 RETO1930
 RETO1940
 RETO1950
 RETO1960
 RETO1970
 RETO1980
 RETO1990
 RETO2000
 RETO2010
 RETO2020
 RETO2030
 RETO2040
 RETO2050
 RETO2060
 RETO2070
 RETO2080
 RETO2090
 RETO2100
 RETO2110
 RETO2120
 RETO2130
 RETO2140
 RETO2150
 RETO2160
 RETO2170
 RETO2180
 RETO2190
 RETO2200

FILE: RETN PLIOPT A1

VM/SP CONVERSATIONAL MONITOR SYSTEM

PAGE 005

```
IF FDEBUG.LEV2 THEN PUT SKIP LIST ('(RETN) ERR RTN ',  
'AFTER CALLING RETP WHICH GIVES RTN CODE',RTN_CODE);  
GO TO RTN;  
END;
```

```
IF MP->RETN_ARG.GET=RETNGET.ANY & K > 0 /*DONE*/  
THEN DO;
```

```
IF ROOT_PTR ^= NULL THEN FREE ROOT_PTR->BEU;  
LEAVE OUTER_LOOP;  
END;
```

```
END /*DO WHILE*/;
```

```
RTP->RETN_RTN.N = K;
```

```
RTN: CALL TRTN(PROC_ADDR.RTP);
```

```
END /*RETN*/;
```

RETO2210
RETO2220
RETO2230
RETO2240
RETO2250
RETO2260
RETO2270
RETO2280
RETO2290
RETO2300
RETO2310
RETO2320
RETO2330
RETO2340

FILE: REIP PLIOPT A1 VM/SP CONVERSATIONAL MONITOR SYSTEM

```

RETP: PROC(PINFO,PTR1,PTR2,RTN_CODE);
/* A S-PROC AT THE N-ARY LEVEL WHICH RETRIEVES THE ID OF THE P ELEM
* THAT FOLLOWS ID1 IN PSETID. IF ID1 IS NULL THEN IT RETRIEVES THE
* THE FIRST OCCR IN PSETID*/
XINCLUDE DBEU.DPCAT;
XINCLUDE EPCT.ESUB1;

DCL RTN_CODE FIXED BIN;
DCL (P,PTR1,PTR2) PTR,(NULL,UNSPEC) BUILTIN;
DCL ID2 BIT (32);

IF PTR1= NULL /*GET FIRST*/
THEN ID2=PINFO.L_ID;
ELSE ID2=PTR1->BEU.ID_ARRAY(PINFO.L_POS);
IF ID2 = UNSPEC (NULL) THEN DO;
RTN_CODE = 0;
PTR2 = NULL;
RETURN;
END;
CALL RET1 (ID2, PTR2, RTN_CODE);
RETURN;
END;

```

```

RET00010
RET00020
RET00030
RET00040
RET00050
RET00060
RET00070
RET00080
RET00090
RET00100
RET00110
RET00120
RET00130
RET00140
RET00150
RET00160
RET00170
RET00180
RET00190
RET00200
RET00210
RET00220
RET00230
RET00240
RET00250

```

FILE: RET1 PLOPT A1 VM/SP CONVERSATIONAL MONITOR SYSTEM

```

RET1:PROG(ID,P,RTN_CODE);
/*S-PROC AT THE LOWEST LAYER OF THE N-ARY LEVEL TO SET UP CALL FOR
** RETRIEVAL OF ELEMENT DESIGNATED BY ID*/

XINCLUDE SERVICE,ARET,DBEU;
DCL ID BIT (32), (I,J,K) FIXED BIN, (P,RP) PTR;
DCL (SUBSTR,UNSPEC,NULL,ADDR) BUILTIN, RTN_CODE FIXED BIN;

ALLOCATE RET ARG; RP=P;
RET_ARG.ID=ID;
CALL TCALL ('RET',1,RP,AP);

/***** */
IF FFF
THEN CALL DUMPT ('RETRIEVED',ID,NULL); /**/
DCL FFF BIT (1) EXT; /**/
/***** */

CALL SVC3 (RP);

IF AP->RET_RTN_CODE=0 THEN DO; /*ID INVALID*/
CALL SVC3(AP);
P=NULL;
RETURN;
END;

P=ADDR(AP->RET_RTN_DATA);
I=NUMPTR;J=DLEN;
ALLOCATE BEU SET (RP);
RP->BEU.DATA=BEU.DATA;
RP->BEU.ID_ARRAY=BEU.ID_ARRAY;
CALL SVC3 (AP);
P=RP;
RTN_CODE=0;
RETURN;
END/*RET1*/;

```

```
SETFH: PROC:/* SUBROUTINE FOR SETTING FH TRACE OPTIONS */
/*** FTCALL -- TCALL ONLY;
FDEBUG -- ALL MODULES AT ALL LEVELS;
FFF -- RET1, REP1, CRT1, DEL1 ONLY;
FFFN -- RETN, UPDN ONLY; *****/

DCL (P,RP) PTR; XINCLUDE FDEBUG;
DCL FTCALL(4) BIT (1) EXT;
ON ERROR SNAP SYSTEM;
PUT SKIP LIST ('ENTER FH TRACE OPTIONS');
PUT LIST ('ENTER TCALL TRACE OPTION (4 BITS)');
DO I = 1 TO 4;
  GET EDIT (FTCALL(I))(B(1));
END;

PUT LIST ('ENTER SYSTEM ERROR MESSAGE TRACE OPTION (4 BITS)');
GET EDIT (FDEBUG.LEV0) (B(1));
GET EDIT (FDEBUG.LEV1) (B(1));
GET EDIT (FDEBUG.LEV2) (B(1));
GET EDIT (FDEBUG.LEV3) (B(1));

DCL FFF BIT(1) EXT;
PUT LIST('ENTER MEMORY REQUEST TRACE OPTION (1 BIT):');
GET EDIT(FFF)(B(1));

DCL FFFN BIT (1) EXT;
PUT LIST ('ENTER UPDN/RETN REQUEST TRACE OPTION (1 BIT):');
GET EDIT (FFFN) (B(1));

DCL FTIME BIT(1) EXT;
PUT LIST ('ENTER TIMING REPORT TRACE OPTION (1 BIT):');
GET EDIT (FTIME) (B(1));
PUT SKIP LIST ('FH TRACE OPTIONS ENDS');
END;
```

SET00010
SET00020
SET00030
SET00040
SET00050
SET00060
SET00070
SET00080
SET00090
SET00100
SET00110
SET00120
SET00130
SET00140
SET00150
SET00160
SET00170
SET00180
SET00190
SET00200
SET00210
SET00220
SET00230
SET00240
SET00250
SET00260
SET00270
SET00280
SET00290
SET00300
SET00310
SET00320
SET00330
SET00340

FILE: SHWE PLIOPT A1 VM/SP CONVERSATIONAL MONITOR SYSTEM

SHWE: PROC;

```

/* T-PROC AT THE ENTITY LEVEL, INVOKED IN RESPONSE TO A CALL FROM
MODULE DDQ AT THE USER INTERFACE LEVEL TO RETRIEVE DATA DEFINITION
INFO */

```

```

XINCLUDE SERVICE, ASHWE, DECAT, EGACT, EGECT, ARETE, EALLA;
DCL (P,LP,TP,PP,RP) PTR, NULL BUILTIN, (RTN_CODE,1,N) FIXED BIN;
DCL MOD BUILTIN;

```

```

/* BEGIN */
CALL TBEG (PROC,ADDR, SHWE_ARG_P);
ALLOCATE SHWE_RTN; LP=SHWE_RTN_P;
IF NAME1='' THEN DO; /*ALL ENTITY SETS QUERIED */

```

```

/* CALL UP RETE TO RETRIEVE DATA */
ALLOCATE RETE_ARG; RP,LP=P;
RETE_ARG.NAME='E*ESET';
RETE_ARG.GET=RETEGET.ALL;

```

```

ALLOCATE RETE_ARG; LP->RETE_ARG.PTR=P; LP=P;
RETE_ARG.NODE=2;
RETE_ARG.NAME='A*ENAME';
RETE_ARG.GET=RETEGET.ALL;
RETE_ARG.PARENT=1;

```

```

CALL TCALL ('RETE',1,RP,TP);
CALL SVC3 (RP);

```

```

/* EXAMINE RETURNS */
P=TP;

```

```

IF RETE_RTN.RTN_CODE=0 THEN DO;
  SHWE_RTN.RTN_CODE=30+RETE_RTN.RTN_CODE; /* 30+ *****/
  CALL SVC3 (P); GO TO RTN; END;

```

```

IF RETE_RTN.N=0 THEN DO;
  SHWE_RTN.RTN_CODE=-2; /* -2 IS NO ENTITY SET DEFINED *****/
  CALL SVC3 (P); GO TO RTN; END;

```

```

/*COPY ENAMES RETRIEVED INTO SHWE_RTN2 */
N=RETE_RTN.N;
ALLOCATE SHWE_RTN2 SET (PP); SHWE_RTN.PTR=PP;
PP->SHWE_RTN2.NUM_ENAME=N;

```

```

DO I = 1 TO N;
  DO P= RETE_RTN.PTR REPEAT RETE_RTN1.PTR WHILE (P^=NULL);
    IF RETE_RTN1.NODE=2 /* ENAME NODE */
    THEN DO; PP->SHWE_RTN2.ENAME(1)=RETE_RTN1.CDATA;
      GO TO NEXTI;
    END; /*GET OUT P LOOP */
  END /* P LOOP */;
NEXTI: END /* DO I */;

```

```

CALL SVC3 (TP); GO TO RTN; /* DONE */

```

```

SHW00010
SHW00020
SHW00030
SHW00040
SHW00050
SHW00060
SHW00070
SHW00080
SHW00090
SHW00100
SHW00110
SHW00120
SHW00130
SHW00140
SHW00150
SHW00160
SHW00170
SHW00180
SHW00190
SHW00200
SHW00210
SHW00220
SHW00230
SHW00240
SHW00250
SHW00260
SHW00270
SHW00280
SHW00290
SHW00300
SHW00310
SHW00320
SHW00330
SHW00340
SHW00350
SHW00360
SHW00370
SHW00380
SHW00390
SHW00400
SHW00410
SHW00420
SHW00430
SHW00440
SHW00450
SHW00460
SHW00470
SHW00480
SHW00490
SHW00500
SHW00510
SHW00520
SHW00530
SHW00540
SHW00550

```

```

END /*IF ENAME = '' */;

/*CHECK ENTITY SET , WHEN ENTITY SET NAME IS GIVEN */
CALL GECT (NAME1,AINFO,RTN_CODE);
IF RTN_CODE ^= 0 THEN DO;
  SHWE_RTN_CODE=-1; /****** -1 IS PROBLEM WITH ENTITY SET NAME */
  GO TO RTN; END;

/*CHECK ATTRIBUTE */
IF NUM_NAME2=999 /* ALL ATTRIBUTES */
THEN DO; /* CALL UP ALLA TO RETRIEVE ALL ATTR NAMES */
  NUM_NAME2 = 0;
  CALL ALLA (NAME1,NAME2,NUM_NAME2,RTN_CODE);
  IF RTN_CODE ^= 0 THEN DO;
    SHWE_RTN_CODE=500 + RTN_CODE; /* 500+ ***** */
    GO TO RTN; END;
  IF NUM_NAME2 = 0 THEN DO; /* NO ATTRIBUTES DEFINED */
    SHWE_RTN_CODE=-3; /* -3 IS NO ATTRIBUTES FOUND ***** */
    GO TO RTN; END;
  END /* IF 999 */;

/* BY NOW NAME2 CONTAINS ATTRIBUTES TO BE SHOWN */
LP=SHWE_RTN_P;
DO I = 1 TO NUM_NAME2;
  CALL GACT (NAME2(I),NAME1,AINFO,RTN_CODE);
  IF RTN_CODE ^= 0 THEN DO;
    SHWE_RTN_CODE=I; /* 0+ IS PROBLEM WITH ATTRIBUTE NAME **** */
    GO TO RTN; END;

  ALLOCATE SHWE_RTN1 SET (TP);
  LP->SHWE_RTN1.PTR=TP; LP=TP;
  TP->SHWE_RTN1.DATA=AINFO, BY NAME; /* COPY AINFO INTO RETURN TOKEN */
  TP->SHWE_RTN1.ANAME=NAME2(I);
  END /* DO I */;

RTN: CALL TRTN (PROC_ADDR, SHWE_RTN_P);
RETURN;
END /*SHWE */;

```

SHW00560
SHW00570
SHW00580
SHW00590
SHW00600
SHW00610
SHW00620
SHW00630
SHW00640
SHW00650
SHW00660
SHW00670
SHW00680
SHW00690
SHW00700
SHW00710
SHW00720
SHW00730
SHW00740
SHW00750
SHW00760
SHW00770
SHW00780
SHW00790
SHW00800
SHW00810
SHW00820
SHW00830
SHW00840
SHW00850
SHW00860
SHW00870
SHW00880
SHW00890
SHW00900
SHW00910
SHW00920
SHW00930
SHW00940
SHW00950

FILE: SRCH PLIOPT A1 VM/SP CONVERSATIONAL MONITOR SYSTEM

```

SRCH: PROC(PSETID,DATA,ID,RTN_CODE);

/* A S-PROC AT THE N-ARY LEVEL WHICH LOCATES DATA IN A PSET AND
* RETURNS ITS ID, IF NOT FOUND, RTN_CODE IS SET TO 1*/

XINCLUDE DBEU,DPCAT;
XINCLUDE EPCT,ESUB1;
DCL (PSETID,ID) BIT (32), DATA BIT (*), RTN_CODE FIXED BIN, P PTR;
DCL(NULL,UNSPEC) BUILTIN;
DCL POS FIXED BIN;
CALL GPCT (PSETID, PINFO, RTN_CODE);
ID=PINFO.L.ID;
POS=PINFO.L_POS;

DO WHILE (ID^=UNSPEC(NULL));
CALL RET1 (ID,P,RTN_CODE);
IF P=NULL THEN DO;
RTN_CODE=2; /*BAD PSET CHAIN*****//
RETURN;
END;
IF BEU.DATA=DATA THEN LEAVE;
ID=BEU.ID_ARRAY(POS); /*NEXT BEU IN THIS PSET*/
END:/*DO WHILE*/

IF ID=UNSPEC(NULL) THEN RTN_CODE=1; /*NOT FOUND*/
ELSE RTN_CODE=0;
END /*SRCH*/;

```

```

SRC00010
SRC00020
SRC00030
SRC00040
SRC00050
SRC00060
SRC00070
SRC00080
SRC00090
SRC00100
SRC00110
SRC00120
SRC00130
SRC00140
SRC00150
SRC00160
SRC00170
SRC00180
SRC00190
SRC00200
SRC00210
SRC00220
SRC00230
SRC00240
SRC00250
SRC00260
SRC00270

```

240

VM/SP CONVERSATIONAL MONITOR SYSTEM

FILE: SVCS1 PLIOPT A1

```

LEN=LEN + P->ARG_LIST.LEN;
END;
/*SET UP SEND CALL*/
LEVEL=PROC_TBL.LEVEL(J);
VPID=0; BOXID=1; TYPE='S';
ALLOCATE PF_SVC;
PF_SVC.SVC = PROC_NAME;
PF_SVC.PTR = PTR;
CALL SEND (LEVEL,VPID,BOXID,TYPE,LEN,PT_SVC);
RTN_CODE=0; /*O.K.*/
END /*SVCS1*/;
SVCS0560
SVCS0570
SVCS0580
SVCS0590
SVCS0600
SVCS0610
SVCS0620
SVCS0630
SVCS0640
SVCS0650
SVCS0660

```

242

FILE: SVC3 PLIOPT A1 VM/SP CONVERSATIONAL MONITOR SYSTEM

```

SVC3:PROC(P);
/*** THIS IS A SERVICE ROUTINE TO FREE UP TOKEN LINK LIST *****/
/* TOKEN HAS BEEN REPRIEVED */
DCL 1 TOKEN BASED (P) .
  2 LEN FIXED BIN(15).
  2 CBTP FIXED BIN(15).
  2 PTR PTR.
  2 DATA CHAR (I REFER (TOKEN.LEN)); /* */

DCL (P,P1) PTR. (I.LEN,CBTP) FIXED BIN(15).
  NULL BUILTIN;
  %INCLUDE AREP.ACRT.ADEFA.ADEFB.ADEFE.ADEFP.ADEL.AMINI.ANINI.ARET;
  %INCLUDE ARETE.ARETN.AUPDE.AUPDN.AVINI.AVNME.ASHWE;
  %INCLUDE HEX;
  %INCLUDE FDEBUG;
DO WHILE (P~NULL);
  P1=TOKEN.PTR; /*SAVE NEXT PTR*/
  CBTP=TOKEN.CBTP;
  LEN=TOKEN.LEN;
/*
IF FDEBUG.LEVO
THEN PUT SKIP EDIT(' SVC3: PTR 'HEX(P)' LEN' LEN' TYP' CBTP)
(A,A(6),A,F(6),A,F(3));
*/
SELECT (CBTP);
  WHEN (1) FREE P->REP_ARG;
  WHEN (2) FREE P->REP_RTN;
  WHEN (3) FREE P->CRT_ARG;
  WHEN (4) FREE P->CRT_RTN;
  WHEN (5) FREE P->DEFA_ARG;
  WHEN (6) FREE P->DEFA_RTN;
  WHEN (7) FREE P->DEFB_ARG;
  WHEN (8) FREE P->DEFB_RTN;
  WHEN (9) FREE P->DEFE_ARG;
  WHEN (10) FREE P->DEFE_RTN;
  WHEN (11) FREE P->DEFP_ARG;
  WHEN (12) FREE P->DEFP_RTN;
  WHEN (13) FREE P->DEL_ARG;
  WHEN (14) FREE P->DEL_RTN;
  WHEN (15) FREE P->MINIT_ARG;
  WHEN (16) FREE P->MINIT_RTN;
  WHEN (17) FREE P->NINIT_ARG;
  WHEN (18) FREE P->NINIT_RTN;
  WHEN (19) FREE P->RET_ARG;
  WHEN (20) FREE P->RET_RTN;
  WHEN (21) FREE P->RETE_ARG;
  WHEN (22) FREE P->RETE_RTN;
  WHEN (23) FREE P->RETE_ARG1;
  WHEN (24) FREE P->RETE_RTN1;
  WHEN (25) FREE P->RETN_ARG;
  WHEN (26) FREE P->RETN_RTN;
  WHEN (27) FREE P->UPDE_ARG;
  WHEN (28) FREE P->UPDE_RTN;
  WHEN (29) FREE P->UPDE_ARG;
  WHEN (29) FREE P->UPDE_RTN;

```

VM/SP CONVERSATIONAL MONITOR SYSTEM

FILE: SVC3 PLIOPT A1

```

WHEN(30) FREE P->UPDN_ARG;
WHEN(31) FREE P->UPDN_RTN;
WHEN(32) FREE P->VINIT_ARG;
WHEN(33) FREE P->VINIT_RTN;
WHEN(34) FREE P->VNME_ARG;
WHEN(35) FREE P->VNME_RTN;
WHEN(36) FREE P->SHWE_ARG;
WHEN(37) FREE P->SHWE_RTN;
WHEN(38) FREE P->SHWE_RTN1;
WHEN(39) FREE P->SHWE_RTN2;

OTHERWISE DO; PUT SKIP LIST ('***SVC3 WARNING',CBTP);
SIGNAL ERROR; END;
END;
P=P1;
END;
/*
IF FDEBUG.LEVO THEN PUT EDIT(' *** SVC3: END') (A);
*/
RETURN;
END /*SVC3*/;
SVC00560
SVC00570
SVC00580
SVC00590
SVC00600
SVC00610
SVC00620
SVC00630
SVC00640
SVC00650
SVC00660
SVC00670
SVC00680
SVC00690
SVC00700
SVC00710
SVC00720
SVC00730
SVC00740
SVC00750
SVC00760

```

FILE: TBEG PLIOPT A1

VM/SP CONVERSATIONAL MONITOR SYSTEM

PAGE 001

TBEG: PROC (PROC_ADDR,P);

 /**** A SERVICE ROUTINE TO STRIP OFF MSG TOKEN FROM A CALLING ***
 *** ARGUMENT LIST; IT IS USED BY A T-PROC WHEN IT IS INVOKED ***/
 /**** PROC_ADDR CONTAINS THE PROC_ADDR OF THE CALLING PROCEDURE. ***
 *** P WILL POINT TO THE FIRST TOKEN OF THE CALLING ARGUMENT LIST. */

DCL P PTR;

 %INCLUDE USERS.VPX,FDEBUG,ESVCS;
 PROC_ADDR=THISVP->VP.WAIT.MSG->MSG.RTN_ADDR;
 P= THISVP->VP.WAIT.MSG->MSG.PTR;
 FREE THISVP->VP.WAIT.MSG->MSG; /*FREE MSG*/
 RETURN;
END;

TBEO0010
TBEO0020
TBEO0030
TBEO0040
TBEO0050
TBEO0060
TBEO0070
TBEO0080
TBEO0090
TBEO0100
TBEO0110
TBEO0120
TBEO0130
TBEO0140
TBEO0150
TBEO0160

FILE: TCALL PLIOPT A1 VM/SP CONVERSATIONAL MONITOR SYSTEM

```

TCALL:PROC (PROC_NAME,I,RP,TP) RECURSIVE ;
/... A SERVICE ROUTINE FOR CROSS-LEVEL PROCEDURE CALLS*/
/... PROC_NAME IS THE NAME OF THE PROCEDURE TO BE INVOKED; ...
**** I IS THE BOX ID TO WAIT ON; RP POINTS TO THE BEGINNING ***
**** TOKEN OF THE CALLING ARGUMENT LIST;
**** TP POINTS TO THE BEGINNING TOKEN OF THE RETURN ARGUMENT ***
**** LIST WHEN THE CALLED PROCEDURE RETURNS *****//

DCL FTCALL(4) BIT(1) STATIC EXT; /* FTCALL IS DEBUG TRACE BIT */
DCL FTIME BIT(1) EXT; /* FTIME IS TIMING REPORT OPTION */
XINCLUDE FSTRUC;

DCL (RP,P,TP) PTR; DCL NULL BUILTIN, I FIXED BIN;
XINCLUDE USERS,VFX,FDEBUG,ESVCS;
DCL RTN_CODE FIXED BIN;

DO ENUM = 1 TO 18; /* FIND ENUM AND LEVEL */
  IF ARCH.PROCNAME(ENUM) = PROC_NAME THEN LEAVE;
END;
I=ARCH.LEVEL(ENUM);

/***** TCALL TRACE OPTION *****/
IF FTCALL(1) /* IF TCALL TRACE BIT FOR THAT LEVEL IS ON*/
  THEN PUT SKIP EDIT ('TCALL: ',PROC_NAME)(A,A);
/*****

/***** TIMING REPORT OPTION *****/
DCL (BEGTIME,ENUM,LOTIME,CIL) FIXED BIN(31);
DCL RTIMER ENTRY RETURNS(FIXED BIN(31));
DCL (ECOUNT(50),ELO(50),ETIME(50)) FIXED BIN(31) EXT INIT((50)0);
IF FTIME THEN DO;
  ALLOCATE LOTIME; LOTIME=0;
  BEGTIME=RTIMER; END;
/*****

  ALLOCATE MSG;
  MSG.PTR=RP; /*CHAIN TO ROOT PTR OF ARG LIST*/
  MSG.RTN_ADDR.LEVEL=THISVP->VP.LEVEL;
  MSG.RTN_ADDR.VPID=THISVP->VP.VPID;
  MSG.RTN_ADDR.BOXID=1;
  CALL SVCS1 (PROC_NAME,P,RTN_CODE);
  IF RTN_CODE ^=0 THEN PUT SKIP LIST ('ERROR IN SVC SEND TO ',
    PROC_NAME);

  CALL WAIT (1);
  TP=THISVP->VP.WAIT.MSG->MSG.PTR; /* CALLED PROC RETURNS */

/***** TIMING REPORT OPTION *****/
IF FTIME THEN DO;
  BEGTIME=RTIMER-BEGTIME;
  ETIME(ENUM)=ETIME(ENUM)+BEGTIME; /* ACCUMULATE TOTAL TIME */
  ECOUNT(ENUM)=ECOUNT(ENUM)+1; /* ACCUMULATE COUNT */
  ELO(ENUM)=ELO(ENUM)+LOTIME; /* ACCUMULATE LOWER LEVEL PROC TIME */
  FREE LOTIME; IF ALLOCATION(LOTIME) ^= 0 THEN LOTIME=LOTIME+BEGTIME;
END;

```

FILE: TCALL PLIDPT A1 VM/SP CONVERSATIONAL MONITOR SYSTEM

```

/*****
/***** TCALL TRACE *****/
IF FTCALL (1) THEN PUT EDIT ('*** ' PROC_NAME, ' RETURNS') (A,A);
/*****
/*****

FREE THISVP->VP.WAIT.MSG->MSG;
RETURN;
END /*TCALL*/;
TCA00560
TCA00570
TCA00580
TCA00590
TCA00600
TCA00610
TCA00620
TCA00630
TCA00640

```

VM/SP CONVERSATIONAL MONITOR SYSTEM

FILE: 1.(N) PLOIOT A1

```

TRTN: PROC(PROC_ADDR, RTP);
/**** A SERVICE ROUTINE USED BY A T-PROC WHEN IT FINISHES PROCESSING
**** IT ADDS A MSG TOKEN TO A RETURN ARGUMENT LIST AND CALLS SVCS2
*** AND THEN FINISHES ITSELF ****/
/*** PROC_ADDR CONTAINS THE ADDR OF THE CALLING PROCEDURE TO RETURN
*** THE RETURN MSG TO; RTP POINTS TO THE RETURN ARGUMENT LIST ***/

XINCLUDE USERS.VPX.EVCS.FDEBUG;
DCL (P.RTP) PTR, RTN_CODE FIXED BIN;
DCL 1 T BASED (P),
    2 LEN FIXED BIN (15);

ALLOCATE MSG;
MSG.PTR=RTP; /*CHAIN TO ROOT PTR OF ARG LIST*/
MSG.RTN_ADDR.LEVEL=THISVP->VP.LEVEL; /*RTN_ADDR NOT IMPORTANT*/
MSG.RTN_ADDR.VPID=THISVP->VP.VPID;
MSG.RTN_ADDR.BOXID=1; /*ALWAYS USE BOX 1*/
CALL SVCS2 (PROC_ADDR,P.RTN_CODE);
CALL FINISH; /*COMMIT SUICIDE*/
END /*TRTN*/;

XINCLUDE HEX;

```

TRT00010
 TRT00020
 TRT00030
 TRT00040
 TRT00050
 TRT00060
 TRT00070
 TRT00080
 TRT00090
 TRT00100
 TRT00110
 TRT00120
 TRT00130
 TRT00140
 TRT00150
 TRT00160
 TRT00170
 TRT00180
 TRT00190
 TRT00200
 TRT00210
 TRT00220
 TRT00230

FILE: UPCT PLIOPT A1 VM/SP CONVERSATIONAL MONITOR SYSTEM

```

UPCT: PROC(PSETID,PINFO,RTN_CODE):
/* A SUBROUTINE WHICH ACCEPTS THE PSETID AND A DECODED PINFO
AND REPLACES THE PCAT ENTRY */
XINCLUDE DPCAT,FOEBUG,DBEU;
XINCLUDE ESUB1;
DCL P PTR, ADDR BUILTIN, RTN_CODE FIXED BIN;
DCL PSETID BIT (32);
DCL BIT_STRING BIT(144) BASED;

CALL RET1 (PSETID,P,RTN_CODE);
BEU_DATA=ADDR(PINFO)->BIT_STRING;
CALL REP1 (PSETID,P,RTN_CODE);
FREE BEU;
RTN_CODE=0;

RETURN;
END;
UPC00010
UPC00020
UPC00030
UPC00040
UPC00050
UPC00060
UPC00070
UPC00080
UPC00090
UPC00100
UPC00110
UPC00120
UPC00130
UPC00140
UPC00150
UPC00160
UPC00170
UPC00180

```

```

UPDE PROC;

/*T-PROC TO PROCESS UPDE **/
XINCLUDE AUPDE,FDEBUG,SERVICE,AUPDN,ARETE;
XINCLUDE DECAT;
XINCLUDE EGECT,EGACT,EVTPE;
XINCLUDE EALLA; /* USES S-PROC ALLA TO OBTAIN ALL ATTRS */

DCL ANAMES(25) CHAR (8);
DCL (N,NODE) FIXED BIN;
DCL (P,TP,RP,LP,TP1,RTN,EP,EP1)PTR, NULL BUILTIN;
DCL (RTN.CODE,1) FIXED BIN, NUM FIXED BIN (31);
DCL BDATA BIT (320), CDATE CHAR (40), DLEN FIXED BIN;
DCL TO BIT FIXED BIN INIT (1), TO_CHAR, FIXED BIN INIT (2);
DCL TO BIT CHK FIXED BIN INIT (3);
DCL (ENAME,ANAME,ENAME1) CHAR (8);
DCL 1 MAINFO (25), /*AN ARRAY OF AINFO TO STORE RETRIEVED ACAT ENT**/
      2 BSETID BIT (32), /*BSETID OF THIS ATTRIBUTE**/
      2 PSETID BIT (32), /*PSETID OF TARGET NODE**/
      2 ENAME CHAR (8), /*IF TARGET IS A ENTITY SET**/
      2 MAX FIXED BIN (31),
      2 MIN FIXED BIN (31),
      2 MLEN FIXED BIN,
      2 FUNC CHAR(1),
      2 TYPE CHAR (1),
      2 VTYPE CHAR (1);
DCL ROOT_OP FIXED BIN;

/*BEGIN PROCESSING**/
CALL TBEG (PROC_ADDR,P);
ALLOCATE UPDE_RTN SET (RTP);
RTP->UPDE_RTN_CODE=0; /*INIT TO O.K.**/
ENAME=UPDE_ARG.NAME; ROOT_OP=UPDE_ARG.OP; /* REMEMBER ROOT OP */
MAINFO.ENAME(1)=ENAME; /*INIT AINFO ARRAY'S FIRST TOKEN*/

/*ALLOCATE ROOT NODE**/
CALL GECT (ENAME,EINFO.RTN_CODE); /*GECT GETS ENTITY CAT ENTRY */
IF RTN_CODE ^= 0
THEN DO;
  RTP->UPDE_RTN_CODE = 100; /*100 IS NO SUCH ENTITY**/
  CALL TRTN (PROC_ADDR,RTP); /***RETURNS**/
  RETURN;
END;

ALLOCATE UPDN_ARG SET (TP);
RP,LP=TP;
TP->UPDN_ARG.OP=UPDE_ARG.OP;
TP->UPDN_ARG.PSETID=EINFO;

/*FOLLOW UPDE TREE **/
DO P=UPDE_ARG.PTR REPEAT P->UPDE_ARG.PTR WHILE (P^=NULL);
  ALLOCATE UPON_ARG SET (TP); /* ALLOCATE UPON TOKENS */
  LP->UPON_ARG.PTR=TP; LP=TP;
  TP->UPON_ARG=UPDE_ARG, BY NAME;
  TP->UPON_ARG.CBTP=RP->UPON_ARG.CBTP; /*FIX UP CBTP*/

```

FILE: UPDE PLOIPT A1 VM/SP CONVERSATIONAL MONITOR SYSTEM

```

TP->UPDN_ARG_PTR = NULL;
NODE=UPDE_ARG.NODE;
ENAME1=MAININFO.ENAME(UPDE_ARG.PARENT);
ANAME=UPDE_ARG.NAME;
CALL GACT (ANAME, ENAME1, AINFO, RTN_CODE); /*GACT GETS ATTR CAT ENT*/
IF RTN_CODE=0
THEN DO;
  RTP->UPDE_ARG.RTN_CODE=NODE+100; /*100+ IS NO SUCH ATTRIBUTE***/
  CALL SVC3 (RTP); /*FREE UP UPDN ALLOCATION UP TO NOW*/
  CALL TRTN (PROC_ADDR, RTP); /*RETURNS**/
  RETURN;
END;
MAININFO(NODE)=AINFO; /*ENTER INTO AINFO ARRAY*/

/*TRANSFORM UPDE OP CODE INTO PROPER UPDN OP CODE*/
IF AINFO.TYPE='V' & UPDE_ARG.PARENT = 1
THEN DO;
  SELECT(UPDE_ARG.OP);
  WHEN (UPDEOP.IST) TP->UPDN_ARG.OP=UPDNOP.CI;
  WHEN (UPDEOP.DEL) TP->UPDN_ARG.OP=UPDNOP.DO;
  WHEN (UPDEOP.REP) TP->UPDN_ARG.OP=UPDNOP.REPP;
  OTHERWISE;
  END /*SELECT*/;
END /* IF V */;
IF UPDE_ARG.PARENT ^= 1 THEN /* FOR ALL ATTR'S NOT DIRECT Y
  RELATED TO ROOT NODE */
  TP->UPDN_ARG.OP=UPDNOP.ID;

/*VERIFY UPDATE DATA ***/
IF AINFO.TYPE='V' & UPDE_ARG.DLEN=0
THEN DO; /*WHEN THERE IS VAL*/
  /*CHECK FOR UNIQUENESS OF KEY*/
  IF AINFO.FUNC='K'
  THEN DO;
    SELECT (TP->UPDN_ARG.OP);
    WHEN(UPDNOP.CI,UPDNOP.REPP) DO;
      ALLOCATE RETE_ARG SET (EP);
      EP->RETE_ARG.NAME=ENAME; /*IN CURRENT VERSION
        PARENT MUST BE ROOT NODE*/
      EP->RETE_ARG.GET = RETEGET.ANY;
      ALLOCATE RETE_ARG SET(EP1);
      EP->RETE_ARG_PTR=EP1;
      EP1->RETE_ARG.NAME=UPDE_ARG.NAME;
      EP1->RETE_ARG.PARENT=1;
      EP1->RETE_ARG.NODE=2;
      EP1->RETE_ARG.N=1; /*1 PRED*/
      EP1->RETE_ARG.PRED.CDATA=UPDE_ARG.CDATA; /*COPY CDATA*/
      EP1->RETE_ARG.PRED.DLEN=UPDE_ARG.DLEN;
      EP1->RETE_ARG.PRED.OP=RETEOP.EQ;
      CALL TCALL ('RETE', 1, EP, EP1);
      IF EP1->RETE_ARG.RTN_CODE=0 | EP1->RETE_ARG.RTN.N=0
      THEN DO; /*PROBLEM DATA, KEY VIOLATION*/
        IF FDEBUG.LEV1

```

```

THEN PUT SKIP LIST ('(UPDE) KEY VIOLATION');
RTP->UPDE_RTN_CODE=300+UPDE_ARG.NODE;
/*** RTN_CODE 300+ IS VIOLATION OF KEY CONSTRAINT*/
CALL SVC3 (EP); CALL SVC3 (EPI); CALL SVC3 (RP);
CALL TRTN (PROC_ADDR,RTP);
RETURN;
END /*RTN_CODE CHK*/;

END /*IST OR REP*/;
OTHERWISE;
END /*SELECT*/;

END /*FUNC='K'*/;

/*CHECK DATA TYPE AND NUMERIC LIMITS*/
COATA=UPDE_ARG.COATA;
DLEN=UPDE_ARG.DLEN;
CALL VTPE (AINFO,TO_BIT_CHK,BDATA,CDATA,DLEN,RTN_CODE);
IF RTN_CODE<=0
THEN DO; /*PROBLEMS*/
RTP->UPDE_RTN_CODE=NODE; /*1+ IS ILLEGAL NODE DATA*/
CALL SVC3(RP); /*FREE UP UPDN ALLOCATION UP TO NOW*/
CALL TRTN (PROC_ADDR, RTP); /*RETURNS*/
RETURN;
END /*RTN_CODE CHK*/;
END /*VERIFY DATA*/;

/*REPLACE ANAME WITH PSETID AND BSETID AND FIX UP DATA*/
TP->UPDN_ARG.PSETID=AINFO.PSETID;
TP->UPDN_ARG.BSETID=AINFO.BSETID;
IF AINFO.TYPE='V' & UPDE_ARG.DLEN<=0
THEN DO;
TP->UPDN_ARG.DLEN=DLEN;
TP->UPDN_ARG.DATA=BDATA;
END;
END /*DO P=UPDE_ARG.PTR...*/;

/* FIX UP THE DELETE ENTITY (DEE) OPERATION TO INCLUDE DELETION
OF ALL ITS ATTRIBUTES */

IF ROOT_OP = UPDEOP.DEE
THEN DO;
CALL ALLA (ENAME, ANAMES, N, RTN_CODE);
NODE = LP->UPDN_ARG.NODE; /* LP POINTS TO THE LAST TOKEN
IN THE UPDN_ARG CHAIN ALLCATED BY CALLER */

DO I = 1 TO N;
CALL GACT (ANAMES (I), ENAME, AINFO, RTN_CODE);
ALLOCATE UPDN_ARG;
LP->UPDN_ARG.PTR= P; LP = P;
UPDN_ARG.PSETID = AINFO.PSETID;
UPDN_ARG.BSETID = AINFO.BSETID;
UPDN_ARG.DLEN = 0;
UPDN_ARG.PARENT = 1; /* ALWAYS ROOT NODE */

```

VM/SP CONVERSATIONAL MONITOR SYSTEM

FILE: UPDE PLIOPT A1

```

UPDN ARG.NODE = NODE + 1;
IF AINFD.TYPE = 'V'
  THEN UPDN_ARG.OP = UPDNOP.DD;
  ELSE UPDN_ARG.OP=UPDNOP.DEL;
END /* DO I */;
END /* IF DEE */;

/*CALL UP UPON */
CALL TCALL ('UPDN',1,RP,TP);

/*CHECK RETURNS*/
I-TP->UPDN_RTN_CODE;
IF I=0 THEN
  RTP->UPDE_RTN_CODE=I+200; /*200+ IS RTN CODE FROM UPDN*/
/*RETURNS*/
CALL SVC3 (RP);
CALL SVC3 (TP);
CALL TRTN (PROC_ADDR, RTP);
RETURN;

END /*UPDE*/;
UPDO1660
UPDO1670
UPDO1680
UPDO1690
UPDO1700
UPDO1710
UPDO1720
UPDO1730
UPDO1740
UPDO1750
UPDO1760
UPDO1770
UPDO1780
UPDO1790
UPDO1800
UPDO1810
UPDO1820
UPDO1830
UPDO1840
UPDO1850
UPDO1860
UPDO1870

```

```

UPDN:PROC:
/.....
*      MODULE      DESCRIPTION
*      .....
/.....
PURPOSE:
THIS MODULE IS THE T-PROC AT THE N-ARY LEVEL WHICH
PERFORMS UPDATES IN THE DATABASE; IT ACCEPTS UPDATE
REQUESTS IN A TREE FORM (UPDN TREE), RESOLVES PREDICATES
ON OCCURENCES TO BE UPDATED, AND CALLS SUBROUTINES TO
PERFORM CRT,DEE AND MOD TYPES OF UPDATES.
.....
METHOD:
1. MAKE A PASS THROUGH THE UPDN TREE TO FILL UP A TEMP
   DATA STRUCTURE WHICH IS USED TO REMEMBER ALL NODES
   WHICH HAVE ID CHILD NODES;
2. USE TEMP. WORKING FROM BOTTOM OF THE TREE TO TOP,
   RESOLVE ALL PREDICATES (I.E. ID OPS) ON THE OCCURENCES
   OF DATA ELEMENTS TO BE UPDATED; REDUCE THE ORIGINAL
   UPDN TREE TO A 2-LEVEL UPDN TREE WITHOUT ID OPS;
3. PASS THE REDUCED TREE TO SUBROUTINES CRTN,DELN OR
   MODN DEPENDING ON UPDN.OP AT THE ROOT NODE.
.....
INPUT PARAMETERS:
AS INDICATED IN UPDN_ARG;
.....
OUTPUT PARAMETERS:
AS INDICATED IN UPDN_RTN;
.....
CALLING ARGUMENTS:
XINCLUDE AUPDN;
/..... CALLS PROCEDURES:
..... INTER-LEVEL T-PROC: NONE;
..... INTRA-LEVEL T-PROC: RETN;
XINCLUDE ARETN;
/..... INTRA-LEVEL S-PROC: MODN,CRTN,DELN;
XINCLUDE EMODN,ECRTN,EDELN;
/..... CONTROL STRUCTURE, PANEL MANAGER AND DEBUGGING FACILITIES: */
XINCLUDE SERVICE,FDEBUG;
/.....
DCL (P.TP,RP,RTN,LP,MP) PTR;
DCL RTN_CODE FIXED BIN;
DCL NULL_BUILTIN;
DCL 1 TEMP (20). /*TEMP STRUCTURE, PARENT NODE# CAN BE UP TO 20*/
      2 K FIXED BIN,
      2 IDN (5) FIXED BIN,
      2 IDP (5) PTR;
DCL(I,J,N) FIXED BIN;
DCL OFF_NODE(25) BIT(1) INIT ((25)('O'B));
DCL NPTR(25) PTR;
/*BEGIN PROCESSING*/
CALL TBEG (PROC_ADDR,MP);

```

```

UPD000560
UPD000570
UPD000580
UPD000590
UPD000600
UPD000610
UPD000620
UPD000630
UPD000640
UPD000650
UPD000660
UPD000670
UPD000680
UPD000690
UPD000700
UPD000710
UPD000720
UPD000730
UPD000740
UPD000750
UPD000760
UPD000770
UPD000780
UPD000790
UPD000800
UPD000810
UPD000820
UPD000830
UPD000840
UPD000850
UPD000860
UPD000870
UPD000880
UPD000890
UPD000900
UPD000910
UPD000920
UPD000930
UPD000940
UPD000950
UPD000960
UPD000970
UPD000980
UPD000990
UPD01000
UPD01010
UPD01020
UPD01030
UPD01040
UPD01050
UPD01060
UPD01070
UPD01080
UPD01090
UPD01100

ALLOCATE UPDN_RTN SET (RTP);
LP=NULL;

/*TREE REDUCTION--REMOVE ALL 'ID' NODES*/
DO N = 1 TO 20; /*INITIALIZE TEMP STRUCTURE*/
  TEMP.K(N) = 0;
  DO J = 1 TO 5;
    TEMP.IDN(N,J) = 0;
    TEMP.IDP(N,J) = NULL;
  END;
END;

/***** UPDN REQUEST TRACE *****/
DCL FFFN BIT (1) EXT;
IF FFFN THEN DO;
  PUT SKIP EDIT('NODE *PSETID* PARENT OP DLEN *BSETID* DATA')(A);

  DO P=MP REPEAT P->UPDN_ARG.PTR WHILE (P^=NULL);
  PUT SKIP EDIT(UPDN_ARG.NODE.HEX4(ADDR(UPDN_ARG.PSETID)->BP),
    UPDN_ARG.PARENT.UPDN_ARG.OP,
    UPDN_ARG.DLEN, ' ')(F(4).X(1).A(8).F(4).F(6).F(4).A);
  IF UPDN_ARG.PARENT^=0 THEN
    PUT EDIT(HEX4(ADDR(UPDN_ARG.BSETID)->BP), ' ' )
      (A(8).A);
  ELSE PUT EDIT(' ')(A);
  DO I1=1 TO UPDN_ARG.DLEN/4; DCL I1 FIXED BIN;
    PUT EDIT(HEX4(ADDR(UPDN_ARG.DATA)->PAA(I1)))(A(8));
  %INCLUDE HEX; DCL BP PTR BASED;
  END;
END;
DCL PAA(10) PTR BASED;
END; END;

/***** FIRST MAKE A PASS THROUGH UPDN ARG TREE TO FILL UP TEMP STRUCTURE */
DO P=MP REPEAT P->UPDN_ARG.PTR WHILE (P^=NULL);

  NPTR(UPDN_ARG.NODE) = P; /* SAVE POINTER OF NODE TOKEN */
  IF UPDN_ARG.OP = UPDNOP.ID
  THEN DO;
    N = UPDN_ARG.PARENT; /* INDEX TEMP BY PARENT NODE # */
    IF N > 20 /* TEMP ARRAY OVERFLOW */
    THEN DO;
      RTP->UPDN_RTN.RTN_CODE=1;
      IF FDEBUG.LEV2 THEN PUT SKIP LIST ('(UPDN) ERR RTN '
        , RTP->UPDN_RTN.RTN_CODE);
      GO TO RTN;
    END;
  ELSE;
    TEMP.K(N) = TEMP.K(N) + 1;
    IF TEMP.K(N) > 5
    THEN DO; /* TEMP IDN OVERFLOW */
      RTP->UPDN_RTN.RTN_CODE=2;
      IF FDEBUG.LEV2 THEN PUT SKIP LIST ('(UPDN) ERR RTN '
        , RTP->UPDN_RTN.RTN_CODE);
      GO TO RTN;
    END;
  END;
END;

```

```

UPD01110
UPD01120
UPD01130
UPD01140
UPD01150
UPD01160
UPD01170
UPD01180
UPD01190
UPD01200
UPD01210
UPD01220
UPD01230
UPD01240
UPD01250
UPD01260
UPD01270
UPD01280
UPD01290
UPD01300
UPD01310
UPD01320
UPD01330
UPD01340
UPD01350
UPD01360
UPD01370
UPD01380
UPD01390
UPD01400
UPD01410
UPD01420
UPD01430
UPD01440
UPD01450
UPD01460
UPD01470
UPD01480
UPD01490
UPD01500
UPD01510
UPD01520
UPD01530
UPD01540
UPD01550
UPD01560
UPD01570
UPD01580
UPD01590
UPD01600
UPD01610
UPD01620
UPD01630
UPD01640
UPD01650

END;
ELSE;
TEMP.IDN(N,TEMP.K(N)) = UPDN_ARG.NODE;
TEMP.IDP(N,TEMP.K(N)) = P;
IF LP=NULL
THEN DO; /* ILLEGAL ROOT */
RTP->UPDN_RTN.RTN_CODE=3;
IF FDEBUG.LEV2 THEN PUT SKIP LIST ('(UPDN) ERR RTN '
, RTP->UPDN_RTN.RTN_CODE);
GO TO RTN;
END;
ELSE;
OFF_NODE(UPDN_ARG.NODE) = '1'B; /* MARK NODES OFF TREE */
END /* IF UPDN_ARG.OP */;
ELSE LP=P;
END /* DO P = P */;

/*FOR EACH PARENT NODE IN TEMP CALL RETN TO 'PIN POINT' THE
OCCURRENCE OF THE NODE*****/
DO N = 20 TO 1 BY -1; /*WORKS FROM THE END NODES UP*/
IF TEMP.K(N) ^= 0
THEN DO;
I = 0; /*NO PRED FOR ROOT NODE*/ ALLOCATE RETN_ARG;
TP,RP = P;
RETN_ARG.GET = RETNGET.ALL;
RETN_ARG.PSETID = NPTR(N)->UPDN_ARG.PSETID;
DO J = 1 TO TEMP.K(N); /*FOR EACH OF ITS CHILD NODE*/
I = 1; ALLOCATE RETN_ARG;
TP->RETN_ARG.PTR = P; TP = P;
RETN_ARG.NODE = J + 1;
RETN_ARG.PSETID = IDP(N,J)->UPDN_ARG.PSETID;
RETN_ARG.BSETID = IDP(N,J)->UPDN_ARG.BSETID;
RETN_ARG.PRED.DLEN(1) = IDP(N,J)->UPDN_ARG.DLEN;
RETN_ARG.PRED.DATA(1) = IDP(N,J)->UPDN_ARG.DATA;
RETN_ARG.PARENT = 1;
RETN_ARG.N = 1;
RETN_ARG.GET = RETNGET.NO;
RETN_ARG.PRED.CN(1) = 1;
RETN_ARG.PRED.OP(1) = RETNOP.EQ;
END /* J */;

/***** CALL UP RETN *****/
CALL TCALL ('RETN',1,RP,TP);
CALL SVC3(RP); /*FREE CALLING ARG LIST*/
IF TP->RETN_RTN.RTN_CODE ^= 0 THEN DO;
/*****RTN CODE 4 IS RETN CALL ERROR IN RETN *****/
RTP->UPDN_RTN.RTN_CODE=4;
IF FDEBUG.LEV2 THEN PUT SKIP LIST ('(UPDN) ERR RTN '
, RTP->UPDN_RTN.RTN_CODE);
CALL SVC3 (TP); /*FREE RTN MSG CHAIN*/
GO TO RTN;
END;

/*CHECK NUMBER OF ELEMENTS RETURNED****/
IF TP->RETN_RTN.N ^= 1 THEN DO;
/*****RTN CODE 10+N REPRESENTS THE NODE NUMBER OF A *****/
***** PARENT NODE WHICH HAS AN ILLEGAL IDENTIFICATION TREE*/

```

FILE: UPDN PLOTPT A1 VM/SP CONVERSATIONAL MONITOR SYSTEM

```

RTP->UPDN_RTN.RTN_CODE=N*10;
IF FDEBUG.LEV2 THEN PUT SKIP LIST ('(UPDN) ERR RTN '
      RTP->UPDN_RTN.RTN_CODE);
CALL SVC3(TP); /*FREE RTN MSG CHAIN*/
GO TO RTN;
END;

P=TP->RETN_RTN.PTR; /*GET RETN DATA*/
NPTR(N)->UPDN_ARG.DLEN=P->RETN_RTN1.DLEN;
NPTR(N)->UPDN_ARG.DATA=P->RETN_RTN1.DATA;
CALL SVC3(TP); /*FREE RTN MSG LIST*/
END /*IF TEMP.K*/;
END /***N*/;

/*IN CURRENT VERSION BY NOW THE UPDN TREE HAS BEEN REDUCED TO A
2-LEVEL TREE WITHOUT ID NODES *****/
/*PERFORM UPDATES****/ P=MP;
SELECT (UPDN_ARG.OP);
WHEN (UPDNOP.CRT) CALL CRTN (P.OFF_NODE,RTN_CODE);
WHEN (UPDNOP.DEE) CALL DELN (P.OFF_NODE,RTN_CODE);
WHEN (UPDNOP.MOD) CALL MODN (P.OFF_NODE,RTN_CODE);
OTHERWISE DO ; /*RTN CODE 31 IS ILLEGAL ROOT NODE OP******/
      RTP->UPDN_RTN.RTN_CODE=31;
      IF FDEBUG.LEV2 THEN PUT SKIP LIST ('(UPDN) ERR RTN '
            RTP->UPDN_RTN.RTN_CODE);
      GO TO RTN;
      END;
END /*SELECT**/;

/****CHECK SUBROUTINE RTN CODE *****/
IF RTN_CODE=-O THEN DO; /*RTN CODE 40+ IS FROM SUBR CALL***/
      RTP->UPDN_RTN.RTN_CODE=RTN_CODE+40;
      IF FDEBUG.LEV2 THEN PUT SKIP LIST ('(UPDN) ERR RTN '
            RTP->UPDN_RTN.RTN_CODE);
      END;

/*SET UP RTN MSG*****/
RTN: CALL TRTN(PROC_ADDR, RTP);

END /*UPDN*/;

```

UPDO1660
 UPDO1670
 UPDO1680
 UPDO1690
 UPDO1700
 UPDO1710
 UPDO1720
 UPDO1730
 UPDO1740
 UPDO1750
 UPDO1760
 UPDO1770
 UPDO1780
 UPDO1790
 UPDO1800
 UPDO1810
 UPDO1820
 UPDO1830
 UPDO1840
 UPDO1850
 UPDO1860
 UPDO1870
 UPDO1880
 UPDO1890
 UPDO1900
 UPDO1910
 UPDO1920
 UPDO1930
 UPDO1940
 UPDO1950
 UPDO1960
 UPDO1970
 UPDO1980
 UPDO1990
 UPDO2000
 UPDO2010
 UPDO2020
 UPDO2030
 UPDO2040

```

UPE1:PROC (OPE,T,OP,RTN_CODE);
  /** SUBROUTINE TO GET DATA FROM USER AND BY MAKING USE OF
  ***** THE T.OPE,OP DATA STRUCTURES PASSED TO IT FROM THE CALLER
  ***** ABLE TO GENERATE PROPER CALL TO UPDE AT THE NEXT LEVEL **/

  XINCLUDE AUPDE.SERVICE.ELEX;
  DCL 1 T, /*DATA STRUCTURE OF NAMES */
    2 ENAME CHAR(8),
    2 TN FIXED BIN, /*NUMBER OF ATTR*/
    2 T1 (25),
    3 PARENT FIXED BIN,
    3 ANAME CHAR (8);

  DCL RTN_CODE FIXED BIN;
  DCL (OP(25), OPE) CHAR(1);
  DCL (P,RP,LP,TP) PTR, NULL BUILTIN;
  DCL CODE FIXED BIN, K FIXED BIN;
  DCL NA(25) BIT(1);
  DCL 1 TOKEN (25);
    2 D CHAR(40);
    2 L FIXED BIN; /*LENGTH OF TOKEN*/
    2 N FIXED BIN . /*SERVE AS TOKEN COUNT*/ I FIXED BIN;

  /*INITIALIZE NA STRUCTURE*/
  DO I=1 TO 25;
    NA(I)='O'B;
  END;

  DO I=1 TO TN;
    IF T1.PARENT(I)~=0 THEN NA(T1.PARENT(I))='1'B;
  END;

  /*BEGIN PROCESSING*/
  DO WHILE ('1'B) /*DO FOREVER*/

    /*ALLOCATE ROOT NODE*/
    ALLOCATE UPDE_ARG;
    LP,RP=P;
    UPDE_ARG.NAME=T.ENAME;
    SELECT (OPE);
      WHEN('M') UPDE_ARG.OP=UPDEOP.MOD;
      WHEN('C') UPDE_ARG.OP=UPDEOP.CRT;
      WHEN('D') UPDE_ARG.OP=UPDEOP.DEE;
    OTHERWISE DO;
      RTN_CODE=1; /****** 1 -- ILLEGAL ESET OP *****/
      PUT SKIP LIST ('ILLEGAL SEMANTICS OF MODIFICATION STATEMENT');
      CALL SVC3(RP);
      RETURN;
    END;
  END /*SELECT*/;
  K=0; /*INIT DATA COUNT*/

  A:
  CALL LEX (N,TOKEN,O,RTN_CODE);
  IF RTN_CODE~=0
    THEN DO;

```

FILE: UPE1 PLIOPT A1 VN/SP CONVERSATIONAL MONITOR SYSTEM

```

PUT SKIP LIST ('ILLEGAL SYNTAX, PLEASE RE-ENTER LINE');
GO TO A; END;

IF N=0 THEN LEAVE; /*NO MORE DATA*/

/*FILL UP UPDE REQUEST*/
DO I=1 TO TN;
  ALLOCATE UPDE_ARG;
  LP->UPDE_ARG.PTR=P; LP=P;
  UPDE_ARG.NODE=I+1; /*ASSIGN NODE NUMBER*/
  UPDE_ARG.PARENT=T1.PARENT(I)+1;
  UPDE_ARG.NAME=T1.ANAME(I);
  SELECT (OP(I));
    WHEN ('I') UPDE_ARG.OP=UPDEP.ID;
    WHEN ('N') UPDE_ARG.OP=UPDEP.IST;
    WHEN ('D') UPDE_ARG.OP=UPDEP.DEL;
    WHEN ('R') UPDE_ARG.OP=UPDEP.REP;
    OTHERWISE DO;
      RTN_CODE=2;
      PUT SKIP LIST ('ILLEGAL SEMANTICS OF MODIFICATION, STMT. ');
      CALL SVC3(RP);
      RETURN;
    END;
  END /*SELECT*/;

IF NA(I)='O'B THEN DO; /*FILL IN DATA*/
  K=K+1;
  IF K>N THEN DO;
    PUT SKIP LIST ('INSUFFICIENT DATA: REENTER LINE');
    CALL SVC3 (RP);
    GO TO END;
  END;
  UPDE_ARG.CDATA=TOKEN.D(K);
  UPDE_ARG.DLEN=TOKEN.L(K);
  END;
END /*DO I*/;

/*ALL NODES ARE ALLOCATED*/
CALL TCALL ('UPDE',1,RP,TP);
CALL SVC3 (RP); /*FREE ARG LIST*/
CODE=TP->UPDE_RTN_CODE;
IF CODE=0
  THEN DO;
    /* CHECK UPDE RTN CODE: IGNORES 100-200 RANGE */
    IF CODE > 300
      THEN PUT SKIP LIST ('KEY VIOLATION');
    ELSE IF CODE > 200
      THEN PUT SKIP EDIT ('RETN RTN_CODE ',CODE-200)(A,F(3));
    ELSE PUT SKIP EDIT ('ILLEGAL DATA FOR ATTRIBUTE ',
      T.T1.ANAME(CODE-1))(A,A);
    PUT SKIP LIST ('DATA ENTERED IGNORED');
    END /* RTN CODE CHECK */;

CALL SVC3 (TP); /*FREE RETURN LIST*/
END;
END /*DO WHILE*/;

```

FILE: UPE1

PL10PT A1

VM/SP CONVERSATIONAL MONITOR SYSTEM

PAGE 003

RETURN:

END /*UPE1*/:

DUM01110
DUM01120

FILE: USER PLOIOT A1 VM/SP CONVERSATIONAL MONITOR SYSTEM

```

USER:PROC;
/*****
*          MODULE      DESCRIPTION
*          ****
*/
***** PURPOSE:
***** A T-PROC FOR THE USER TO SELECT SUBSYSTEMS AFTER LOGIN IN
***** CALLS PROCEDURES:
***** DBA, BNV, RELV:  INTRA-LEVEL S-PROC REPRESENTING DBA MODE,
***** BASE DATA MODEL VIEW, AND RELATIONAL DATA MODEL VIEW.
***** PRESENTLY, THE LATTER TWO ARE NOT AVAILABLE.
*****
XINCLUDE EDBA.SERVICE; /* ENTRY DCLS OF SUBSYSTEMS*/
DCL NAME CHAR(7),P PTR;

/*BEGIN PROCESSING*/
CALL TBEG(PROC_ADDR,P); /* JUST FOR CONVENTION */

DO WHILE ('1'B);
  PUT SKIP LIST ('SUBSYSTEMS: DBA, BV(BASE VIEW) OR RV(RELATIONAL VIEW)
  ?');
  GET EDIT (NAME) (A(7));
  IF NAME=(7)' ' THEN LEAVE;
  SELECT (NAME);
  WHEN ('DBA','D')CALL DBA;
  WHEN ('BV','B') CALL BNV;
  WHEN ('RV','R')CALL RELV;
  OTHERWISE PUT SKIP EDIT (NAME,' IS NOT A VALID COMMAND') (A,A);
END;

END;
/*IF NO SUBSYSTEM COMMAND IS GIVEN, THEN END SESSION*/
PUT SKIP LIST ('-- END OF USER SESSION--');

/*****
BNV: PROC;
PUT SKIP LIST ('BASE DATA MODEL VIEW SUBSYSTEM NOT AVAILABLE. ');
RETURN;
END;

RELV: PROC;
PUT SKIP LIST ('RELATIONAL DATA MODEL VIEW SUBSYSTEM NOT AVAILABLE. ');
RETURN;
END;

CALL TRTN (PROC_ADDR,P);
END /*USER*/;

```

FILE: VERIF PLIOPT A1 VM/SP CONVERSATIONAL MONITOR SYSTEM

```

VERIF: PROC(PINFO,P,VOP,BDATA,RTN_CODE,YES);
/*A S-PROC IN THE N-ARY LEVEL TO VERIFY AND CONVERT DATA ***/

XINCLUDE DPCAT,DBEU;
XINCLUDE EPCT,ESUB1;
DCL BDATA BIT (*) VAR ALIGNED,
VOP BIT(8) ALIGNED, RTN_CODE FIXED BIN, YES BIT(1) ALIGNED;
DCL PTYPE CHAR (1), BDATA1 BIT (320) VARYING;
DCL (NUM,NUM1) FIXED BIN (31);
DCL 1 OP ALIGNED, /* SAME AS RETNOP */
2 EQ BIT (8) INIT ('00000000'B),
2 GT BIT (8) INIT ('00000010'B),
2 LT BIT (8) INIT ('00000001'B);
DCL P PTR, NULL BUILTIN;
DCL UNSPEC BUILTIN;

/*BEGIN PROCESSING*/
YES='O'B; /*INIT TO NO*/
PTYPE=PINFO.PTYPE;
SELECT (PTYPE);

WHEN('N') DO; /*CONVERSION*/
/* PUT SKIP LIST ('(VERIF), WHEN N'); */
UNSPEC(NUM)=BDATA;
UNSPEC (NUM1)=BEU.DATA;
SELECT (VOP);

WHEN (OP.EQ) IF NUM1=NUM THEN YES='1'B;
WHEN(OP.GT) IF NUM1>NUM THEN YES='1'B;
WHEN (OP.LT) IF NUM1<NUM THEN YES='1'B;
OTHERWISE DO;
RTN_CODE=3; /****3 IS ILLEGAL OP******/
RETURN;
END;

END /*SELECT OP*/;
END /*WHEN N*/;

OTHERWISE DO; /*NO CONVERSION*/
BDATA1=BEU.DATA;
SELECT (VOP);
WHEN (OP.EQ) IF BDATA1=BDATA THEN YES='1'B;
WHEN (OP.GT) IF BDATA1>BDATA THEN YES='1'B;
WHEN (OP.LT) IF BDATA1<BDATA THEN YES='1'B;
OTHERWISE DO;
RTN_CODE=3; /****3 IS ILLEGAL OP*/
RETURN;
END;

END /*SELECT OP*/;
END /*OTHERWISE*/;
END /*SELECT PTYPE*/;
END/*VERIF*/;

```

```

VIN00010
VIN00020
VIN00030
VIN00040
VIN00050
VIN00060
VIN00070
VIN00080
VIN00090
VIN00100
VIN00110
VIN00120
VIN00130
VIN00140
VIN00150
VIN00160
VIN00170
VIN00180
VIN00190
VIN00200
VIN00210
VIN00220
VIN00230
VIN00240
VIN00250
VIN00260
VIN00270
VIN00280
VIN00290
VIN00300
VIN00310
VIN00320
VIN00330
VIN00340
VIN00350
VIN00360
VIN00370
VIN00380
VIN00390
VIN00400
VIN00410
VIN00420
VIN00430
VIN00440
VIN00450
VIN00460
VIN00470
VIN00480
VIN00490
VIN00500
VIN00510
VIN00520
VIN00530
VIN00540
VIN00550

VINIT :PROC;
/*****
*
*      MODULE      DESCRIPTION
*/*****/
PURPOSE: INITIALIZES THE ENTITY LEVEL EITHER FROM AN
EXISTING FILE OR FROM SCRATCH.
METHOD:
1. DETERMINE WHETHER IT IS A FILE INIT OR A NEW INIT;
2. IF NEW INIT, CALL DEFP (THROUGH DFP1) AND ODFB (THROUGH
   OFB1) TO CREATE PSETS AND BSETS USED IN STORING CATALOGUE
   ENTRIES AT THIS LEVEL; THE PSETID'S AND BSETID'S RETURNED
   ARE SAVED IN A STATIC STRUCTURE CALLED KEY, TO BE
   SHARED BY RELEVANT MODULES AT THIS LEVEL.
3. KEY IS INSERTED IN TO AN ANCHOR PSET BY CALLING UPDN;
4. PSETID OF ANCHOR PSET IS PASSED TO THE N-ARY LEVEL
   IN NINIT (OP-STK) CALL.
5. IF FILE INIT THEN' OBTAIN THE PSETID OF THE ANCHOR
   PSET FROM NINIT_RTN AND IN TURN RETRIEVE ALL KEYS.
INPUT PARAMETERS:
AS INDICATED IN VINIT_ARG;
OUTPUT PARAMETERS:
AS INDICATED IN VINIT_RTN;
CALLING ARGUMENTS:
%INCLUDE AVINI;
/***** CALLS PROCEDURES:
INTER-LEVEL T-PROC: NINIT,RTN,UPDN;
%INCLUDE ANINI,ARETN,AUPDN;
INTRA-LEVEL T-PROC: NONE;
/*****/
INTRA-LEVEL S-PROC: DFP1,DFB1;
%INCLUDE EDFP1,EDFB1;
/*****/ CONTROL STRUCTURE, PANEL MANAGER AND DEBUGGING FACILITIES:
%INCLUDE SERVICE,FDEBUG;
/*****/ EXTERNAL STATIC VARIABLES TO HOLD STATIC DATA: KEY
%INCLUDE DKEY;
DCCL (P,TP,RP,MP,RTP) PTR;
DCCL NAME CHAR(8), I FIXED BIN;
DCCL NULL BUILTIN;
DCCL RTN CODE FIXED BIN (15);
DCCL 1 PC(7), /DATA FOR DEF OF 7 PSETS IN CATALOGUE; SEQUENCE IS
    ESET,ENM,EINFO,ASET,ANM,AINFO,KEY*/
2 PTYPE CHAR (1) INIT ('X','C','B','X','C','B','B'),
2 PLEN FIXED BIN INIT (0,B,4,0,B,29,4),
2 TYPE CHAR(1) INIT ('E','V','V','E','V','V','V'),
2 PSETID BIT (32);
DCCL 1 BC(5), /DATA FOR DEF OF 5 BSETS IN CATALOGUE; SEQUENCE IS
    ENAME,EINFO,ESET,ENAME,AINFO*/
2 FUNC CHAR(1) INIT ('S','S','M','S','S'),

```

FILE: VINIT PLIOPT A1 VM/SP CONVERSATIONAL ACQUISITOR SYSTEM

```

2 PSETID(2) BIT(32).
2 RTN_CODE FIXED BIN.
2 BSETID BIT(32);
DCL AKEY BIT(32);

/*BEGIN PROCESSING*/
CALL TBEG (PROC_ADDR.MP);

    ALLOCATE VINIT_RTN SET (RTP);
    RTP->VINIT_RTN_CODE=0; /*INITIALIZE RTN CODE TO O.K.*/
    /*SET UP MSG TO CALL NINT */
    NAME=MP->VINIT_ARG.FNAME; /*GET FILE NAME*/
    ALLOCATE NINIT_ARG;
    NINIT_ARG.FNAME=NAME;
    NINIT_ARG.OP='I'; /*INITIALIZE NEXT LEVEL*/
    CALL TCALL ('NINIT', 1,P,TP);
    CALL SVC3 (P); /*FREE CALLING ARG*/

    IF TP->NINIT_RTN_CODE ^=0 THEN DO;
        RTP->VINIT_RTN_CODE=1; /*SET RTN CODE INVALID*/
        CALL SVC3(TP); /*FREE RTN MSG*/
        GO TO RTN; /*GO TO END*/
        END; /*END OF STANDARD MSG PROCESSING*/

/*PROCESS INITIALIZATION OF CURRENT LEVEL*/
IF NAME=(8) THEN /*NEW INIT*/
DO: /*NEW INIT: DEFINE CATALOGUES*/

    CALL SVC3 (TP); /*NINIT_RTN NO LONGER NEEDED*/
    DO I=1 TO 7; /*DEFINE 7 PSETS*/
        CALL DFP1(PC.PTYPE(I), PC.PLEN(I), PC.TYPE(I), RTN_CODE,
            PC.PSETID(I));
        IF RTN_CODE ^=0 THEN DO;
            RTP->VINIT_RTN_CODE =1;
            IF FDEBUG.LEV1 THEN PUT SKIP LIST ('VINIT: DFP1 ABEND');
            GO TO RTN;
        END;
        IF I=7 THEN AKEY=PC.PSETID(1);
        ELSE KEY(1) = PC.PSETID(1);
        END /*DEFINE 7 PSETS*/;

    /*SET PSETID FOR BC STRUCTURE*/
    BC.PSETID(1,1)=KEY(1);
    BC.PSETID(1,2)=KEY(2);
    BC.PSETID(2,1)=KEY(1);
    BC.PSETID(2,2)=KEY(3);
    BC.PSETID(3,1)=KEY(4);
    BC.PSETID(3,2)=KEY(1);
    BC.PSETID(4,1)=KEY(4);
    BC.PSETID(4,2)=KEY(5);
    BC.PSETID(5,1)=KEY(4);
    BC.PSETID(5,2)=KEY(6);
    DO I=1 TO 5; /*DEFINE 5 BSETS*/
        CALL DFB1(BC.FUNC(I),BC.PSETID(1,1),BC.PSETID(1,2),
            BC.RTN_CODE(1),BC.BSETID(1));

```

FILE: VNIT PLIOPY A1 VM/SP CONVERSATIONAL MONITOR SYSTEM

```

IF BC.RTN CODE (1) ^=0 THEN DO;
  RTP->VNIT.RTN.RTN_CODE=1;
  GO TO RTN;END;
KEY(1+6)=BC.BSETID(1);
END /*DEFINE 5 BSETS*/;

/*STORE 11 KEYS INTO THE ANCHOR PSET*/
DO I=1 TO 11;
  ALLOCATE UPDN_ARG;
  UPDN_ARG.PSETID=AKEY;
  UPDN_ARG.OP=UPDNOP.CRT;
  UPDN_ARG.DLEN=4;
  UPDN_ARG.DATA=KEY(I);
  CALL TCALL ('UPDN',1,P,TP);
END/*I*/;

CALL SVC3(P); /*FREE CALLING ARG LIST*/
IF TP->UPDN.RTN.RTN_CODE ^=0 THEN DO;
  RTP->VNIT.RTN.RTN_CODE=1; CALL SVC3(TP);
  GO TO RTN;
END;

CALL SVC3(TP); /*FREE RTN MSG CHAIN*/

/*CALL UP THE NEXT LEVEL TO STORE KEY TO THE ANCHOR SET*/
ALLOCATE NINIT_ARG;
NINIT_ARG.KEY=AKEY;
NINIT_ARG.OP='S'; /*ASKS NINIT TO STORE KEY*/
CALL TCALL ('NINIT',1,P,TP);

CALL SVC3(P); /*FREE CALLING ARG LIST*/
IF TP->NINIT.RTN.RTN_CODE ^=0 THEN DO;
  RTP->VNIT.RTN.RTN_CODE=1; CALL SVC3 (TP);
  GO TO RTN;
END;

CALL SVC3(TP); /*FREE RTN MSG LIST*/

/*** BOOTSTRAP CATALOGUES HERE *****/
CALL EBOT;/*EBOT IS A SUBROUTINE WHICH BOOTSTRAP CAT ENTRIES*/

END /*NEW INIT PROCESSING*/;

ELSE DO: /*FILE INIT*/
  /*OBTAIN ANCHOR KEY*/

  AKEY=TP->NINIT.RTN.KEY;
  CALL SVC3(TP);
  /*OBTAIN THE REST OF THE 11 KEYS*/
  ALLOCATE RETN_ARG;
  RETN_ARG.PSETID=AKEY;
  RETN_ARG.GET=RETNGET.ALL;

  CALL TCALL ('RETN',1,P,TP);
  CALL SVC3(P); /*FREE CALLING ARG*/

```

VM/SP CONVERSATIONAL M...ITOR SYSTEM

FILE: V...IT PLIOPT A1

```

IF TP->RETN_RTN_CODE ^=0 THEN DO;
  IF FDEBUG.LEV1 THEN PUT SKIP LIST ('INSIDE FILE INIT, JUST
RETURNED FROM RETN, AND RTN_CODE IS NOT O.K.'):
  RTP->VINIT_RTN_CODE =1; CALL SVC3(TP);
  GO TO RTN;
END;

/*GET KEYS*/
P=TP->RETN_RTN_PTR;
DO I=1 TO 11;
  KEY(12-I)=RETN_RTN1.DATA;
  P=RETN_RTN1_PTR;
END;
CALL SVC3(TP); /*FREE RTN MSG LIST*/
END /*FILE INIT*/;

RTN: CALL TRTN (PROC_ADDR,RTP);
END /*VINIT*/;

```

```

VINO1660
VINO1670
VINO1680
VINO1690
VINO1700
VINO1710
VINO1720
VINO1730
VINO1740
VINO1750
VINO1760
VINO1770
VINO1780
VINO1790
VINO1800
VINO1810
VINO1820
VINO1830
VINO1840
VINO1850

```

```

VNME: PROC:
VNMO0010
VNMO0020
VNMO0030
VNMO0040
VNMO0050
VNMO0060
VNMO0070
VNMO0080
VNMO0090
VNMO0100
VNMO0110
VNMO0120
VNMO0130
VNMO0140
VNMO0150
VNMO0160
VNMO0170
VNMO0180
VNMO0190
VNMO0200
VNMO0210
VNMO0220
VNMO0230
VNMO0240
VNMO0250
VNMO0260
VNMO0270
VNMO0280
VNMO0290
VNMO0300
VNMO0310
VNMO0320
VNMO0330
VNMO0340
VNMO0350
VNMO0360
VNMO0370
VNMO0380
VNMO0390

/* T-PROC AT THE ENTITY LEVEL, INVOKED BY DMB (BASE DATA MANIPULATION)
SUBSYSTEM AT THE USER INTERFACE LEVEL, TO CHECK FOR LEGALITY OF NAMES
STATED BY USER BEFORE ACCEPTING DATA */
XINCLUDE SERVICE, AVNME, ERECT, EGACT, DECAT;
DCL ENAME CHAR (8);
DCL (P,RTP) PTR, RTN CODE FIXED BIN, NULL BUILTIN;
DCL PARENT_ENAME(25) CHAR (8); /* TEMP STRUCTURE */
DCL I FIXED BIN (31);

/* BEGIN */
CALL TBEG (PROC_ADDR,P);
ALLOCATE VNME_RTN SET (RTP);

CALL GECT (T.ENAME,EINFO, RTN_CODE);
IF RTN_CODE ^= 0 THEN DO;
RTP->VNME_RTN.N=-1; /**** PROBLEM WITH ENTITY SET NAME *****/
GO TO RTN;
END;

/*VALIDATE ATTRIBUTES */
DO I = 1 TO T.TN;
IF T.PARENT(I)=0 THEN ENAME=T.ENAME;
ELSE ENAME = PARENT_ENAME (T.PARENT (I));
CALL GACT (T.ENAME (I), ENAME, AINFO, RTN_CODE);
IF RTN_CODE ^= 0 THEN DO;
RTP->VNME_RTN.N=I; /*O+ IS PROBLEM WITH ATTRIBUTE NAME *****/
GO TO RTN;
END;

IF AINFO.TYPE='E' THEN PARENT_ENAME (I)=AINFO.ENAME; /*REMEMBER
TARGET ESET NAME */
END /* DO I */;

RTP->VNME_RTN.N=0; /* NO PROBLEM */
RTN: CALL TRTN (PROC_ADDR, RTP);
RETURN;
END /* VNME */;

```

VM/SP CONVERSATIONAL MONITOR SYSTEM

FILE: VNM1 PLODPT A1

```

VNM1: PROC (T, RTN_CODE, N);
  /**SUBROUTINE TO SET UP CALL TO VNME**/
  %INCLUDE AVNME.SERVICE.FOEBUG;
  DCL (RTN_CODE,N) FIXED BIN;
  DCL 1 T, /*DATA STRUCTURE OF NAMES TO BE VERIFIED */
      2 ENAME CHAR(8),
      2 T1 FIXED BIN, /*NUMBER OF ATTR*/
      2 T1 (25),
      3 PARENT FIXED BIN,
      3 ANAME CHAR (8);
  DCL (TP,P)PTR, NULL BUILTIN;

  ALLOCATE VNME_ARG;
  VNME_ARG.T-T;

  CALL TCALL('VNME',1,P,TP);
  CALL SVC3(P);
  RTN_CODE= TP->VNME_RTN_CODE;
  N=TP->VNME_RTN.N;
  IF N ^= 0
  THEN DO;
    IF N = -1 THEN PUT SKIP LIST ('ILLEGAL ENTITY SET NAME');
    ELSE PUT SKIP EDIT ('ILLEGAL ATTRIBUTE NAME ',T.T1.ANAME(N))(A,A);
  END;

  RETURN; END;

```

DUM00010
 DUM00020
 DUM00030
 DUM00040
 DUM00050
 DUM00060
 DUM00070
 DUM00080
 DUM00090
 DUM00100
 DUM00110
 DUM00120
 DUM00130
 DUM00140
 DUM00150
 DUM00160
 DUM00170
 DUM00180
 DUM00190
 DUM00200
 DUM00210
 DUM00220
 DUM00230
 DUM00240
 DUM00250
 DUM00260

FILE: VTPE PLIOPT A1 VM/SP CONVERSATIONAL MONITOR SYSTEM

```

VTPE: PROC (AINFO, MODE, BDATA, CDATA, DLEN, RTN_CODE);
/*A S-PROC AT THE ENTITY LEVEL TO CHECK FOR DATA TYPE VALIDITY AND
PERFORMS CONVERSION FROM CHAR TO BIT OR VICE VERSA DEPENDING ON
THE MODE GIVING*/

XINCLUDE DECAT; /*FOR AINFO TEMPLATE*/
XINCLUDE FOEBUG;
DCL MODE FIXED BIN;
DCL TO_CHAR FIXED BIN INIT (2), TO_BIT FIXED BIN INIT (1);
DCL TO_BIT_CHK FIXED BIN INIT (3);
DCL CDATA CHAR (40), BDATA BIT (320), (DLEN, RTN_CODE) FIXED BIN;
DCL NUM FIXED BIN (31); /*TO HOLD CONVERTED FIXED BIN NUMERIC DATA*/
DCL VERIFY BUILTIN; DCL BIN BUILTIN;
DCL DATA CHAR (40) VARYING;
DCL (UNSPEC,SUBSTR)BUILTIN, I FIXED BIN;

/*BEGIN PROCESSING...*/
RTN_CODE=0; /*INIT TO O.K.*/
SELECT (MODE);

WHEN (TO_BIT, TO_BIT_CHK) DO;
/*FROM CHAR STRING TO BIT STRING*/

IF AINFO.TYPE='V' THEN DO; /*FOR V TYPE MODE ONLY*/

  SELECT (AINFO.VTYPE);

  WHEN ('N') DO; /*NUMERIC DATA*/
    DATA=SUBSTR(CDATA,1,DLEN);
    I=VERIFY(DATA,'0123456789 ');
    IF I=0 THEN DO;
      RTN_CODE=1; /*ILLEGAL NUM DATA*/
      RETURN;
    END;
    IF DATA = (40) ' ' THEN NUM=0;
    ELSE NUM=DATA; /*CONVERSION */

    /*CHECK FOR LIMITS*/
    IF MODE=TO_BIT_CHK THEN DO;
      IF NUM<AINFO.MAX & NUM>AINFO.MIN THEN LEAVE;
    ELSE DO; /*VIOLATING LIMITS*/
      RTN_CODE=9; /****5 IS LIMIT VIOLATIONS******/
      RETURN;
    END;
  END;

  SUBSTR(BDATA,1,32)=UNSPEC (NUM); /*TO BIT STRING*/
  DLEN=4; /*4 BYTES LONG*/
  END /*WHEN 'N'*/;

OTHERWISE DO;
  BDATA=UNSPEC(SUBSTR(CDATA,1,AINFO.MLEN));
  DLEN = AINFO.MLEN; END;

```

VTPO0010
VTPO0020
VTPO0030
VTPO0040
VTPO0050
VTPO0060
VTPO0070
VTPO0080
VTPO0090
VTPO0100
VTPO0110
VTPO0120
VTPO0130
VTPO0140
VTPO0150
VTPO0160
VTPO0170
VTPO0180
VTPO0190
VTPO0200
VTPO0210
VTPO0220
VTPO0230
VTPO0240
VTPO0250
VTPO0260
VTPO0270
VTPO0280
VTPO0290
VTPO0300
VTPO0310
VTPO0320
VTPO0330
VTPO0340
VTPO0350
VTPO0360
VTPO0370
VTPO0380
VTPO0390
VTPO0400
VTPO0410
VTPO0420
VTPO0430
VTPO0440
VTPO0450
VTPO0460
VTPO0470
VTPO0480
VTPO0490
VTPO0500
VTPO0510
VTPO0520
VTPO0530
VTPO0540
VTPO0550

VM/SP CONVERSATIONAL MONITOR SYSTEM

FILE: VTPE PLIOPT A1

```

        END /*SELECT VTTYPE*/;
        END /*TYPE*/;
        END /*TO_BIT MODE*/;

        WHEN (TO_CHAR) DO;
            IF AINFO.TYPE='V' THEN DO;
                SELECT (AINFO.VTYPE);

                WHEN ('N') DO;
                    UNSPEC (NUM)=SUBSTR(BDATA,1,32); /*GET DATA*/
                    PUT STRING(CDATA) EDIT(NUM)(F(AINFO.MLEN));
                    DLEN=AINFO.MLEN;
                    END;

                OTHERWISE UNSPEC(CDATA) = BDATA;

                END /*SELECT VTTYPE*/;
                END /*TYPE*/;
                END /*MODE TO_CHAR*/;

            END /*SELECT MODE*/;
            END /*VTPE*/;
    
```

VTP00560
 VTP00570
 VTP00580
 VTP00590
 VTP00600
 VTP00610
 VTP00620
 VTP00630
 VTP00640
 VTP00650
 VTP00660
 VTP00670
 VTP00680
 VTP00690
 VTP00700
 VTP00710
 VTP00720
 VTP00730
 VTP00740
 VTP00750
 VTP00760
 VTP00770
 VTP00780
 VTP00790

