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**

<table>
<thead>
<tr>
<th>1. REPORT NUMBER</th>
<th>2. GOVT ACCESSION NO.</th>
<th>3. RECIPIENT'S CATALOG NUMBER</th>
</tr>
</thead>
<tbody>
<tr>
<td>Technical Report #9</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>4. TITLE (and Subtitle)</th>
<th>5. TYPE OF REPORT &amp; PERIOD COVERED</th>
</tr>
</thead>
<tbody>
<tr>
<td>FSTV - The Software Test Vehicle For The Functional Hierarchy of The Infoplex Database Computer</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>7. AUTHOR(s)</th>
<th>8. CONTRACT OR GRANT NUMBER(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Meichun Hsu</td>
<td>N0039-81-c-0663</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>9. PERFORMING ORGANIZATION NAME AND ADDRESS</th>
<th>10. PROGRAM ELEMENT, PROJECT, TASK AREA &amp; WORK UNIT NUMBERS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sloan School of Management, Massachusetts Institute of Technology, Cambridge MA 02139</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>11. CONTROLLING OFFICE NAME AND ADDRESS</th>
<th>12. REPORT DATE</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>January 1982</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>13. NUMBER OF PAGES</th>
<th>14. MONITORING AGENCY NAME &amp; ADDRESS (IF DIFFERENT FROM CONTROLLING OFFICE)</th>
</tr>
</thead>
<tbody>
<tr>
<td>270</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>15. SECURITY CLASS: (OF THIS REPORT)</th>
<th>16. DISTRIBUTION STATEMENT (OF THIS REPORT)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Unclassified</td>
<td>Approved for public release, distribution unlimited</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>17. DISTRIBUTION STATEMENT (OF THE ABSTRACT ENTERED IN BLOCK 20, IF DIFFERENT FROM REPORT)</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>18. SUPPLEMENTARY NOTES</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>19. KEY WORDS (Continue on reverse side if necessary and identify by block number)</th>
</tr>
</thead>
<tbody>
<tr>
<td>database computer, database management system, software test vehicle, hierarchical systems</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>20. ABSTRACT (Continue on reverse side if necessary and identify by block number)</th>
</tr>
</thead>
</table>

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, database 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 database capabilities and architectural compatibility.
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, database 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 database 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

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

This report is divided into 6 chapters. The next chapter provides an overview of the functionalities of FSTV as a database 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 database 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 Schema 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 Definition 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 function 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

\[
\text{WORKS\_IN (MAIN\_LOC (CITY))}
\]

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

\[
\text{EMPLOYEE}
\]

\[
\text{EMPNAME, EMPADDR, AGE > 50, WORKS\_IN (DEPTNUM = 15)}
\]

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 DATABASE INTERNAL SCHEMA

The database internal schema describes how data in the database 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"
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:

\[
\begin{align*}
2 & \text{CNTL\_INFO}, \\
3 & \text{LEN \text{ FIXED BIN (15)}}, \\
3 & \text{CBTP \text{ FIXED BIN (15)}}, \\
3 & \text{PTR PTR}, \\
2 & \text{DATA}, \\
& \ldots
\end{align*}
\]
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. Similarly, 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 further 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 AUTHORIZATION LEVEL

LEVEL-2

VIEW TRANSLATION LEVEL

VIEW ENFORCEMENT LEVEL

VALIDITY CHECKING LEVEL

VIRTUAL INFORMATION LEVEL

N-ARY ENTITY LEVEL

N-ARY/B-ARY LEVEL

UNARY SET LEVEL

DATA ENCODING LEVEL

MEMORY MGMT LEVEL

LEVEL-3

LEVEL-4

USER INTERFACE /SECURITY PROCESSING GROUP

BASE DATA MODEL PROCESSING GROUP

INTERNAL DATA MODEL PROCESSING GROUP

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 CONTROL 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.
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, INVAL 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.
Condition: \( > 50 \)

Condition: = 'NEWYORK'

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.
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 CAtalouge 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.
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 DFBI 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 Bootstrap) 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 subroutine 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.)

LEX. 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.

UPEI. 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 UPEI 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. UPEI 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.

VNMI. 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.
<table>
<thead>
<tr>
<th>LEVEL</th>
<th>PROCNAME</th>
<th>COUNT</th>
<th>TOT ELAPSED TIME</th>
<th>TOT RUN TIME</th>
<th>RUN TIME/INVOCATION</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>USER</td>
<td>1</td>
<td>1505283</td>
<td>80955</td>
<td>80955</td>
</tr>
<tr>
<td>2</td>
<td>VINIT</td>
<td>1</td>
<td>123976</td>
<td>5233</td>
<td>5233</td>
</tr>
<tr>
<td>1</td>
<td>VIXKE</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>DEFE</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>RENTE</td>
<td>1</td>
<td>1146343</td>
<td>17482</td>
<td>17482</td>
</tr>
<tr>
<td>1</td>
<td>UPDE</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>DEFA</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>SHWE</td>
<td>1</td>
<td>1424328</td>
<td>11921</td>
<td>11921</td>
</tr>
<tr>
<td>3</td>
<td>DEFP</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>DEFB</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>RETN</td>
<td>8</td>
<td>1503249</td>
<td>567142</td>
<td>70892</td>
</tr>
<tr>
<td>3</td>
<td>UPON</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>NINIT</td>
<td>1</td>
<td>10419</td>
<td>5637</td>
<td>5637</td>
</tr>
<tr>
<td>4</td>
<td>RET</td>
<td>138</td>
<td>939625</td>
<td>939625</td>
<td>6808</td>
</tr>
<tr>
<td>4</td>
<td>DEL</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>REP</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>CRT</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>5</td>
<td>MINIT</td>
<td>1</td>
<td>1264</td>
<td>1264</td>
<td>1264</td>
</tr>
<tr>
<td>5</td>
<td>L1</td>
<td>801</td>
<td>180119</td>
<td>180119</td>
<td>224</td>
</tr>
</tbody>
</table>

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
FSTV: THE SOFTWARE TEST VEHICLE FOR THE FUNCTIONAL HIERARCHY OF ETC(U)
JAN 82 M HSU
N0039-81-C-0663

2 or 3
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 'as 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


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)


APPENDIX I

SAMPLE TERMINAL SESSION (a)
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 UON/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

5 entity sets in the system

PROJECTS
DEPT
EMPLOYEE
E*ASET
E*ESET

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

<table>
<thead>
<tr>
<th>ATTRIBUTE NAME</th>
<th>FUNCTION</th>
<th>TYPE</th>
<th>EENAME</th>
<th>VTYPE</th>
<th>MAX LEN</th>
<th>MAX VALUE</th>
<th>MIN VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>UNDER</td>
<td>M:1</td>
<td>E</td>
<td>DEPT</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PROJNUM</td>
<td>KEY</td>
<td>V</td>
<td>N</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>99999</td>
<td>-99999</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PROJNAME</td>
<td>1:1</td>
<td>V</td>
<td>C</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

3 attributes defined for PROJECTS

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

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

<table>
<thead>
<tr>
<th>ENTITY SET NAME</th>
<th>EMPLOYEE</th>
</tr>
</thead>
<tbody>
<tr>
<td>ATTRIBUTE NAME</td>
<td>FUNCTION</td>
</tr>
<tr>
<td>WORKS_IN</td>
<td>M:1</td>
</tr>
<tr>
<td>BOSS</td>
<td>M:1</td>
</tr>
<tr>
<td>AGE</td>
<td>1:1</td>
</tr>
<tr>
<td></td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>99</td>
</tr>
<tr>
<td>EMPADDR</td>
<td>1:1</td>
</tr>
<tr>
<td></td>
<td>40</td>
</tr>
<tr>
<td>EMPNAME</td>
<td>1:1</td>
</tr>
<tr>
<td></td>
<td>20</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>ENTITY SET NAME</th>
<th>DEPT</th>
</tr>
</thead>
<tbody>
<tr>
<td>ATTRIBUTE NAME</td>
<td>FUNCTION</td>
</tr>
<tr>
<td>DEPTNAME</td>
<td>1:1</td>
</tr>
<tr>
<td></td>
<td>20</td>
</tr>
<tr>
<td>DEPTNUM</td>
<td>KEY</td>
</tr>
<tr>
<td></td>
<td>5</td>
</tr>
</tbody>
</table>

ENTITY SET NAME? USE * IF ALL ENTITY SET NAMES ARE DESIRED.
DBA: DATA DEFINITION (DD) OR DATA MANIPULATION (DM) OR DATA DEFINITION QUERY (DOQ)?

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

<table>
<thead>
<tr>
<th>DEPTNAME, DEPTNUM</th>
</tr>
</thead>
<tbody>
<tr>
<td>ENTITY SET NAME</td>
</tr>
<tr>
<td>-------------------</td>
</tr>
<tr>
<td>DEPTNAME</td>
</tr>
<tr>
<td>Economics</td>
</tr>
<tr>
<td>'Sloan School'</td>
</tr>
</tbody>
</table>

Answer to the query

ENTER ENTITY SET NAME

CREATE QUERY session

DM: ENTER MANIPULATION COMMAND:

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

MODIFY(MOD):

TYPE YES IF NEED HELP

MODIFY: ENTER ENTITY SET NAME

DEPT
ENTER MODIFICATION OPERATORS AND NAMES OF ATTRIBUTES SEPARATED BY COMMAS

<table>
<thead>
<tr>
<th>DEPTNUM, DEPTNAME</th>
</tr>
</thead>
<tbody>
<tr>
<td>ENTITY SET NAME</td>
</tr>
<tr>
<td>DEPT</td>
</tr>
</tbody>
</table>

10: DEPTNUM, REP: DEPTNAME

The way to modify: identify by DEPTNUM, then replace DEPTNAME

ENTER DATA:
15, Sloan School

Change DEPTNAME of DEPT with DEPTNUM 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

<table>
<thead>
<tr>
<th>DEPTNUM</th>
<th>DEPTNAME</th>
</tr>
</thead>
<tbody>
<tr>
<td>14</td>
<td>Economics</td>
</tr>
<tr>
<td>15</td>
<td>Sloan School</td>
</tr>
</tbody>
</table>

Answer to query
ENTER ENTITY SET NAME

DEPT

ENTER ATTRIBUTE NAMES AND PREDICATE, SEPARATED BY COMMAS

DEPTNAME,DEPTNUM=15

ENTITY SET NAME | DEPT |
| DEPTNAME | DEPTNUM |
| Sloan School | 15 |

Answer for the query

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 |

Answer

Enter entity set name

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

CRT

Leave query session

DM: ENTER MANIPULATION COMMAND:
CREATE: ENTER ENTITY SET NAME

DEPT

ENTER NAMES OF ATTRIBUTES SEPARATED BY COMMA

DEPARTMENT, DEPTNAME

ENTER DATA:
6, EE, Computer Science

21, Humanities

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

EMPLOYEE, EMPADDR, AGE, WORKS IN(DEPARTMENT)

ENTER DATA:
Hoo-min Toong, E53-300, 50, 15

Mike Abraham, somewhere in Somerville, 40, 14

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)

<table>
<thead>
<tr>
<th>ENTITY SET NAME</th>
<th>EMPLOYEE</th>
</tr>
</thead>
<tbody>
<tr>
<td>EMPNAME</td>
<td>WORKS_IN (DEPTNUM</td>
</tr>
<tr>
<td>Mike Abraham</td>
<td>14</td>
</tr>
<tr>
<td>Hoo-min Toong</td>
<td>15</td>
</tr>
</tbody>
</table>

ENTER ENTITY SET NAME

**EMPLOYEE**

ENTER ATTRIBUTE NAMES AND PREDICATE, SEPARATED BY COMMAS

: EMPNAME, AGE<45

ENTITY SET NAME

**EMPLOYEE**
<table>
<thead>
<tr>
<th>EMPNAME</th>
<th>AGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mike Abraham</td>
<td>40</td>
</tr>
</tbody>
</table>

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 TEST01

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 issu(-80k)

ENTER TCALL TRACE OPTION (4 BITS)

ENTER SYSTEM ERROR MESSAGE TRACE OPTION (4 BITS)

ENTER MEMORY REQUEST TRACE OPTION (1 BIT):

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

ATTRIBUTE AGE DEFINED

PRESS ENTER TO CONTINUE

NEW another panel session defined the BOSS attribute

ATTRIBUTE BOSS DEFINED

PRESS ENTER TO CONTINUE

NEW a panel session defined the DEPT entity set

ENTITY SET DEPT DEFINED

PRESS ENTER TO CONTINUE

NEW Attribute DEPTNUM defined here

ATTRIBUTE DEPTNUM DEFINED

PRESS ENTER TO CONTINUE

NEW Attribute DEPTNAME defined here

ATTRIBUTE DEPTNAME DEFINED

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

OLD
- Defined attribute WORKS-IN in EMPLOYEE

ATTRIBUTE WORKS-IN DEFINED

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

OLD
- Defined attribute UNDER in PROJECTS.entity set

ATTRIBUTE UNDER DEFINED

PRESS ENTER TO CONTINUE
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 a 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 PAGE 002

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):

/ *
//DCL OF ARGUMENTS TO COMMUNICATE WITH THE MODULE 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 ('B'),
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 PTR PTR INIT (NULL),
2 RTN_CODE FIXED BIN (15), /*LEN OF PRIMITIVE ELEM OF THE PSET, IN BYTES. IF -1 THEN VARIABLE IN LENGTH*/
2 IMP,
3 HOW BIT (8). /*SEEF PHOW FOR DETAIL*/
3 NUMPRT FIXED BIN (15); /*RELY ONLY IF HOW = SA*/
DCL 1 DEFP_HOW.
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), /*O 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).
FILE: DATA MACLIB A VM/SP CONVERSATIONAL MONITOR SYSTEM

2 LEN FIXED BIN (15) INIT(13), F1A00030
2 CBTP FIXED BIN (15) INIT( 15 ), F1A00040
2 PTR PTR INIT(NULL()), F1A00050
2 OP CHAR (1), /*'1'='INIT, '5'='STORE KEY*/ F1A00060
2 FNAME CHAR(8), /*NAME OF FILE*/ F1A00070
2 KEY BIT(32); /*KEY VALUE RETURNED IN A FILE INIT*/ F1A00080
DCL 1 NINIT_RTN BASED(P), F1A00090
2 LEN FIXED BIN (15) INIT(6), F1A00100
2 CBTP FIXED BIN (15) INIT( 16 ), F1A00110
2 PTR PTR INIT(NULL()), F1A00120
2 RTN_CODE FIXED BIN (15) INIT (0), /*O=Q.K./*/ F1A00130
2 KEY BIT(32); /*KEY VALUE RETURNED IN A FILE INIT*/ F1A00140

DCL 1 NINIT_ARG BASED(P), F1A0020
2 LEN FIXED BIN (15) INIT(13), F1A0030
2 CBTP FIXED BIN (15) INIT( 17 ), F1A0040
2 PTR PTR INIT(NULL()), F1A0050
2 OP CHAR (1), /*'1'='INIT, '5'='STORE KEY*/ F1A0060
2 FNAME CHAR(8), /*NAME OF FILE*/ F1A0070
2 KEY BIT(32); /*KEY VALUE RETURNED IN A FILE INIT*/ F1A0080
DCL 1 NINIT_RTN BASED(P), F1A0090
2 LEN FIXED BIN (15) INIT(6), F1A0100
2 CBTP FIXED BIN (15) INIT( 18 ), F1A0110
2 PTR PTR INIT(NULL()), F1A0120
2 RTN_CODE FIXED BIN (15) INIT (0), /*O=Q.K./*/ F1A0130
2 KEY BIT(32); /*KEY VALUE RETURNED IN A FILE INIT*/ F1A0140

DCL 1 REP_ARG BASED(P), F1A0020
2 LEN FIXED BIN (15) INIT(108), F1A0030
2 CBTP FIXED BIN (15) INIT( 1), F1A0040
2 PTR PTR INIT (NULL), /* ------ */ F1A0050
2 ID BIT (32), /* ------ */ F1A0060
2 N FIXED BIN, /*LEN OF DATA IN BIT*/ F1A0070
2 DATA BIT(800); /*UP TO 100 BYTES LONG*/ F1A0080
DCL 1 REP_RTN BASED(P), F1A0100
2 LEN FIXED BIN (15) INIT( 2), F1A0110
2 CBTP FIXED BIN (15) INIT( 2), F1A0120
2 PTR PTR INIT (NULL), /* ------ */ F1A0130
2 ID BIT (32); /* ------ */ F1A0140
2 RTN_CODE FIXED BIN INIT (0); F1A0150
DCL 1 RET_ARG BASED(P), F1A0160
2 LEN FIXED BIN (15) INIT( 4 ), F1A0020
2 CBTP FIXED BIN (15) INIT( 19 ), F1A0030
2 PTR PTR INIT (NULL), /* ------ */ F1A0040
2 ID BIT (32); /* ------ */ F1A0050
DCL 1 RET_RTN BASED(P), F1A0060
2 LEN FIXED BIN (15) INIT(104), F1A0070
2 CBTP FIXED BIN (15) INIT( 20 ), F1A0080
2 PTR PTR INIT (NULL), /* ------ */ F1A0090
2 RTN_CODE FIXED BIN INIT (0), /*N FIXED BIN, */ F1A0100
2 ID BIT (32); /*LEN OF DATA IN BIT*/ F1A0110
2 DATA BIT(800); /*UP TO 100 BYTES LONG*/

DCL 1 RETE_ARG BASED(P), F1A00200
2 LEN FIXED BIN (15) INIT (61), F1A00030
2 CBTP FIXED BIN (15) INIT( 21 ), F1A00040
2 PTR PTR INIT (NULL), F1A00050
2 NODE FIXED BIN (15) INIT(1), /*NODE NUMBER*/ F1A00060
2 NAME CHAR (8). /*NAME OR ENAME*/ F1A00070
2 PARENT FIXED BIN (15) INIT(0). /*PARENT NODE NUMBER*/ F1A00080
2 GET BIT(B) INIT ('00000000000'), /*SEE RETEGET FOR DETAIL*/ F1A00090
2 N FIXED BIN INIT(0). /*NUMBER OF RELEVANT 'OR' PREDICATES*/ F1A00100
2 PRED (1)/*IN THIS VERSION ONLY ONE PREDICATE EACH TIME IS
ALLOWS; IN THE FUTURE MAY USE REFER OPTION TO N*/. F1A00110
3 CN FIXED BIN (15) INIT(1),/*CONDITION NUMBER*/ F1A00130
3 SETUP BIT(8), /*RELEVANT IF BSET IF MULTI-TARGET;
NOT USED IN CURRENT VERSION*/ F1A00140
3 OP BIT(B), /*COMPARISON OPS. SEE RETEOP FOR DETAIL*/ F1A00160
3 OLEN FIXED BIN (15) INIT(0), /* LEN OF COMPARE DATA IN BYTES*/ F1A00170
3 CDATA CHAR (40): /*40 BYTES MAXIMUM*/ F1A00180

DCL 1 RETE_ARG1 BASED (P), /*USED FOR 'BELONG' COMP OP*/ F1A00190
/*NOT USED IN CURRENT VERSION*/ F1A00200
2 LEN FIXED BIN INIT (8 ), /*PLUS VALUE LEN*/ F1A00210
2 CBTP FIXED BIN (15) INIT( 23 ), F1A00220
2 PTR PTR, /*POINT TO THE NEXT RETE_ARG*/ F1A00230
2 N FIXED BIN, /* # OF ELEMENTS IN THIS IMMED. VALUE SET*/ F1A00240
2 VALUE (1 REFER(RETE_ARG1.N)) CHAR(40); /*MAX OF 40 BYTES*/ F1A00250

DCL RETEVALUE LEN FIXED BIN (15) INIT (40); F1A00260

DCL 1 RETEGET, F1A00270
2 ANY BIT(B) INIT ('000000001'B), F1A00280
2 ALL BIT (B) INIT ('000000011'B), F1A00290
2 NO BIT (B) INIT ('000000000'B); /*NOTE ROOT NODE CANNOT
ASSUME THIS VALUE*/ F1A00300

DCL 1 RETEOP, F1A00310
2 EQ BIT (B) INIT ('000000000'B), F1A00320
2 GT BIT(B) INIT ('000000100'B), F1A00330
2 LT BIT(B) INIT ('000000011'B), F1A00340
2 BEL BIT(B) INIT ('000000011'B), F1A00350
2 NULL BIT (B) INIT ('000001000'B); /*NOT SPECIFIED*/ F1A00370

DCL 1 RETE_RTN BASED(P), F1A00380
2 LEN FIXED BIN (15) INIT (4). F1A00390
2 CBTP FIXED BIN (15) INIT( 22 ), F1A00400
2 PTR PTR INIT (NULL), F1A00410
2 RTN_CODE FIXED BIN INIT (0), F1A00420
2 N FIXED BIN (15); /*NUMBER OF INSTANCES OF ROOT NODE RETURNED*/ F1A00430

DCL 1 RETE_RTHI BASED(P), F1A00440
2 LEN FIXED BIN (15) INIT (46). F1A00450
2 CBTP FIXED BIN (15) INIT( 24 ), F1A00460
2 PTR PTR INIT (NULL), F1A00470
2 NODE FIXED BIN (15), /*NODE NUMBER*/ F1A00480
2 INDX FIXED BIN (15), /*IF MULTI OCCUR*/ F1A00490
2 DLEN FIXED BIN, /*LEN OF DATA IN BYTES*/ F1A00500
2 CDATA CHAR(40); /*DATA RETURNED*/ F1A00510

DCL 1 RETN_ARG BASED(P), F1A000200
2 LEN FIXED BIN (15) INIT (61), F1A00030
FILE: DATA MACLIB A VM/SP CONVERSATIONAL MONITOR SYSTEM

2 DATA,
3 NAME1 CHAR (8),
3 NAME2 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);
DCL 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 MLEN FIXED BIN,
3 TYPE CHAR (1),
3 FUNC CHAR (1),
3 VTYPE CHAR (1);
DCL 1 SHWE_RTN2 BASED (SHWE_RTN_P),
DCL 1 SHWE_ARG BASED (SHWE_RTN_P),
DCL 1 SHWE_ARG_BASED (SHWE_RTN_P),
2 LEN,
3 LEN FIXED BIN INIT (202),
3 CBTP FIXED BIN INIT (39),
3 PTR PTR INIT (NULL),
2 DATA,
3 NUMENAME FIXED BIN,
3 ENAME (25) CHAR (8);
2 UPDE_ARG BASED (P),
2 LEN,
3 CBTP FIXED BIN (15) INIT (56),
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 DLEN2 FIXED BIN (15), /*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 I0T FIXED BIN INIT(2), /*INSERT ATT*/
3 ADD FIXED BIN INIT(3), /*ADD ATT*/
DCL 1 VINIT_RTN BASED(P), F1A00090
  2 LEN_ FIXED BIN (15) INIT (6 ), F1A00100
  2 CBTP FIXED BIN (15) INIT( 33 ), F1A00110
  2 PTR PTR INIT (NULL), F1A00120
  2 RTN CODE FIXED BIN INIT (0 ), F1A00130
  2 KEY BIT(32); F1A00140
/
DCL 1 VNAME_ARG BASED(P), F1A00020
  2 LEN FIXED BIN (15) INIT (260), F1A00030
  2 CBTP FIXED BIN (15) INIT( 34 ), F1A00040
  2 PTR PTR INIT (NULL), F1A00050
  2 N, F1A00060
  3 NNAME CHAR(8), F1A00070
  3 TN FIXED BIN, F1A00080
  3 T1 (25 ), F1A00090
  4 PARENT FIXED BIN, F1A00100
  4 NAME CHAR(8); F1A00110
DCL 1 VNME_RTN BASED(P), F1A00120
  2 LEN_ FIXED BIN (15) INIT (4), F1A00130
  2 CBTP FIXED BIN (15) INIT( 35 ), F1A00140
  2 PTR PTR INIT (NULL), F1A00150
  2 RTN CODE FIXED BIN INIT (0 ), F1A00160
  2 N FIXED BIN; F1A00170
/
//DCL OF BCAT TEMPLATE*/
DCL 1 BINF0, F1D00020
  2 PSETID(2) BIT (32), F1D00030
  2 POS FIXED BIN, //POS OF POINTER ARRAY OF PSETID1 USED */ F1D00040
  2 FUNC CHAR (1); //"S" OR "M"*/; F1D00050
/
//DCL OF BASIC ENCODING UNIT (BEU) USED IN THE N-ARY LEVEL****/
DCL 1 BEU BASED(P), F1D00020
  2 N>Name FIXED BIN, F1D00030
  2 ID ARRAY(1 REFER (BEU.NPMPTR))BIT(32) INIT ((1)UNSPEC(NULL)), F1D00040
  2 DLEN FIXED BIN, F1D00050
  2 DATA BIT(J REFER (BEU.DLEN)); F1D00060
/
// CATALOGUE STRUCTURES AT THE ENTITY LEVEL*****/
DCL EINFO BIT (32); //HOLDS PSETID OF AN ENTITY SET**/ F1D00020
DCL 1 AINFO, F1D00030
  2 BSETID BIT (32), //BSETID OF THIS ATTRIBUTE**/ F1D00040
  2 PSETID BIT (32), //PSETID OF TARGET NODE**/ F1D00050
  2 ENAME CHAR (8), //IF TARGET IS A ENTITY SET*/ F1D00060
  2 MAX FIXED BIN (31), F1D00070
  2 MIN FIXED BIN (31), F1D00080
  2 MLEN FIXED BIN, F1D00090
  2 FUNC CHAR(1), F1D00010
  2 TYPE CHAR (1); F1D00110
  2 VTYPE CHAR (1); F1D00120
/
// DCLS OF STATIC KEYS USED AS THE ENTITY LEVEL *****/
// ONLY MODULE VINIT HAS THE WRITE ACCESS: OTHER MODULES R/O**/ F1D00020
// DCL KEY (1) BIT (32) EXT STATIC; F1D00030
DCL 1 KY //INDEX OF 11 KEYS*/ STATIC, F1D00040
  2 PESET FIXED BIN INIT (1). F1D00050
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 PAINF0 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 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*/
 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(B) INIT ('00000000'B),
  2 DATA BIT (I REFER (SEU.N));

/ / DCL 1 FDEBUG,
  2 LEV0 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 */

2 N FIXED BIN,
2 INVAL CHAR(1) INIT ('N'),
2 DATA CHAR(I REFER (SU.P)),
(SEU PTR)

/* ARCHITECTURAL LEVELS */
DCL 1 ARCH(19),
  2 LEVEL FIXED BIN(31) INIT (1,2,2,2,2,
    3,2,2,3,3,
    3,3,4,4,4,
    4,4,5,5),

  2 PROCNAME CHAR (8) INIT ('USER', 'VINIT', 'VMME', 'DEFE', 'RETE',
    'UPDE', 'DEFA', 'SHWE', 'DEFP', 'DEBF',
    'RETN', 'UPD', 'FINIT', 'RET', 'DEL',
    'REP', 'CRT', 'MINIT', 'L1');

/* SIMULATED STORAGE HIERARCHY, KNOWN TO DUMFSTV, DUMLI, FHL */
DCL SH AREA (30000) BASED (SH_BASE);
DCL SH_BASE PTR EXT;
DCL PACKET(30000) CHAR (8) BASED (PP), PP OFFSET (SH) EXT;

/* THIS IS THE PROC_TBL TO BE INCLUDED IN THE SVC51 ROUTINE */
DCL 1 PROC_TBL (18),
  2 N CHAR(7) INIT ('VINIT', 'DEFA', 'DEF', 'RETE', 'UPDE', 'SHWE',
    'VMME',
    'RETN', 'UPD', 'DEFP', 'DEBF', 'FINIT',
    'RET', 'REP', 'CRT', 'DEL', 'MINIT', 'USER'),

  2 LEVEL FIXED BIN INIT ((16,(5),(5),0,7),
    13-17 ARE IN THE MEMORY MGMT LEVEL (ASSIGNED LEVEL 0), 'L1'
    IS THE INTERFACE SUBMODULE AT THE SH-STV (ASSIGNED LEVEL 1). */

DCL PROC_TBL_LEN FIXED BIN INIT (18);

ACRT ADEFA ADEFB ADEFE ADEFP
ADEL AMINI >ANINI @AREP ARET
ARETE YARETN ASHWE AUPDE AUPREP
AVINI #AVMME I0BCAT DBEU DEGAT
DKEY DNKEY EDPCAT DSEU QFOEBUG
DSEU FSTRUC 55H SVCTB SVCTB
LIBPOS
DCL VTEP 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 (32O), CHAR (40), FIXED BIN, FIXED BIN);
  FIE00020
  FIE00030
  FIE00040
  FIE00050
  FIE00060
/ / /
DCL ALLA ENTRY //ENTRY DCL FOR MODULE ALLA*/
  (CHAR (8), 25 CHAR (8), FIXED BIN, FIXED BIN);
  FIE00020
  FIE00030
/ / /
DCL GBCF ENTRY / ENTRY DCL FOR MODULE GBCF WHICH RETRIEVES BINF*/
  (BIT(32), 1,2 (2) BIT(32), 2 FIXED BIN, 2 CHAR(1), FIXED BIN);
  FIE00020
  FIE00030
/ / /
/ / ENTRY DCL OF MODULE BINF*/
DCL BINF ENTRY;
  FIE00020
  FIE00030
/ / /
DCL CRTB ENTRY //ENTRY DCL FOR MODULE CRTB*/
  (BIT(32), BIT(32), BIT(32), BIT(*), CHAR(1), FIXED BIN);
  FIE00020
  FIE00030
/ / /
DCL CRTN ENTRY //ENTRY DCL FOR MODULE CRTN*/
  PTR, (25) BIT(1), FIXED BIN;
  FIE00020
  FIE00030
/ / /
DCL CRTP ENTRY /*ENTRY DCL FOR MODULE CRTP*/
  (BIT(32), BIT(*), BIT (32), FIXED BIN);
  FIE00020
  FIE00030
/ / /
/ / ENTRY DCL OF DBA MODULE*/
DCL DBA ENTRY;
  FIE00020
  FIE00030
/ / /
DCL DDG ENTRY; //ENTRY DCL FOR MODULE DDG*/
  FIE00020
  FIE00030
/ / /
DCL DDV ENTRY; //ENTRY DCL FOR MODULE DDV*/
  FIE00020
  FIE00030
/ / /
DCL DELB ENTRY //ENTRY DCL FOR MODULE DELB*/
  (BIT(32), BIT(32), BIT(32), CHAR(1), FIXED BIN);
  FIE00020
  FIE00030
/ / /
DCL DELN ENTRY //ENTRY DCL FOR MODULE DELN*/
  (PTR, (25) BIT (1), FIXED BIN);
  FIE00020
  FIE00030
/ / /
DCL Delp ENTRY //ENTRY DCL FOR MODULE DELP*/
  (BIT(32), BIT(32), FIXED BIN);
  FIE00020
  FIE00030
/ / /
DCL DFB1 ENTRY //ENTRY DCL FOR MODULE DFB1*/
  (CHAR(1), BIT (32), BIT (32), FIXED BIN, BIT(32));
  FIE00020
  FIE00030
/ / /
DCL DFP1 ENTRY (CHAR(1), FIXED BIN, CHAR(1), FIXED BIN, BIT(32));
  FIE00020
  FIE00030
/ / /
DCL DMB ENTRY; //ENTRY DCL FOR MODULE DMB*/
  FIE00020
  FIE00030
/ / /
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);
  FIE00020
  FIE00030
  FIE00040
  FIE00050
/ / /
DCL GECT ENTRY //ENTRY DCL FOR MODULE GECT*/
  (CHAR(8), BIT(32), FIXED BIN);
  FIE00020
  FIE00030
FILE: ENTRY MACLIB A VM/SP CONVERSATIONAL MONITOR SYSTEM PAGE 002

/ /
DCL LEX ENTRY /*ENTRY DCL FOR MODULE LEX*/
(FIXED BIN, 1 (25), 2 CHAR (40), 2 FIXED BIN, FIXED BIN,
FIXED BIN):
F1E00020
F1E00030
F1E00040
/ /
DCL LEXT ENTRY /*ENTRY DCL FOR MODULE LEXT*/
(FIXED BIN, 1 (25), 2 CHAR (40), 2 FIXED BIN, 1 (25), 2 CHAR (1),
2 CHAR (40), 2 FIXED BIN, (25) CHAR (1), FIXED BIN):
F1E00020
F1E00030
F1E00040
F1E00050
F1E00060
/ /
DCL MODN ENTRY /*ENTRY DCL FOR MODULE MODN*/
(PTR (25) BIT (1), FIXED BIN):
F1E00020
F1E00030
/ /
DCL MPST ENTRY /*ENTRY DCL FOR MODULE MPST*/
(BIT (32), FIXED BIN):
F1E00020
F1E00030
/ /
/*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 BIT (32), FIXED BIN):
F1E00020
F1E00030
F1E00040
F1E00050
F1E00060
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):
F1E00020
F1E00030
F1E00040
F1E00050
F1E00060
DCL PRNT ENTRY /*ENTRY DCL FOR MODULE PRNT*/
(PTR, PTR, FIXED BIN):
F1E00020
F1E00030
/ /
/*ENTRY DCL OF MODULE RELV*/
DCL RELV ENTRY:
F1E00020
F1E00030
/ /
DCL REPB ENTRY /*ENTRY DCL FOR MODULE REPB*/
(BIT (32), BIT (32), BIT (32), BIT (+), CHAR (1), FIXED BIN):
F1E00020
F1E00030
/ /
DCL SRCH ENTRY /*ENTRY DCL FOR MODULE SRCH*/
(BIT (32), BIT (+), BIT (32), FIXED BIN):
F1E00020
F1E00030
/ /
DCL RET1 ENTRY /*ENTRY DCL FOR N-ARY LOW LEVEL SUBROUTINE RET1*/
(BIT (32), PTR, FIXED BIN):
F1E00020
F1E00030
DCL DEL1 ENTRY /*ENTRY DCL FOR N-ARY LOW LEVEL SUBROUTINE DEL1*/
(BIT (32), FIXED BIN):
F1E00040
F1E00050
DCL REP1 ENTRY /*ENTRY DCL FOR N-ARY LOW LEVEL SUBROUTINE REP1*/
(BIT (32), PTR, FIXED BIN):
F1E00060
F1E00070
DCL CRT1 ENTRY /*ENTRY DCL FOR N-ARY LOW LEVEL SUBROUTINE CRT1*/
(BIT (32), PTR, FIXED BIN):
F1E00080
F1E00090
/ /
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):
F1E00020
F1E00030
F1E00040
F1E00050
/ /
/*DCL OF ENTRY FOR USER*/
DCL USER ENTRY:
EUS00020
EUS00030
/ /
DCL VNM1 ENTRY /*ENTRY DCL FOR MODULE VNM1*/
F1E00020
FILE: ENTRY MACLIB A VM/SP CONVERSATIONAL MONITOR SYSTEM PAGE 003

(1, 2 CHAR(8), 2 FIXED BIN, 2 (25), 3 FIXED BIN, 3 CHAR(8),
 FIXED BIN, FIXED BIN);
F1E00030
F1E00040
F1E00050
/
/*SERVICE ROUTINES DCLS****/
DCL TCALL ENTRY (CHAR(7), FIXED BIN, PTR, PTR);
DCL TRTN 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 PROC_NAME CHAR(7);
DCL 1 PROC_ADDR,
(2 LEVEL,
 2 VPID,
 2 BOXID) FIXED BIN;
F1E00030
F1E00040
F1E00050
/
/*DCLS OF SVC SEND SERVICES*/
DCL SVC51 ENTRY (CHAR(7), PTR, FIXED BIN);
DCL SVC52 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 RN_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 SVC51 IS CALLED*/
DCL 1 PROC_ADDR, /*USED IF SVC52 IS CALLED*/
(2 LEVEL,
 2 VPID,
 2 BOXID) FIXED BIN (15);
F1E00030
F1E00040
F1E00050
/
DCL HEX4 EXT ENTRY (PTR) RETURNS(CHAR(8));
/* ROUTINE TO CONVERT A PTR TO HEX STRING */
DCL HEX EXT ENTRY (PTR) RETURNS(CHAR(8));
F1E00030
HEX0010
HEX0020
/
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);
F1E00030
F1E00040
F1E00050
/
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);
F1E00030
F1E00040
/
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);
F1E00030
F1E00040
F1E00050
/
DCL BUILD ENTRY /*ENTRY DCL FOR MODULE BUILD*/
(25) PTR, (25) BIT(32) ALIGNED,
F1E00030
<table>
<thead>
<tr>
<th>FILE: ENTRY</th>
<th>MA LIB A</th>
<th>VM/SP CONVERSATIONAL MONITOR SYSTEM</th>
<th>PAGE 004</th>
</tr>
</thead>
</table>

```c
PTR, FIXED BIN, PTR, FIXED BIN); F1E00040

//
DCL PAKT ENTRY /*ENTRY DCL FOR MODULE PAKT*/;
(FIXED BIN(31), CHAR(8), CHAR(1)); F1E00020

//
DCL GMEM ENTRY /*ENTRY DCL FOR MODULE GMEM*/;
(FIXED BIN, BIT(32)); F1E00020

//
DCL GSEU ENTRY /* ENTRY DCL FOR GSEU SUBROUTINE */;
(BIT(32), PTR, FIXED BIN); F1E00020

//
DCL PSEU ENTRY /* ENTRY DCL FOR PSEU SUBROUTINE */;
(BIT(32), PTR, FIXED BIN); F1E00020

//
EVIPCE EALLA EBCT EBNV ECRTB
ECRTN ECRTP EDBA EDBB EDDV
EDELB EDELN EDELP EDFB1 EDFP1
EDMB EIGACT EGECT ELEX ELEXT
EMDDON EMSTP EPCT EPRINT ERELV
EREPB ERSCH IESUB1 EUPE1 EUSER
EVNM1 SERVICE ESVCS *HEX NEVERIM
EREIB ERETIP EBUIL UEPAKT YEGMEM
EGSEU EPSEU EPSEU EPSEU EPSEU
```
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 WHOS NAMES IS GIVEN
AS ENAME, AND THESE ATTRIBUTE NAMES ARE TO BE RETURNED IN THE
ARRAY ANAMES */

INCLUDE SERVICE, DECAT, ARETE;

DCL (P, RP, TP, PP, RP) PTR, NULL BUILTIN, (RTN_CODE, 1) FIXED BIN;

DCL ENAME CHAR (B), ANAMES (25) CHAR (B), NUM_ANAMES FIXED BIN;

/* BEGIN */

ALLOCATE RETE_ARG; RP, LP = P;

RETE_ARG.NAME = 'E*ASET';

RETE_ARG.GET = RETEGET.ALL;

DO 1 = 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'; /* NAME 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) = 6;

RETE_ARG.CDATA = ENAME;

END /* WHEN 3 */;

END /* SELECT */;

END /* DO 1 */;

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

CALL SVC3 (RP);

/* EXAMINE RETURN */

P = TP;

IF RETE_RTN.RTN_CODE = 0 THEN DO;

/* RTN_CODE = 0 IS FROM RETE CALL *******/

CALL SVC3 (P); GO TO RTN; END;

NUM_ANAMES = RETE_RTN.N;
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 = 2 /* NODE 2 IS ANAME NODE */
            THEN DO; ANAMES(I)=RETE_RTN1.CDATA;
               GO TO NEXT; /* GET OUT OF P LOOP */
         END;
      END /* P */;
   END /* DO I */;
   CALL SVC3 (TP);
   RTN: END /* ALLA */;

133
FILE: BUILD PLIOPT A1 VM/SP CONVERSATIONAL MONITOR SYSTEM PAGE 001

BUILD PROC /* SUBROUTINE USED BY RETN TO BUILD RETURN DATA*/
(CURSOR,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,RTN,TP,SP,LP) PTR,(NULL,UNSPEC) BUILTIN, TP1 PTR;
DCL (RNODE,RTN.CODE) FIXED BIN;

%INCLUDE 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.INDX=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=RETNGET.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.INDX=1;
TP1 = CURSOR(RETN_ARG.NODE);
IF TP1=NULL THEN DO;
RTN_CODE=1; /* BAD CURSOR */
RETURN;
END;
TP->RETN_RTN1.DLEN=TP1->BEU.DLEN/8;
TP->RETN_RTN1.DATA=TP1->BEU.DATA;
END /*IF GET*/;
END /*DO P=...*/;

BU00000
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, FDEBUG, EPSEU, EGMEM;
DCL P PTR, (NULL, UNSPEC) BUILTIN;
DCL (I, RTN_CODE) FIXED BIN, (ID, ID1) BIT(32);

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

CALL GMEM (I+5, ID); /* S: 3 FOR INVALID & N, 2 FOR SAFETY */
ID=ID;
CALL PSEU (ID, 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*/;
CRTB: PROC (PARENT, BSETID, ID1, DATA, MODE, RTN_CODE);

%INCLUDE DBCAT, DBEU;
%INCLUDE EBCT, ESRC, ECRTP, ESUB1;

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

CALL GBCT (BSETID, BINFO, RTN_CODE);

SELECT (MODE);

WHEN ('N') DO: /* NODE 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; /*RTN_CODE 1 IS TARGET OF ARC NOT FND*/
END /*WHEN 'A'*/;

END /*SELECT*/;

/*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*/;
FILE: CHKIN  PLOPT  A1  VM/SP CONVERSATIONAL MONITOR SYSTEM  PAGE 001

CRTN: PROC(MP,OFF_NODE,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*/

%INCLUDE AUPDN;
%INCLUDE ECRTP;

DCL (MP,P)PTR,OFF_NODE(25) BIT (1) CONNECTED, (RTN_CODE,N) FIXED BIN;
DCL DATA BIT (320) VARYING, (SUBSTR,NULL) BUILTIN,
   (PS,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(PS,DATA,ID,RTN_CODE);
IF RTN_CODE='O' THEN RETURN; /*O+ IS RTN CODE FROM CRTP*/

/*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 (UPDN_ARG.IST) MODE='A';
      WHEN (UPDN_ARG.IST) MODE='N';
      OTHERWISE DO;
      RTN_CODE='Q';/*+O IS ILLEGAL OP*/
   RETURN; END;
   END /*SELECT*/;
   CALL CRTP(PS,BS,ID,DATA,MODE,RTN_CODE);
   END /*DO IF*/;
   END /*DO WHILE*/;
IF RTN_CODE='O' THEN
   RTN_CODE=RTN_CODE+20; /*20+ IS FROM SUBROUTINE CALL*/
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
CRT00260
CRT00270
CRT00280
CRT00290
CRT00300
CRT00310
CRT00320
CRT00330
CRT00340
CRT00350
CRT00360
CRT00370
CRT00380
CRT00390
CRIP: PROC(PSETID,DATAD.ID,RTN_CODE);

/* A S-PROC AT THE N-ARY LEVEL TO CREATE A PRIMITIVE ELEMENT WITHIN */
*C A PSET DESIGNATED BY PSETID******/

%INCLUDE DPCAT,DEU;
%INCLUDE EPCT,ESUB1;
DCL (PSETID,ID,RTN_CODE) 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*/
ID=PINFO.L_ID; /*ID1 POINTS TO THE FIRST ELEMENT*/
L_POS=PINFO.L_POS;
L_POS2=PINFO.L_POS2;
I=PINFO.NUMTR;
J=PINFO.PLEN; /*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*/;
DBA:PROC;

/****************************************************************************

 MODULE DESCRIPTION

***************************************************************************/

**** PURPOSE: FOR DBA TO SELECT AMONG THE FOLLOWING DBA

 **** SUBSYSTEMS: DDB (FOR DEFINING BASE DATA), DDBV (FOR

 **** DEFINING VIEW DATA) AND DMB (FOR MANIPULATING

 **** BASE DATA).

 **** PRESENTLY THE DDBV SUBSYSTEM IS NOT AVAILABLE.

***************************************************************************/

**** CALLS PROCEDURES:

 **** INTER-LEVEL T-PROC: NONE;

 **** INTER-LEVEL S-PROC: DDBV, DMB;

 **** INCLUDE EDDB, EDM;

 **** CONTROL STRUCTURE SERVICES: */

***************************************************************************/

DCL NAME CHAR(28),

 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 QUERY (DDQ) ?');

 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 DDBV;

 OTHERWISE PUT SKIP LIST (NAME, ' IS NOT A VALID CMD.');

 END;

 END /* DO WITHIN DD */;

DBA00010
DBA00015
DBA00020
DBA00025
DBA00030
DBA00035
DBA00040
DBA00045
DBA00050
DBA00055
DBA00060
DBA00065
DBA00070
DBA00075
DBA00080
DBA00085
DBA00090
DBA00095
DBA00100
DBA00105
DBA00110
DBA00115
DBA00120
DBA00125
DBA00130
DBA00135
DBA00140
DBA00145
DBA00150
DBA00155
DBA00160
DBA00165
DBA00170
DBA00175
DBA00180
DBA00185
DBA00190
DBA00195
DBA00200
DBA00205
DBA00210
DBA00215
DBA00220
DBA00225
DBA00230
DBA00235
DBA00240
DBA00245
DBA00250
DBA00255
DBA00260
DBA00265
DBA00270
DBA00275
DBA00280
DBA00285
DBA00290
DBA00295
DBA00300
DBA00305
DBA00310
DBA00315
DBA00320
DBA00325
DBA00330
DBA00335
DBA00340
DBA00345
DBA00350
DBA00355
DBA00360
DBA00365
DBA00370
DBA00375
DBA00380
DBA00385
DBA00390
DBA00395
DBA00400
DBA00405
DBA00410
DBA00415
DBA00420
DBA00425
DBA00430
DBA00435
DBA00440
DBA00445
DBA00450
DBA00455
DBA00460
DBA00465
DBA00470
DBA00475
DBA00480
DBA00485
DBA00490
DBA00495
DBA00500
DBA00505
DBA00510
DBA00515
DBA00520
DBA00525
DBA00530
DBA00535
DBA00540
DBA00545
DBA00550
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*/;
FILE: Libd
PLIPT: A1
VM/SP CONVERSATIONAL MONITOR SYSTEM

DDB:PROC;
/*---------------------------------------------------------------*/
*             MODULE DESCRIPTION                                *
*---------------------------------------------------------------*/
* PURPOSE: THIS MODULE IS THE INTERFACE ROUTINE USED BY THE      *
* DBA TO DEFINE THE BASE DATA IN THE ENTITY NETWORK MODEL.     *
*---------------------------------------------------------------*/
* METHOD: THIS MODULE MAKES USE OF THE PANEL MANAGER AVAILABLE ON*
* CMS TO INTERACT WITH THE USER BY DISPLAYING THE PREDESIGNED    *
* PANEL FOR USER TO ENTER DATA DEFINITION.                     *
*---------------------------------------------------------------*/
** INPUT PARAMETERS: AS SHOWN IN THE PANEL;                    *
**---------------------------------------------------------------*/
** OUTPUT PARAMETERS: AS SHOWN IN DEFDE_ARG & DEFA_ARG, WHICH    *
** ARE DATA STRUCTURES FOR COMMUNICATING WITH MODULE DEFE/DEFA.  *
**---------------------------------------------------------------*/
** CALLS PROCEDURES:                                          *
**---------------------------------------------------------------*/
** INTER-LEVEL T-PROC: DEFDE,DEFA;                             *
** /%INCLUDE ADEFE,ADEFA; /* ARGUMENT DCLS---------------------*/
** INTRA-LEVEL T-PROC: NONE;                                   *
** INTRA-LEVEL S-PROC: NONE;                                   *
** CONTROL STRUCTURE,PANEL MANAGEMENT & DEBUGGING SERVICES:    *
** /%INCLUDE SERVICE,FODEBUG,EUDPLI; */                      *
**---------------------------------------------------------------*/
DCL 1 TEMP, /*USED AS TEMPORARY STORAGE FOR USER INPUT BEFORE CONV*/
  2 MLEN CHAR (2),
  2 MAX CHAR (6),
  2 MIN CHAR (6),
  2 FUNC CHAR (3);
DCL 1 UNLDLIST,
  2 UNLOAD (9) PTR,
  2 FENCE BIT (32) INIT ((32)'1'B);
DCL 1 LLDIST,
  2 LD PTR,
  2 FENCE BIT (32) INIT ((32)'1'B);
DCL RTN_CODE FIXED BIN (15);
DCL (P,T,P,RP,LP) PTR, RSTATUS BIT (8);
DCL ADDR BUILTIN;
DCL NULL BUILTIN;
DCL CURSOR FIXED BIN;
DCL PNAME CHAR (8);
DCL 1 ARG LIST BASED (P),
  2 LEN FIXED BIN,
  2 PTR PTR;
DCL ENAME1 CHAR (8); /*USED TO HOLD PARENT ENTITYNAME*/
DCL CMD CHAR (8);
/*BEGIN PROCESSING*/
PUT SKIP LIST ('--BASE DATA DEFINITION SESSION--');
PUT SKIP LIST ('YOU MAY DEFINE NEW ENTITY SETS OR ADD ATTRIBUTES TO EXISTING');
FILE: DDB  PLIOPT  A1  VM/SP CONVERSATIONAL MONITOR SYSTEM  PAGE 002

STING 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=0; /*SET TEMP & ROOT PTRS*/

PNAME='PANCS'; CURSOR=1; LD = ADDR (DEFE_ARG.NAME);

DISPLAY PNAME(PNAME) RSTATUS (RSTATUS) UDLLIST (ULDLIST);

IF DEFE_ARG.NAME='(B)' THEN GO TO RECYCLE_DEFE; /*NO MORE NEW ESETS*/

ENNAME=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.RTN_CODE =0 THEN DO:

CALL DEFEMSG (TP->DEFE_RTN.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, ENNAME);

GO TO ATTR; /*START DEFINE ATTRIBUTE LOOP*/

END /* WHEN NEW*/;

WHEN ('OLD') DO; /*OLD ENTITY SET*/

PNAME='PANCS'; CURSOR=1; LD=ADDR(ENNAME);

DISPLAY PNAME (PNAME) RSTATUS (RSTATUS) UDLLIST (ULDLIST);

IF ENNAME='(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 /* SELECT*/;

ATTR: DO WHILE ('1'B); /*ATTRIBUTE DEF LOOP*/

STRING(TMP)=REPEAT(' ',100); /*INITIALIZE UNLOAD VAR TO BLANKS*/

ALLOCATE DEFA_ARG;

RP=P;
UNLOAD (1) = ADDR(RP->DEFE_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;
PD\LPLAY PNAME(PNAME) RSTATUS (RSTATUS) UDL\LST (UNDL\LST)
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 DEFAMSG (-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 DEFAMSG (-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 DEFAMSG (-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 DEFAMSG (-4); /* ILLEGAL ATTRIBUTE TYPE */
    GO TO RECYCLE.DEFA;
    END /* OTHERWISE */;
FILE: DDB PLOPT A1 VM/SP CONVERSATIONAL MONITOR SYSTEM PAGE 004

END /* SELECT */
DDB01660
DDB01670

SELECT (TEMP.FUNC); /* CHECK FUNCTION TYPE */
DDB01680
WHEN ("1:1") DEFARG.FUNC="S";
DDB01690
WHEN ("M:1") DEFARG.FUNC="M";
DDB01700
WHEN ("KEY") DEFARG.FUNC="K";
DDB01710
WHEN (" " ) DEFARG.FUNC="M"; /* DEFAULT TO M */
DDB01720
OTHERWISE DO;
DDB01730
    CALL DEFAMSQ (-5); /* ILLEGAL FUNCTION TYPE */
DDB01740
    GO TO RECYCLE.DEFA;
DDB01750
    END;
DDB01760
    END; /* SELECT */
DDB01770
/*PASS MSG TO DEFA*/
DDB01780
CALL TCALL ("DEFA",1,RP,TP);
DDB01800
CALL SVC3 (RP); /*FREE CALL ARG LIST*/
DDB01810
RTN_CODE=TP->DEFA_RTN.RTN_CODE;
DDB01820
CALL SVC3 (TP);
DDB01830
IF RTN_CODE == 0 THEN DO;
DDB01840
    CALL DEFAMSQ (RTN_CODE);
DDB01850
    GO TO RECYCLE.DEFA;
DDB01860
    END; /*RTN_CODE IF*/
DDB01870
ELSE CALL CONFIRM (2,DEFARG.NAME);
DDB01880
RECYCLE.DEFA:
DDB01890
END /*ATTR DEF LOOP DO WHILE 'I'B*/;
DDB01900
DDB01940
RECYCLE.DEFE:
DDB01950
END /*ENTITY SET DEF LOOP*/;
DDB01960
DDB01970
DDB01980
DDB01990
DDB02000

********** SUBROUTINE FOR CONFIRM MESSAGE AND ERROR MESSAGE **********
DDB02010
CONFIRM: PROC (CODE,NAMES);
DDB02020
DCL CODE FIXED BIN (31);
DDB02030
DCL NAME CHAR(8);
DDB02040
SELECT (CODE);
DDB02050
WHEN (1) PUT SKIP EDIT ('ENTITY SET ',NAME ,'DEFINED') (A,A,A);
DDB02060
WHEN (2) PUT SKIP EDIT ('ATTRIBUTE ',NAME,' DEFINED') (A,A,A);
DDB02070
OTHERWISE;
DDB02080
END /* SELECT */;
DDB02090
CALL PROMPT;
DDB02100
END /*CONFIRM */;
DDB02110
DDB02120

***********************************************************************
DDB02130
DEFEMSG: PROC (CODE);
DDB02140
DCL CODE FIXED BIN (31);
DDB02150
PUT SKIP LIST ('ILLEGAL ENTITY SET NAME: DEFINITION IGNORED.');
DDB02160
CALL PROMPT;
DDB02170
END /*DEFEMSG */;
DDB02180
DDB02190

****************************************************************************/
DDB02200
FILE: L9  PLOPT A1  VM/SP CONVERSATIONAL MONITOR SYSTEM  PAGE 005

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 (5) PUT SKIP LIST ('ILLEGAL ENTITY SET NAME.');
  WHEN (1) PUT SKIP LIST ('DUPLICATE ATTRIBUTE NAME.');
  WHEN (7) PUT SKIP LIST ('UNKNOWN DOMAIN ENTITY SET NAME');
  OTHERWISE:
    PUT SKIP LIST ('ILLEGAL ATTRIBUTE DEFINITION: DEFINITION IGNORED.');
CALL PROMPT;
END /*DEFAMSG */;

END /*SELECT */;
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.
*****

*/--- INCLUDE ASHWE, SERVICE, ELEX; */
DCL ENAME CHAR (B);
DCL NUM_TOKEN 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.BlANS FIXED BIN INIT (1):
DCL FUNC1 CHAR (3):
DCL NULL_BUILTIN, P PTR;

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

DO WHILE ('Y'='B');
   PUT SKIP LIST ('ENTITY SET NAME? USE * IF ALL ENTITY SET NAMES ARE
ESIRED. ');)
      CALL LEX (NUM_TOKEN, TOKEN, NO.BLANKS, RTN_CODE);
      IF NUM_TOKEN=0 THEN LEAVE; /* GET OUT */
      ENAME = TOKEN.0(1);
      ALLOCATE SHWE_ARG;
      SHWE_ARG.NAME=ENAME;
      IF ENAME = '*' THEN GO TO SEND; /* WANT ALL */

END:
LOOP:
   PUT SKIP LIST ('ATTRIBUTE NAMES? SEPERATE BY COMMAS. USE * IF ALL AT
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.0(1),1,1)="*" /* ALL */
         THEN SHWE_ARG.NUM_NAME2=999;
      ELSE DO;
         SHWE_ARG.NUM_NAME2=NUM_TOKEN;
         DO I = 1 TO NUM_TOKEN;
      DDQQ0010
      DDQQ0020
      DDQQ0030
      DDQQ0040
      DDQQ0050
      DDQQ0060
      DDQQ0070
      DDQQ0080
      DDQQ0090
      DDQQ0100
      DDQQ0110
      DDQQ0120
      DDQQ0130
      DDQQ0140
      DDQQ0150
      DDQQ0160
      DDQQ0170
      DDQQ0180
      DDQQ0190
      DDQQ0200
      DDQQ0210
      DDQQ0220
      DDQQ0230
      DDQQ0240
      DDQQ0250
      DDQQ0260
      DDQQ0270
      DDQQ0280
      DDQQ0290
      DDQQ0300
      DDQQ0310
      DDQQ0320
      DDQQ0330
      DDQQ0340
      DDQQ0350
      DDQQ0360
      DDQQ0370
      DDQQ0380
      DDQQ0390
      DDQQ0400
      DDQQ0410
      DDQQ0420
      DDQQ0430
      DDQQ0440
      DDQQ0450
      DDQQ0460
      DDQQ0470
      DDQQ0480
      DDQQ0490
      DDQQ0500
      DDQQ0510
      DDQQ0520
      DDQQ0530
      DDQQ0540
      DDQQ0550
SHWE_ARG_NAME2(I) * TOKEN_D(I);
END;
END;
SEND;
PUT LIST ('*** DDO TO CALL TCALL 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 **0 THEN DO; /* ERROR RETURN FROM SHWE */
   CALL SHWENSG (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 NUMENAME;
   IF MOD(I,8) = 1 THEN PUT SKIP EDIT (SHWE_RTN2.ENAME(I))(A(8));
   ELSE PUT SKIP EDIT (SHWE_RTN2.ENAME(I)) (COL((I-1)*9+1),A(8));
END;
CALL SVC3 (P);
END */ IF ENAME = '**';
ELSE DO;
   /* SKIP LIST ('ENTITY SET NAME', ENAME); */
   FIELD 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 VTYP = ' ';  
      SELECT (FUNC);
      WHEN ('$') FUNC1='1:1';
      WHEN ('') FUNC1='KEY';
      WHEN ('M') FUNC1='M:1';
      OTHERWISE;
      END */ SELECT /;
   /* PUT SKIP EDIT (ENAME,FUNC1,TYPE,SHWE_RTN1.DATA.ENAME,
   VTYP) (X(6),A(8),X(7),A(3),
   X(5),A(1),X(2),A(8),X(6),A(11));
   IF TYPE = 'V' THEN DO;
      PUT EDIT (MLEN) (COL(52),F(5));
      IF VTYP = '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  PLIOPT  A1  VM/SP CONVERSATIONAL MONITOR SYSTEM  PAGE 003

END:
END; /* DO WHILE */

/*********************************************/
SHOWMSG: PROC(CODE); /* INTERNAL MSG SUBROUTINE */
DCL CODE FIXED BIN;

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(I));

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

END /* DDQ */;
FILE: Defa

PLIOPT  A1

VM/SP CONVERSATIONAL MONITOR SYSTEM

PAGE 001

DEFa:PROC;
DEF00010
DEF00020

/* A T-PROC AT THE ENTITY LEVEL FOR DEFINING ATTRIBUTES*/
DEF00030
DEF00040

%INCLUDE ADEFA, ARETE, AUPDE, SERVICE, FDEBUG, ADEFE;
DEF00050
%INCLUDE DECAT; /*CATALOGUE TEMPLATES*/
DEF00060
%INCLUDE EGECT, EDGET, EFPI1, EFPI2;
DEF00070
DEF00080

DCL (P, MP, TP, RP, RT, LP) PTR, NULL BUILTIN;
DEF00090
DCL (ENAME, ANAME) CHAR (8), RTN_CODE FIXED BIN;
DEF00100
DCL (PSETID1, PSETID2, BSETID) BIT (32);
DEF00110
DCL BIT_STRING BIT (232) BASED; /*AINFO IS 19 BYTES LONG
ENCODED STRING*/
DEF00120
DCL (STRING, UNSPEC) BUILTIN;
DEF00130

/*BEGIN*/
DEF00140
CALL TBEG (PROC_ADDR, MP);
DEF00150
ALLOCATE DEFA_RTN SET (RTP);
DEF00160
DEF00180
AINFO-MP->DEFA_ARG, BY NAME;
DEF00190
ENAME-MP->DEFA_ARG.ENAME1; /*PARENT ENTITY NAME*/
DEF00210
ANAME-MP->DEFA_ARG_NAME; /*ATTRIBUTE NAME*/
DEF00220
DEF00230
/*CHECK PARENT ENTITY SET*/
DEF00240
CALL GECT (ENAME, EINFO, RTN_CODE);
DEF00250
IF RTN_CODE='0' THEN DO;
DEF00260
RTP->DEFA_RTN.RTN_CODE=5; /*IS PROBLEM WITH PARENT ENTITY*/
DEF00270
CALL TRN(=PROC_ADDR, RTP);
DEF00280
RETURN; /*RETURNS*/
DEF00290
END;
DEF00300
DEF00310
DECLARE PSETID1=EINFO;
DEF00320
DEF00330
/*CHECK IF ATTRIBUTE NAME DUPLICATED*/
DEF00340
CALL GACT (ANAME, ENAME, AINFO, RTN_CODE);
DEF00350
IF RTN_CODE='2' THEN DO; /*FROM GACT IS NO SUCH ATTRIBUTE*/
DEF00360
RTP->DEFA_RTN.RTN_CODE=1; /*IS DUPLICATE ATTRIBUTE NAME*/
DEF00370
CALL TRN(=PROC_ADDR, RTP); /*RETURNS*/
DEF00380
RETURN;
DEF00390
END;
DEF00400
DEF00410
/*HANDLING TARGET PSET*/
DEF00420
IF AINFO_TYPE='V' THEN DO; /*MUST CREATE TARGET PSET*/
DEF00430
IF AINFO_VTYPE='N' THEN DO; /*NUMERIC*/
DEF00440
/*IF FDEBUG.LEV1 THEN PUT SKIP LIST ('DEFA: CREATING V PSET'); */
DEF00450
CALL DF11 ('N', 'V', RTN_CODE, PSETID2);
DEF00460
END;
DEF00470
ELSE CALL DF11 (AINFO_VTYPE, AINFO_MLEN, 'V', RTN_CODE, PSETID2);
DEF00480
END;
DEF00490
DEF00500
*/
/*ENTITY TYPE, CHECK TO SEE IF EXISTS ALREADY*/
DEF00510
A: CALL GECT (AINFO.ENAME, EINFO, RTN_CODE);
DEF00520
SELECT (RTN_CODE);
DEF00530
WHEN (2) DO; /*MEANS TARGET ENTITY SET DOES NOT EXIST YET*/
DEF00540
/* IN THIS VERSION THIS MEANS AN ERROR; FUTURE VERSIONS*/
DEF00550
WILL Optionally DEFINES THE TARGET ENTITY SET */
RTP->DEFA_RTN.RTN_CODE=7; /** 7 IS UNKNOWN TARGET ESET *//*
CALL TRTN (PROC_ADDR, RTP);
RETURN;
END /* WHEN 2*/;

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

OTHERWISE DO: /*ILLEGAL TARGET ESET*/
RTP->DEFA_RTN.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 DBT1 (AINFO.FUNC, PSETID1, PSETID2, RTN_CODE, BSETID); D
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; /*ASET NODE*/
LP->UPDE_ARG.PTR=P; LP=P;
UPDE_ARG.NAME='A*ASET ';
UPDE_ARG.OP=UPDEOP.IST;
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.CDATA=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.IST;
UPDE_ARG.PARENT=1;
UPDE_ARG.DLEN=8;
UPDE_ARG.CDATA=ANAME;
UPDE_ARG.NODE = 4;
ALLOCATE UPDE_ARG; /*AINFO NODE*/
LP->UPDE_ARG.PTR=P; LP=P;
UPDE_ARG.NAME="A*AINFO ";
UPDE_ARG.OP=UPDEOP.IST;
UPDE_ARG.NODE = 5;
UPDE_ARG.PARENT=1;
UPDE_ARG.OLEN=20;
UPDE_ARG.CDATA=ADD indifferent BLOCK TO BIT_STRING;
CALL UPDE_ARG:
CALL TCALL ("UPDE",1,RP,TP);
/*CALL UPDE*/
/*RETURN*/
IF TP->UPDE_RTN.RTN_CODE=0 THEN
   RTP->DEF_RTN.RTN_CODE=6; /*0 IS PROBLEM IN UPDATING ENTITY CAT.*/
/*RETURN*/
CALL SVC3 (RP); DEF01280
CALL SVC3 (TP); DEF01290
CALL TRN( PROC ADOP, RTP); DEF01300
RETURN;
END /*DEF*/
DEFE:PROC;

 /******************************************************************************
 * MODE DESCRIPTION
 * PURPOSE: THIS ROUTINE ENABLES DEFINITION OF AN ENTITY SET;
 * IT IS PAIRED WITH MODULE DEFA TO COMPLETE DO OF THE
 * ENTITY NETWORK IN THE BASE DATABASE
 ******************************************************************************/

 METHOD: 1. CALLS UPDE TO INSERT THE

 ENTITY NAME INTO THE 'ESET' ENTITY; IF ERROR RETURN,
 THEN DUPLICATED ESET NAME IS DETECTED AND THE CALLER
 SO ADVISED;

 2. CALLS DEFP (THROUGH SBR DFP1) TO DEFINE A PSET

 CORRESPONDING TO THIS ENTITY SET;

 3. CALLS UPDE AGAIN TO INSERT THE PSETID

 INTO 'ESET' ENTITY SET;

 4. SET UP RETURN MSG FOR CALLER;

 INPUT PARAMETERS: AS INDICATED BY DEFE_ARG;

 OUTPUT PARAMETERS: AS INDICATED BY DEFE_RTN;

 CALLING ARG LIST :

 ** INCLUDE ADEFE;

 ** CALLS PROCEDURES:

 ** INTER-LEVEL T-PROC: DEFP (THROUGH SBR DFP1)

 ** INTRA-LEVEL T-PROC: UPDE

 ** INCLUDE AUPE;

 ** INTRA-LEVEL S-PROC: DFP1_GECT

 ** INCLUDE EDFP1_GECT;

 ** INCLUDE CTRL STRUCTURE, PANEL MANAGER AND DEBUGGING FACILITIES;

 ** INCLUDE SERVICE, DDEBUG;

 /******************************************************************************/

 DCL (P, TP, RP, TP1, RTP) PTR;
 DCL RTN_CODE FIXED BIN;
 DCL NAME CHAR (8), PSETID BIT (32);
 DCL NULL BUILTIN;
 DCL EINFO BIT (32);
 />BEGIN PROCESSING;

 CALL TBEG (PROC_ADDR, P);
 NAME = P -> DEFE_ARG_NAME;

 >/ALLOCATE DEFE_RTN/;

 ALLOCATE DEFE_RTN_SET (RTP);
 RTP -> DEFE_RTN_RTN_CODE = 0; /* INITIALIZE RTN CODE TO 0.K.*/

 >/CHECK IF NAME DUPLICATED BY CALLING GECT*/

 CALL GECT (NAME, EINFO, RTN_CODE);
 IF RTN_CODE = 2 THEN DO; /* NAME ALREADY EXISTS */

 RTN_CODE = 1; /* ILLEGAL DEFE*/
GO TO RTN;
END;

/*CALL SUBROUTINE TO CALL DEFP*/
CALL DFPI ("*", 0, 'E', RTN_CODE, PSETID);
IF RTN_CODE ^=O THEN DO;
   IF FDEBUG.LEVEL THEN PUT SKIP LIST ("JUST RETURNED FROM
      DFPI CALL. RTN_CODE SET TO'.RTN_CODE);
   RTP->DEFE_RTN.RTN_CODE+2; /*ILLEGAL DEF*/
   GO TO RTN;
   END;

/*CALL UPDE TO CREATE ECAT ENTRY FOR THIS ENTITY*/
/*BY CREATING AN ENTITY IN THE E*ESET ENTITY SET*/
ALLOCATE UPDE_ARG; RP=IP; /*ROOT PTR*/
UPDE_ARG.NAME='E*ESET '
UPDE_ARG.OP=UPDEOP.CRT;
ALLOCATE UPDE_ARG; /*2ND TOKEN*/
RP->UPDE_ARG.PTR=TP; TP=IP;
UPDE_ARG.NAME='A*INFO '
UPDE_ARG.PARENT=1;
UPDE_ARG.NODE=2;
UPDE_ARG.DLEN=4;
UNSPEC(UPDE_ARG.CDATA)=PSETID; UPDE_ARG.OP=UPDEOP.LIST;
ALLOCATE UPDE_ARG; /*3RD TOKEN*/
TP->UPDE_ARG.PTR=RP;
UPDE_ARG.NAME='A*ENAE '
UPDE_ARG.PARENT=1;
UPDE_ARG.NODE=3;
UPDE_ARG.DLEN=8;
UPDE_ARG.CDATA=NAME;
UPDE_ARG.OP=UPDEOP.IST;
CALL TCALL ('UPDE',1,RP,TP);

CALL SVC3 (RP); /*FREE CALLING ARG LIST*/
/*CHECK RTN_CODE*/
IF TP->UPDE_RTN.RTN_CODE ^=O THEN DO;
   IF FDEBUG.LEVEL 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.LEVEL THEN PUT SKIP LIST ("DEFE RETURNS");*/
END /*DEFE*/;
DEFP PROC;

/* T-PROC AT THE N-ARY LEVEL FOR DEFINING PRIMITIVE SETS*/

%INCLUDE SERVICE, ADEFP, DPCAT, DNAKEY;

%INCLUDE EMPST, ECRTP;

DCL (P, AP, RTP) PTR, (RTN_CODE, I) 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 (RTN);

RTN->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=PINFO_TYP.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='O' THEN RTP->DEFP_RTN.RTN_CODE=RTN_CODE;

ELSE RTP->DEFP_RTN.PSETID=ID;

CALL TRIN(PROC_ADDR, RTP);

RETURN;

END /*DEFP*/;
DEL:PROC:

/*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, SEUPR,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, SEUPR, RTN_CODE); /* SET SEU INVAL FLAG */
END;

RETURN;
END /*DEL*/;
DELB: PROC (PARENT, BSETID, ID1, MODE, RTN_CODE);
/** A S-PROC AT THE N-ARY LEVEL TO DELETE A BINARY ASSOCIATION+/
%INCLUDE DBEU, DBCAT;
%INCLUDE 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 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 RTN_CODE
   FROM Delp *********************/
IF RTN_CODE <> 20 THEN RETURN;
END /* IF THERE EXISTS... */;
END /* DELB */;
DELP: 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 MODE*/
%INCLUDE AUPTDN;
%INCLUDE EDELB,ESRCH,EDELP;

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

/*DO P=MP REPEAT UPDN_ARG.PTR WHILE(P=*NULL);
PUT SKIP LIST('DELP 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)='0' THEN DO;
   BS=UPDN_ARG.BSSETID;
   SELECT (UPDN_ARG.OP);
   WHEN (UPDN.DEL) CALL DELB (PS,BS,ID,'A',RTN_CODE);
   WHEN (UPDN.DDD) 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 /*DELP*/;
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*/
%INCLUDE DPCAT, DBEU;
%INCLUDE 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=NULL 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(NULL) 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(NULL) 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*/;
FILE: DEL1
PL1OPT A1
VM/SP CONVERSATIONAL MONITOR SYSTEM

DEL1: PROC (ID,RTN_CODE);
/* A S-PROC AT THE LOWEST LEVEL OF N-ARY LEVEL TO SET UP CALL TO
* DEL MODULE AT THE WM LEVEL TO DELETE BUZ DESIGNATED BY ID*/

XINCLUDE 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.RTN_CODE;
CALL SVC3(P);
CALL SVC3(TP);
RETURN;
END /*DEL1*/;
DFB1:PROC(FUNC, PSE4ID1, PSE4ID2, RTN_CODE, BSETID);
/* A 5-PROC AT THE ENTITY LEVEL FOR SETTING UP CALLS TO DEFB*/
%INCLUDE 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;
DFB_ARG.PSETID(1) = PSETID1;
DFB_ARG.PSETID(2) = PSETID2;
DFB_ARG.FUNC = FUNC;
IF FUNC = 'K' THEN DEFB_ARG.FUNC = 'S';
CALL TCALL ('DEFB', 1, P, TP);
RTN_CODE = TP->DEFB RTN.RTN_CODE;
IF RTN_CODE = 0 THEN BSETID = TP->DEFB RTN.BSETID;
CALL SVC3 (P);
CALL SVC3 (TP);
RETURN;
END /*DFB1*/;
DFP1:PROC (PTYPE,PLEN,TYPE,RTN_CODE,PSETID);

//***************************************************************************

* MODULE DESCRIPTION

***************************************************************************

***** PURPOSE: THIS IS THE SUBROUTINE USED AT THE ENTITY LEVEL

***** TO SET UP CALL TO DEFINE ONE PSET.

***** INPUT PARAMETERS: PTYPE, CHAR(4), IS THE DATA TYPE OF THE

***** PSET TO BE DEFINED. PLEN, FIXED BIN, IS THE LENGTH;

***** TYPE, CHAR(4), INDICATES 'E' OR 'V', TO BE USED BY THE

***** INTERNAL ROUTINE IMPP FOR DETERMINING IMP PORTION OF

***** DEFP_ARG;

***************************************************************************

***** OUTPUT PARAMETERS:

***** RTN_CODE AND PSETID AS OBTAINED FROM DEFP CALL;

***************************************************************************

***** CALLS PROCEDURES:

***** INTER-LEVEL T-PROC: DEFP;

%INCLUDE ADEFP;

****** INTRA-LEVEL T-PROC: NONE;

****** INTRA-LEVEL S-PROC: NONE;

****** DFPOPROC240: CONTROL STRUCTURE, PANEL MANAGER AND DEBUGGING FACILITIES:

%INCLUDE SERVICE,FDEBUG;

***************************************************************************

DCL (P,TP,RP)PTR, NULL BUILTIN;

DCL RTN_CODE FIXED BIN;

DCL PTYPE CHAR(4), PLEN FIXED BIN, TYPE CHAR(4);

DCL PSETID BIT(32);

/BEGIN PROCESSING*/

ALLOCATE DEFP_ARG; RP=*; /*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.RTN_CODE;

IF FDEBUG.LEVL THEN DO;

IF DEFP_RTN.RTN_CODE="O THEN DO;

PUT SKIP LIST ('DFP1: DEFP RTN CODE NOT 0, IS',DEFP_RTN.RTN_CODE);

RTN_CODE=0;

ENDIF;

ENDIF;

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*/

DFP00040
/*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.IMP=DEFPHOW.SA; /*STAND ALONE*/
  DEFP_ARG.IMP.NUMPTR=12; /*DEFAULT TO 12*/
END;
ELSE DO;
  DEFP_ARG.IMP.IMP=DEFPHOW.SA; /*ALSO STAND ALONE*/
  DEFP_ARG.IMP.NUMPTR=4; /*DEFAULT TO 4*/
END; /*IMPP*/;
END /*DFPL*/;
FILE: DMB  PLOPT  A1  VM/SP CONVERSATIONAL MONITOR SYSTEM  PAGE 001

DMB: PROC;
DCL CMD CHAR(7);

PUT SKIP LIST(‘--- DATA MANIPULATION SESSION ---’);
DO WHILE ('1' < LEN CMD):
   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', 'C') CALL DMC;
      WHEN ('MODIFY', 'M') CALL DMM;
      WHEN ('DELETE', 'D') CALL DMD;
      WHEN ('QUERY', 'Q') CALL DMQ;
      WHEN ('Q') LEAVE;
   OTHERWISE PUT SKIP LIST(‘ILLEGAL COMMAND, PLEASE USE UPPER CASE’);
   END /*SELECT*/;
END /*DO WHILE*/;
END /*DM*/;
DMC: PROC; /*PROCESSING CREATE COMMANDS*/
%INCLUDE ELEX,ELEX,ENVM1,EUXPE1;
DCL IT. /*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;
DCL MODE FIXED BIN;
DCL N1 FIXED BIN; /*USED TO COMM. WITH VNM*/
DCL 1 PRED(25), /*STRUCTURE TO COMMUNICATE WITH LEX*/
2 OP CHAR (1), /*.*<,**/
2 VALUE CHAR (40),
2 VLEN FIXED BIN;
DO WHILE ('1'); /*DO FOREVER*/
A1:PUT SKIP LIST ('CREATE: 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 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='O' 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='O' THEN DO;
PUT SKIP LIST ('IMPROPER SYNTAX, PLEASE RE-ENTER');
GO TO A;
END;
/*CALL UP VNAME TO CHECK LEGALITY OF NAMES*/
CALL VNM1(T,RTN_CODE,N1); /*VNM1 WILL PRINT ERR MSG IF ANY*/
IF N1=1 THEN GO TO A1; /*ESSET NAME ERROR*/
ELSE IF N1 = 0 THEN GO TO A; /*ATR NAME ERROR*/
/*START ASKING FOR DATA THROUGH SUBROUTINE UPE1*/
OPE='C';
DO I=1 TO 25; OP(I)=N; /*INITIALIZE OPE TO BE 'LIST'*/
END;
PUT SKIP LIST ('ENTER DATA:');
CALL UPE1 (OPE,T.OP,RTN_CODE); /*UPE1 WILL PRINT ERR MSG IF ANY*/
END /*DO WHILE*/;
FILE: DMD  PLIDPT  A1  VM/SP CONVERSATIONAL MONITOR SYSTEM  PAGE 001

DMD: PROC; /*PROCESSING DELETE COMMANDS*/
%INCLUDE ELEM, ELEX, EVMM1, EUPE1;
DCL T, /*DATA STRUCTURE USED TO COMMUNICATE WITH LEX*/
  2 ENAME CHAR(8),
  2 TN FIXED BIN, /*NUMBER OF ATTR*/
  2 T(25),
  3 PARENT FIXED BIN,
  3 NAME CHAR (0);
DCL RTN_CODE FIXED BIN;
DCL (DT(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 VMM1*/
DCL 1 PRED(25); /*STRUCTURE TO COMMUNICATE WITH LEX*/
  2 DP CHAR (1), /* * >, <, */
  2 VALUE CHAR (40),
  2 VLEN FIXED BIN;
DCL CMD CHAR (0);

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

DO WHILE ('YB'); /*DO FOREVER*/
A: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');
  NODE=1; /*SUPPRESS BLANKS*/
  CALL LEX (N, TOKEN, NODE, RTN_CODE); /*GET INPUT LINE THRU LEX*/
  IF RTN_CODE='O THEN DO;
    PUT SKIP LIST ('IMPROPER SYNTAX (LEX). PLEASE RE-ENTER');
    GO TO A;
    END;
  IF N=O THEN LEAVE; /*GET OUT OF DMD*/
  CALL LEX (N, TOKEN, T, T.N, PRED, OPE, RTN_CODE); /*BUILD T STRUCTURE*/
  IF RTN_CODE='O THEN DO;
    PUT SKIP LIST ('IMPROPER SYNTAX, PLEASE RE-ENTER');
    GO TO A;
    END;
  CALL VMME TO CODE FOR LEGALITY OF NAMES*/
  CALL VMME(T, RTN_CODE, N1);
  IF N1<1 THEN GO TO A1; /*SET NAME ERROR */
  ELSE IF N1='O THEN GO TO A; /*ATTR NAME ERROR */
  /*START ASKING FOR DATA THROUGH SUBROUTINE UPE*/
  OPE='D';
FILE: DMD  PLOPT  A1  VM/SP CONVERSATIONAL MONITOR SYSTEM  PAGE 002

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

HELP: PROC; /*** INTERNAL SUBROUTINE **********/**
PUT SKIP LIST ('********** BRIEF EXPLANATION ON HOW TO DO DELETE ****'); DMD00670
PUT SKIP LIST ('AT THE PROMPT FOR IDENTIFIER ATTRIBUTE NAMES PLEASE'); DMD00680
PUT SKIP LIST ('ENTER ATTRIBUTES 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 PROMPT FOR DATA'); DMD00760
PUT SKIP LIST ('********** NOW TRY IT **********'); DMD00770
END /*HELP*/;

END /*DMD*/;
```
DMN: PROC; /*PROCESSING CREATE COMMANDS*/

MODULE DESCRIPTION

PURPOSE:

THIS IS A S-PROC AT THE USER INTERFACE LEVEL, CALLED BY

MODULE DMN, 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 LEX 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.

XINCLUDE ELEX, LEXT, VNM1, UPE1;

DCL 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 NODE 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;

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 ('*8'); /*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 DMN*/

T.ENAME = TOKEN.D(1);
FILE: DMM PLOPT A1 VM/SP CONVERSATIONAL MONITOR SYSTEM PAGE 002

A:PUT SKIP LIST ('ENTER MODIFICATION OPERATORS AND NAMES OF ATTRIBUTES') DUMO0660
S SEPARATED BY COMMAS! DUMO0880
MODE+1: /*SUPPRESS BLANKS*/ DUMO0590
CALL LEX (N, TOKEN, MODE, RTN_CODE); /*GET INPUT LINE THRU LEX*/ DUMO0680
IF RTN_CODE="O THEN DO:
   PUT SKIP LIST ('IMPROPER SYNTAX (LEX), PLEASE RE-ENTER'); DUMO0690
   GO TO A;
   END;
ELSE IF N=O THEN LEAVE; /*GET OUT OF DMC*/ DUMO0640
   END;
CALL LEXT (N,TOKEN,TI,TN,PRED,OP,RTN_CODE); /*BUILD T STRUCTURE*/ DUMO0680
IF RTN_CODE="O THEN DO:
   PUT SKIP LIST ('IMPROPER SYNTAX, PLEASE RE-ENTER'); DUMO0700
   GO TO A;
   END;
/*CALL UP VNME TO CHECK FOR LEGALITY OF NAMES*/ DUMO0730
CALL VNMIIT(RTN_CODE,N1); /*VNMI WILL PRINT ERR MSG IF ANY*/ DUMO0740
IF N1=-1 THEN GO TO A1; /*ESSET NAME ERROR*/ DUMO0780
ELSE IF N1=-O THEN GO TO A; /*ATTR NAME ERROR*/ DUMO0770
/*START ASKING FOR DATA THROUGH SUBROUTINE UPE1*/ DUMO0790
OPE='M';
PUT SKIP LIST ('ENTER DATA:');
/*UPEI WILL PRINT ERR MSG*/ DUMO0820
CALL UPE1 (OPE,T,OP,RTN_CODE);
END /*DO WHILE*/;
RETURN;

HELP: PROC; /**** INTERNAL SUBROUTINE ***********/ DUMO0870
PUT SKIP LIST ('********** BRIEF EXPLANATION ON HOW TO DO MODIFY **********') DUMO0880
****
PUT SKIP LIST ('AT THE PROMPT FOR MODIFICATION OPERATORS AND ATTR.'): DUMO0900
PUT SKIP LIST ('BUTE NAMES PLEASE ENTER THE OPERATOR (INSERT,DELETE, ... )'): DUMO0910
PUT SKIP LIST ('REPLACE') PREFIXED WITH '*' AND NAMES OF ATTRIBUTES'): DUMO0920
PUT SKIP LIST ('THAT WILL BE THIS MODIFIED. YOU SHOULD ALSO PROVIDE'): DUMO0930
PUT SKIP LIST ('A SPECIAL OPERATOR (ID) FOR NAMES OF ATTRIBUTES USED'): DUMO0940
PUT SKIP LIST ('TO IDENTIFY THE ENTITY TO BE MODIFIED. AN EXAMPLES'): DUMO0950
PUT SKIP LIST ('FOLLOWS:'): DUMO0960
PUT SKIP LIST ('TO CHANGE THE DEPARTMENT AN EMPLOYEE (WHSO)'): DUMO0970
PUT SKIP LIST ('EMPLOYEE NUMBER WILL BE GIVEN) WORKS IN TO ANOTHER'): DUMO0980
PUT SKIP LIST ('DEPARTMENT (WHSO DEPARTMENT NUMBER WILL BE GIVEN, )'): DUMO0990
PUT SKIP LIST ('ISSUE THE FOLLOWING MODIFICATION STATEMENT AT THE'): DUMO1000
PUT SKIP LIST ('PROMPT FOR ATTRIBUTE NAMES AND MODIFICATION'): DUMO1010
PUT SKIP LIST ('OPERATORS'): DUMO1020
PUT SKIP LIST ('-ID: EMPNUM, -REP: WORKS IN (DEPTNUM)'): DUMO1030
PUT SKIP LIST ('AND LATER ENTER DATA FOR EMPNUM AND DEPTNUM'): DUMO1040
PUT SKIP LIST ('HELP'): DUMO1050
END /*HELP*/;
END /*DMM*/;
DMQ: PROC;

MISSION STATEMENT

* * *

MODIFIED DESCRIPTION

* * *

PURPOSE: THIS IS A

**T-PROC AT THE USER INTERFACE LEVEL, CALLED BY THE DMB MODULE, TO
**PROCESS QUERY REQUESTS ISSUED BY THE USERS WITHIN A DATA
**MANIPULATION- QUERY SESSION.

***** METHOD:

1. CALLS LEX TO PARSE INPUT STRING INTO TOKENS.

2. CALLS LEX TO BUILD NAME TREE AND PREDICATE TABLE.

3. SET UP RETE CALL TO THE NEXT LEVEL.

4. CALL PRNT TO PRINT RESULT.

*****

CALS PROCEDURES:

INTER-LEVEL T-PROC: RETE.

INTRA-LEVEL T-PROC: NONE.

INTRA-LEVEL S-PROC: LEX, LEX, PRNT

CONTROL STRUCTURE, PANEL MANAGER AND DEBUGGING FACILITIES:

TCALL, SVC3

*******************************************************************************

%INCLUDE ELEX, LEXT, SERVICE, ABETE, EPRT;

DCL T INTEGER;

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 NAME CHAR (8);

DCL RTN_CODE FIXED BIN;

DCL N FIXED BIN; /*THE FOLLOWING USED TO CALL LEX*/

DCL I TOKEN(25),

2 D CHAR (40),

2 L FIXED BIN;

DCL MODE FIXED BIN;

DCL N1 FIXED BIN; /*USED TO COMM. WITH VWW*/

DCL I pred (25); /*PRED AND MOP COMMUNICATE WITH LEXT*/

2 OP CHAR (1), /* =, >, <, */

2 VALUE CHAR (40),

2 VLEN FIXED BIN;

DCL MOP (25) CHAR (1); /* NOT USED IN DMQ */

DCL CMD CHAR (8);

DCL (TP, RP, LP) PTR, NULL BUILTIN;

DCL I FIXED BIN (31);

/*BEGIN PROCESSING*/

PUT SKIP LIST ('- - QUERY SESSION - - ');

PUT SKIP LIST (' TYPE YES IF NEED HELP ');

GET EDIT (CMD) (A(8));
FILE: DMO  PLCOPT A1  VM/SP CONVERSATIONAL MONITOR SYSTEM

IF CMD='YES' THEN CALL HELP;

/* MAIN COURSE */

/* DO WHILE ('1'!=~0 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 DMQ*/
   T.ENAME = TOKEN.O (1);
A1: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<>O THEN DO;
      PUT SKIP LIST ('IMPROPER SYNTAX (LEX), PLEASE RE-ENTER');
      GO TO A1;
   END;
   IF N=0 THEN LEAVE; /*GET OUT OF DMQ*/
   CALL LEXT (N,TOKEN,1,T,1.TN,PRED,MOP,RTN_CODE); /*BUILD T STRUC*/
   IF RTN_CODE<>O THEN DO;
      PUT SKIP LIST ('IMPROPER SYNTAX, PLEASE RE-ENTER');
      GO TO A1;
   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 =O; /*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)=O 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 */;
CALL TCALL ('RETE',1,RP,TP); /* TP -> RETE_RTN */
IF TP->RETE_RTN.RTN_CODE -= 0 /* CHECK RETE RETURN CODE */
THEN DO;
    CALL RETEMSG (TP->RETE_RTN.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);
END /DO WHILE */;

/******************** INTERNAL SUBROUTES **********************/
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 QUERING 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 SUBROUTES **********************/
RETEMIS: PROC (CODE); /* INTERNAL SUBROUTINE FOR RETE ERROR */
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: DUMFSTDV PLOPT A1
VM/SP CONVERSATIONAL Monitor System

FSTV: PROC;
/* A DRIVER PROGRAM USED WHEN FH IS RUN WITHOUT CONTROL STRUCTURE */
/*
-------------------------------
* 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.)

*/

/*INCLUDE EUSER; /*INTRA-LEVEL S-PROC ENTRY DCLS*/
/*INCLUDE AVINI; /*INTER-LEVEL T-PROC ARG LIST*/
/*INCLUDE SERVICE; /*CONTROL STRUCTURE SERVICES*/
/*INCLUDE FOBEG,SH;
DCL SETFH ENTRY;
ON ZERO DIVIDE;
DCL NULL BUILTIN;
DCL RIN,CODE FIXED BIN(15);
DCL (TP,P) POINTER;
DCL (NAME, FNAME) CHAR(8);
DCL SAVESH FILE RECORD BUFFERED SEQUENTIAL ENV
(U BKLSIZE(30016));
DCL OUT BIT(1) INIT ('O'B);

CALL SETFH; /* SETS UP TRACE OPTIONS */
ALLOCATE SH;

/* BEGIN SESSION */
PUT SKIP (2) LIST ('--FUNCTIONAL HIERARCHY STV CONSOLE PROGRAM--');
DO WHILE ('B');
PUT SKIP LIST ('INITIALIZATION: FILE OR NEW:');
GET EDIT (NAME) (A(8));
FILE: DUMFSIV PLOPT  A1
VM/SP CONVERSATIONAL MONITOR SYSTEM

SELECT(NAME);

WHEN ('F', 'FILE', 'F', 'file') DO;
QU='1'B;
PUT SKIP LIST ('FILE NAME?');
GET EDIT (FILENAME) (A(8));
CALL READ (FILENAME); /*READ IS INT SBR TO READ FILE INTO SH*/
CALL NEXTI; /*INT SBR 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; /*REINIT PROCESSOR INIT*/
CALL CINIT; /*BN VIEW PROCESSOR INIT*/
FILENAME=' '(A(8)) ' '; /*NEW IS INT SBR TO SET UP CALL FOR NEW INIT IN NEXT LV*/
CALL NEXTI;
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',I,TP,P);

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

*************** TIMING REPORT OPTION ***************
ON ZERO DIVIDE;
DCL (ETIME,ELD,ECOUNT(50) FIXED BIN(31)) EXT;
DCL TIME BIT(1) EXT;
%INCLUDE FSTRUC;
IF FTIME THEN DO;
PUT EDIT ('LEVEL PRONAME COUNT TOT ELAPSED TIME TOT RUNTIME RUN')
TIME/INVOCATION' (SKIP, A);
PUT EDIT ('------ ------- ------ ---------- ------ --------------')
(SKIP, A);
PUT EDIT ((ARCH, LEVEL(I), ARCH, PRONAME(I), ECOUNT(I), ETIME(I),
ETIME(I) = ELD(I)),
(ETIME(I) = ELD(I))/ECOUNT(I))
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;

PAGE 002
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 DEBUG.LEVO THEN PUT SKIP LIST ('FSTV CALLS VINIT');
CALL TCALL ('VINIT',1,P,TP);
IF DEBUG.LEVO THEN PUT SKIP LIST ('FSTV: VINIT RETURNS');
CALL SVC3(P);
IF TP->VINIT_RTN.RTN_CODE ^= 0 THEN DO;
PUT SKIP LIST ('ERROR IN INITIALIZATION, RESTART');
OUT='0'B; /*REST INIT LOOP*/
END;
CALL SVC3(TP);
END /*NEXTI*/;

**SAVE** PROC(FNAME); /* SAVE EVERYTHING IN STORAGE HIERARCHY*/
DCL FNAME CHAR(B);
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(B);
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;

**RUNIT** PROC;
/* IN THE FUTURE, THIS MODULE Initializes THE RELATIONAL VIEW
SUBSYSTEM; PRESENTLY IT IS NOT AVAILABLE*/
RETURN;
END;
END /*FSTV*/;
LI: 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:
DUMPIT: PROC (STRING, ID, P);
INCLUDE OBEU;
DCL STRING CHAR(*);
DCL ID BIT(32); DCL (P, BP BASED) PTR;
DCL I FIXED BIN;
DCL PAA(40) PTR BASED;
DCL NULL BUILTIN;
%INCLUDE HEX;

PUT SKIP EDIT('***MEMWG: ', 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(1)) - >BP))(X(1), A); END;
PUT EDIT('DLEN=', DLEN)(X(1), A, F(3));
DO I = 1 TO (DLEN+31)/32; PUT EDIT(HEX4(ADDR(DATA) - >PAA(I)))(A); END;
END;
END;
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;

FILE: DUMTBEG PL/IOPT A1
VM/SP CONVERSATIONAL MONITOR SYSTEM

PAGE 001
(SUBRGG):
TCALL:PROC (PROCNAME,1,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; /* TCALL IS DEBUG TRACE BIT */
DCL DTIME BIT(1) EXT; /* TTIME IS TIMING TRACE BIT*/
XINCLUDE FSTRUCT; /* FSTRUCT IS STRUCTURE OF FH */
DCL PROCNAME CHAR(7);

DCL (VINIT,DEFE,RETE,UPDE,DEP,DEFB,RETN) ENTRY;
DCL (UPDN,DEFA, NINIT,VNAME,SHWE) ENTRY;
DCL (NINIT,RET,REP,DEL,CRF,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),EL050),ETIME(50)) FIXED BIN(31) EXT INIT((50);0); STAT_P1=P1;

DO Enum = 1 TO 18; /* FIND LEVEL */
    IF ARCH.PROCNAME(Enum) = PROCNAME THEN LEAVE;
END;
I=ARCH.LEVEL(Enum);

*************** TCALL TRACE **********************************************/

IF TCALL>100 /* IF TCALL TRACE BIT FOR THAT LEVEL IS ON*/
THEN PUT SKIP EDIT ('TCALL: ',PROCNAME)(A,A);

*************** TIMING REPORT **********************************************/

IF DTIME THEN DO;
ALLOCATE LOTIME; LOTIME=0;
BEGTIME=RTIMER; END;

SELECT (PROCNAME):

WHEN ('FSTV ') CALL FSTV;
WHEN ('VINIT ') CALL VINIT;
WHEN ('VNAME ') CALL VNAME;
WHEN ('DEFE ') CALL DEFE;
WHEN ('RETE ') CALL RETE;
WHEN ('UPDE ') CALL UPDE;
WHEN ('DEBP ') CALL DEBF;
WHEN ('SHWE ') CALL SHWE;
WHEN ('DEFB ') CALL DEFB;
WHEN ('RETN ') CALL RETN;
WHEN ('UPDP ') 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;

DUMO00010
DUMO00020
DUMO00030
DUMO00040
DUMO00050
DUMO00060
DUMO00070
DUMO00080
DUMO00090
DUMO00100
DUMO00110
DUMO00120
DUMO00130
DUMO00140
DUMO00150
DUMO00160
DUMO00170
DUMO00180
DUMO00190
DUMO00200
DUMO00210
DUMO00220
DUMO00230
DUMO00240
DUMO00250
DUMO00260
DUMO00270
DUMO00280
DUMO00290
DUMO00300
DUMO00310
DUMO00320
DUMO00330
DUMO00340
DUMO00350
DUMO00360
DUMO00370
DUMO00380
DUMO00390
DUMO00400
DUMO00410
DUMO00420
DUMO00430
DUMO00440
DUMO00450
DUMO00460
DUMO00470
DUMO00480
DUMO00490
DUMO00500
DUMO00510
DUMO00520
DUMO00530
DUMO00540
DUMO00550
FILE: DUMTCALL PlOJPT 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=rtimer-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 fcall(i) then
put edit(' *** TCALL: END ', procname)(a,a);
end /* tcall*/;
FILE: DUMTRN PL1OPT 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 ADDRS OF THE CALLING PROCEDURE TO RETURN
   ** THE RETURN MSG TO; RTP POINTS TO THE RETURN ARGUMENT LIST ***/

%INCLUDE FDEBUG;
DCL (P,RTP) PTR, RTN_CODE FIXED BIN;
DCL 1 T BASED (P);
2 LEN FIXED BIN (15);
DCL STAT_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
FILE: EBOT PLIOPT A1 VM/SP CONVERSATIONAL MONITOR SYSTEM PAGE 001

EBOT:PROC;

/* A 5-PROC AT THE ENTITY LEVEL USED BY VINIT MODULE TO BOOTSTRAP
** ENTITY CATALOGUE ENTRIES *********************************************/

INCLUDE AUPOND, SERVICE, OKEY, 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+ESSET'),

2 MAX FIXED BIN (31),

2 MIN FIXED BIN (31),

2 MLEN FIXED BIN INIT (8, 4, 0, 0, 0),

2 FUNC CHAR (1) INIT ('K', 'S', 'W', 'S', 'S'),

2 TYPE CHAR (1) INIT ('V', 'v', 'v', 'E', 'v', 'v'),

2 VTYPE CHAR (1) INIT ('C', 'B', 'C', 'C', 'B');

DCL BIT_STRING BIT (232) BASED: /*BIT STRING FORM OF AINFO*/

DCL BAINFO (5) BIT (32);

DCL SEQ (5) FIXED BIN INIT (2, 3, 1, 5, 6);

DCL ADDR BUILTIN;

/*BEGIN PROCESSING*/

/*SET UP BINFO*/

DO I=1 TO 5:

MAINFO.BSETID (I) = KEY (I*6);

MAINFO.PSETID (I) = KEY (SEQ (I));

BINFO (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.PSET);

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 (1);

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+ESSET ');

ELSE UPDN_ARG.DATA = UNSPEC ('E+ASET ');

END /WHEN 1/;

WHEN (2) DO:

UPDN_ARG.PSETID = KEY (KY.PINFO);

UPDN_ARG.BSETID = KEY (KY.BINFO);

EB00010

EB00020

EB00030

EB00040

EB00050

EB00060

EB00070

EB00080

EB00090

EB00100

EB00110

EB00120

EB00130

EB00140

EB00150

EB00160

EB00170

EB00180

EB00190

EB00200

EB00210

EB00220

EB00230

EB00240

EB00250

EB00260

EB00270

EB00280

EB00290

EB00300

EB00310

EB00320

EB00330

EB00340

EB00350

EB00360

EB00370

EB00380

EB00390

EB00400

EB00410

EB00420

EB00430

EB00440

EB00450

EB00460

EB00470

EB00480

EB00490

EB00500

EB00510

EB00520

EB00530

EB00540

EB00550
FILE: L.JT   PLIOPT A1   VM/SP CONVERSATIONAL MONITOR SYSTEM

UPON_ARG.DLEN=4;
IF J=1 THEN UPDN_ARG.DATA=KEY(KY.PASET);
ELSE UPDN_ARG.DATA=KEY(KY.PASET);
END /*WHEN 2*/;
END /*SELECT I*/;
END /*DO I*/;
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.PSID=KEY(KY.PASET);
UPDN_ARG.OP=UPDNOP.CRT;
UPDN_ARG.DLEN=0;
DO I=1 TO 4;
ALLOCATE UPDN_ARG;
LP->UPDN_ARG.PR=T; LP=P;
UPDN_ARG.NODE=I+1; /*ASSIGN NODE NUMBER*/
SELECT (I);
WHEN (1) DO; /*NAME NODE*/
UPDN_ARG.PSID=KEY(KY.PNAME);
UPDN_ARG.PARENT=1;
UPDN_ARG.BSID=KEY(KY.BNAME);
UPDN_ARG.OP=UPDNOP.CI;
UPDN_ARG.DLEN=8;
SELECT (J);
WHEN (1) UPDN_ARG.DATA=UNSPEC('A*NAME ');
WHEN (2) UPDN_ARG.DATA=UNSPEC('A*EINFO ');
WHEN (3) UPDN_ARG.DATA=UNSPEC('A*ET ');
WHEN (4) UPDN_ARG.DATA=UNSPEC('A*NAME ');
WHEN (5) UPDN_ARG.DATA=UNSPEC('A*EINFO ');
END /*SELECT J*/;
END /*WHEN 1*/;
WHEN (2) DO; /*SET NODE*/
UPDN_ARG.PSID=KEY(KY.PASET);
UPDN_ARG.PARENT=1;
UPDN_ARG.BSID=KEY(KY.BSET);
UPDN_ARG.OP=UPDNOP.IS;
UPDN_ARG.DLEN=0;
END /*WHEN 2*/;
WHEN(3) DO; /*EASE NAME*/
UPDN_ARG.PSID=KEY(KY.PNAME);
UPDN_ARG.PARENT=3;
UPDN_ARG.BSID=KEY(KY.BNAME);
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*/;
L1:PROC; /* A DUMMY ROUTINE TO SIMULATE ENTIRE SH-STV */
L1 00010
%INCLUDE USERS,VPX,ARG,SH;
L1 00020
ARGPTR = VP.WAIT.MSG;
L1 00030
INDEX = ARG.VIRTUAL_ADDRESS/B + 1;
L1 00040
IF ARG.REQ_TYPE = 'READ'
L1 00050
THEN ARG.DATA = PACKET(INDEX);
L1 00060
ELSE PACKET(INDEX) = ARG.DATA;
L1 00070
CALL SEND (0, ARG.VPID, ARG.BOXID, 'S', 20, ARGPTR);
L1 00080
CALL FINISH;
L1 00090
RETURN;
L1 00100
END;
L1 00110
L1 00120
FILE: FSTV  PLIOPT  A1  VW/SP CONVERSATIONAL MONITOR SYSTEM  PAGE 001

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.
**** NEW INIT, IT CALLS SBSR TO INITIALIZE CAT FILES OF THIS LEVEL
**** AND PERCOLATES THE NEW INIT DOWN.
**************************************************************************************

**** CALLS PROCEDURES: RINIT, CNINIT, VINIT, VSAVE, USER;
**** (INTRA-LEVEL S-PROCS: RINIT, CNINIT: 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 VVL 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 SVRS,VSIV,TX,TC/CONTROL STRUCTURE SERVICES*/
%INCLUDE FDBUG;
DCL NULL BUILTIN;
DCL RTH_CODE FIXED BIN(15);
DCL (TP,P) POINTER;
DCL (NAME,FNAME) (CHAR(8));
DCL (SETFH) ENTRY;
/* BEGIN SESSION */
PUT SKIP (2) LIST ('''FUNCTIONAL HIERARCHY STV CONSOLE PROGRAM''');
CALL SETFH: /* SETTING FSTV TRACE OPTIONS */
CALL NEXTI: /* INITIALIZE SYSTEM CATALOGE VST */
CALL TCALL('USER', (P,T); /* THEN START USER PROCESSING */
/* P AND TP IRRELEVANT */

/* WHEN USER QUIT USER SESSION */
******************************************************************************
******** TIMING REPORT OPTION **************
******** ON ZERO/DOVE;
DCL (ETIME,ELD,ECCOUNT)(50) FIXED BIN(31)) EXT;
DCL FTIME BIT(1) EXT;
%INCLUDE FSTRUC;
IF FTIME THEN DO;
PUT EDIT ('''LEVEL, PROCNAM, COUNT, TOT ELAPSED TIME, TOT RUN TIME RUNFST')
TIME/INVOCATION'(SKIP,A);--
PUT EDIT ('''--') (SKIP,A);--
**************************************************************************************

FST0010
FST0020
FST0030
FST0040
FST0050
FST0060
FST0070
FST0080
FST0090
FST0100
FST0110
FST0120
FST0130
FST0140
FST0150
FST0160
FST0170
FST0180
FST0190
FST0200
FST0210
FST0220
FST0230
FST0240
FST0250
FST0260
FST0270
FST0280
FST0290
FST0300
FST0310
FST0320
FST0330
FST0340
FST0350
FST0360
FST0370
FST0380
FST0390
FST0400
FST0410
FST0420
FST0430
FST0440
FST0450
FST0460
FST0470
FST0480
FST0490
FST0500
FST0510
FST0520
FST0530
FST0540
FST0550
FILE: FSIV  PLIOPT A1  VM/SP CONVERSATIONAL Monitor SYSTEM  PAGE 002

PUT ED: T({ARCH.LEVEL(I),ARCH.PROCNAME(I),ECOUNT(I),ETIME(I)},
    (ETIME(I)-EL0(I)),
    (ETIME(I)-EL0(I))/ECOUNT(I))
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 BBR FOR PASSING INIT DOWN NEXT LEVEL*/
GETI:  PUT LIST ('REGENERATE(NEW) OR OLD DATA BASE(FILE):');
GET EDIT (FILENAME) (A(B));
SELECT (FILENAME);
    WHEN ('N', 'NEW', 'n', 'new') FILENAME = '';
    WHEN ('F', 'FILE', 'f', 'file') FILENAME = 'F';
OTHERWISE GO TO GETI; END /*SELECT */;
ALLOCATE VINIT_ARG; /*SETUP ARG LIST*/
VINIT_ARG PTR= NULL;
VINIT_ARG.FNAME= NULL;
IF FDEBUG.LEV0 THEN PUT SKIP LIST ('FSIV CALLS VINIT');
CALL TCALL ('VINIT', 1, P, TP);
IF FDEBUG.LEV0 THEN PUT SKIP LIST ('FSIV: 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 /*FSIV*/;
GACT: PROC (ANAME, ENAME, AINFO, RTN_CODE);

/***** A S-PROC IN THE ENTITY LEVEL RESPONSIBLE FOR RETRIEVING ATTR-
**** BUTE CAT ENTRY GIVEN ATTRIB NAME AND ENTITY NAME **********/
GAC00030
GAC00040

INCLUDE DECAT, DKEY, ARETN, SERVICE, FDEBUG;
GAC00050

DCL (ANAME, ENAME) CHAR (8);
DCL (P, RP, LP, TP, RTP) PTR,(NULL_ADDR) BUILTIN;
DCL BIT_STRING BIT (252) BASED;
DCL RTN_CODE FIXED BIN, I FIXED BIN;
DCL UNSPEC BUILTIN, STRING BUILTIN;
DCL 00110
DCL 00120
DCL 00130

//BEGIN PROCESSING*/
RTN_CODE=0; /*INIT TO O.K.*/
GAC00140
GAC00150

//RETN TREE ROOT NODE IS ASET*/
I+1; /*NO PREDICATE*/
GAC00160
GAC00170
ALLOCATE RETN_ARG; RP,LP,P;
GAC00180
RETN_ARG.PSETID$KEY(KY.PASET);
RETN_ARG.GET=RETNGET.ANY;
RETN_ARG.N+/*NO PREDICATE*/
GAC00190
GAC00200
GAC00210
GAC00220
GAC00230
GAC00240
GAC00250
GAC00260
GAC00270
GAC00280
GAC00290
GAC00300
GAC00310
GAC00320
GAC00330
GAC00340
GAC00350
GAC00360
GAC00370
GAC00380
GAC00390

/ANAME NODE*/
I+1; /*I PREDICATE*/
ALLOCATE RETN_ARG;
LP->RETN_ARG.PTR=P; LP=P;
RETN_ARG.PSETID$KEY(KY.PASET);
RETN_ARG.NODE=2;
RETN_ARG.PARENT=1;
RETN_ARG.BSETID$KEY(KY.BNAME);
RETN_ARG.GET=RETNGET.NO;
RETN_ARG.N+/*I PREDICATE*/
GAC00400
GAC00410
GAC00420
GAC00430
GAC00440
GAC00450
GAC00460
GAC00470
GAC00480
GAC00490
GAC00500
GAC00510
GAC00520
GAC00530
GAC00540
GAC00550

/*ESET NODE*/
I+0;
ALLOCATE RETN_ARG;
LP->RETN_ARG.PTR=P; LP=P;
RETN_ARG.PSETID$KEY(KY.PASET);
RETN_ARG.NODE=3;
RETN_ARG.PARENT=1;
RETN_ARG.BSETID$KEY(KY.BSET);
RETN_ARG.GET=RETNGET.NO;
RETN_ARG.N+/*NO PREDICATE*/
GAC00560
GAC00570
GAC00580
GAC00590
GAC00600
GAC00610
GAC00620
GAC00630
GAC00640
GAC00650
GAC00660

/ENAME NODE*/
I+1; /*I 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=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)=B;
RETN_ARG.PRED.DATA=UNSPEC (ENAME);

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

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

/*DECODE RETURN*/
IF TP->RETN_RTN.RTNCODE=*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;
GAC00950
GAC00960
GAC00970
RETURN: CALL SVC3 (RP); CALL SVC3 (TP);
RETURN: /*RETURNS*/
END /*GACT*/;
FILE: GECF
PLDT: A1
VM/SP CONVERSATIONAL MONITOR SYSTEM

000000000000000000089

P- N,
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
*:*
FILE: GECT  PLOIPT  AI  VM/SP CONVERSATIONAL MONITOR SYSTEM  PAGE 001

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*/

%INCLUDE DECAT, ARETN, DKEY, SERVICE, FDEBUG;

DCL RTN_CODE FIXED BIN, ENAME CHAR (8);
DCL (P,T,P,RP,L,P) PTR, 1 FIXED BIN, NULL BUILTIN;
DCL UNSPEC BUILTIN;

/*BEGIN PROCESSING*/
RTN_CODE=0; /*INIT TO 0.K.*/
/*ALLOCATE ROOT RTN TREE - ESET*/
I=0; /*NO PRECICATE*/
ALLOCATE RETN_ARG; LP,RP,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.N=1; /*1 PRECICATE*/
RETN_ARG.CM(1)=1;
RETN_ARG.OP(1)=RETNOP.EO;
RETN_ARG.OLN(1)=0;
RETN_ARG.DAT(1)=UNSPEC (ENAME);

/*NEXT NODE -- ENAME**/
I=1; /*NO PRECICATE*/
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.N=1; /*1 PRECICATE*/
RETN_ARG.CM(1)=1;
RETN_ARG.OP(1)=RETNOP.EO;
RETN_ARG.OLN(1)=0;
RETN_ARG.DAT(1)=UNSPEC (ENAME);

/*NEXT NODE -- EINFO **/
I=0; /*NO PRECICATE*/
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; /*0 PRECICATE*/

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

/*CHECK RETURNS*/
IF TP->RETN.RTN.RTN_CODE=-O THEN DO;
RTN_CODE=1; /*PROBLEM IN RETRIEVAL*/
CALL SVC3 (RP);
CALL SVC3 (TP);
RETURN;

GEC00010
GEC00020
GEC00030
GEC00040
GEC00050
GEC00060
GEC00070
GEC00080
GEC00090
GEC01000
GEC01100
GEC01200
GEC01300
GEC01400
GEC01500
GEC02000
GEC02100
GEC02200
GEC02300
GEC02400
GEC02500
GEC02600
GEC02700
GEC02800
GEC02900
GEC03000
GEC03100
GEC03200
GEC03300
GEC03400
GEC03500
GEC03600
GEC03700
GEC03800
GEC03900
GEC04000
GEC04100
GEC04200
GEC04300
GEC04400
GEC04500
GEC04600
GEC04700
GEC04800
GEC04900
GEC05000
GEC05100
GEC05200
GEC05300
GEC05400
GEC05500
FILE: GECT   PLIOPT A1

VM/SP CONVERSATIONAL MONITOR SYSTEM

END;

IF TP->RETN_RTN.N.OP=1 THEN DO;
  RTN_CODE=2; /*NO SUCH ENTITY*/
  /* IF FDEBUG.LEVEL THEN PUT SKIP LIST ('GECT: NO SUCH ENTITY'); */
  CALL SVC3 (TP);
  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*/;
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,1) FIXED BIN(31), (UNSPEC, SUBSTR) BUILTIN;

%INCLUDE EPAKT;
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: GSEU   PLOPT A1   VM/SP CONVERSATIONAL Monitor System  PAGE 001

GSEU: PROC (ID, SEUPTR, RTN_CODE);

**** 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 ***************/

%INCLUDE DSEU1, EPAKT;
DCL DATA CHAR(8), VA FIXED BIN(31), TP PTR, DTS CHAR(5);
DCL (I, RTN_CODE, L, 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');  /* OBTAIN FIRST 8 BYTES OF THIS SEU */
SEUPTR = ADDR (DATA);  /* IMPOSE SEU STRUCTURE ON DATA */
IF SEU.INVAL = 'N' THEN DO;
   RTN_CODE = 1;
   RETURN;
END;
I*SEU.N;
   /* ESTABLISH FIRST 8 BYTES OF SEU */
ALLOC SEU SET (TP);  GSE00220
TP->SEU.INVAL = SEU.INVAL;
SEUPTR = TP;
L = MIN (5, SEU.N);  GSE00250
DATA = SUBSTR (DATA, 4, L);
DTS = SUBSTR (DATA, 1, L);
SUBSTR (SEU.DATA, 1, L) = DTS;
   /* ESTABLISH THE REST OF SEU */
L = (SEU.N + 2)/8;  GSE00300
   /* L = # OF PACKETS IN THIS SEU - 1 */
DO I = 1 TO L;  GSE00220
   /* GET ALL PACKETS */
   CALL PAKT (VA+I*8, DATA, 'R');  GSE00220
   L1 = I*8-2; L2 = MIN(0, SEU.N-L1+1);
   DATA = SUBSTR (DATA, 1, L2);
   SUBSTR (SEU.DATA, L1, L2) = DATA;
END:
RETURN;
END;
FILE: LEX  PLOOPT A1  VM/SP CONVERSATIONAL MONITOR SYSTEM

LEX: PROC (I, 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 STRUNG 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 1 TOKEN (25),
  2 D CHAR(40),
  2 L FIXED BIN;
DCL RTN_CODE FIXED BIN;
DCL 1 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(BO),
  POS FIXED BIN(15);
DCL (VERIFY, SUBSTR, INDEX, LENGTH) BUILTIN;

/*****************************************************************************/

/* START SUBROUTINE */
RTN_CODE=0;

/* GET INPUT STRING */
STR = ''; 

LOOP: GET EDIT (LINE) (A(BO));
POS = INDEX(LINE,'/');
IF POS = 0 THEN DO:
  STR = STR || SUBSTR(LINE,1,POS-1);
  GO TO LOOP;
END:
STR = STR || LINE;
/* FIX UP LINE */
DO I = 1 TO LENGTH STR BY -1 WHILE (SUBSTR (STR, I, I) = ' ')
\[\]
END;
\[\]
STR = SUBSTR (STR, I, I) \[\] \
BEGIN:
\[\]
/* SUPPRESS BLANKS */
IF MODE = 1 THEN DO:
\[\]
DO I = 1 TO LENGTH STR;
\[\]
/* SKIIP BLANKS IN QUOTES */
IF SUBSTR (STR, I, I) = "" THEN DO WHILE ("" B);
\[\]
I = I + 1;
\[\]
IF I > LENGTH STR THEN DO:
\[\]
RTN_CODE = 2; /* ILLEGAL SYNTAX */
\[\]
RETURN;
\[\]
END;
\[\]
IF SUBSTR (STR, I, I) = "" THEN LEAVE;
\[\]
END:
\[\]
ELSE
\[\]
IF SUBSTR (STR, I, I) = "" THEN DO:
\[\]
STR1 = SUBSTR (STR, I - 1);
\[\]
STR2 = SUBSTR (STR, I + 1);
\[\]
STR = STR1 || STR2;
\[\]
I = I + 1; /* FIX UP I */
\[\]
END;
\[\]
END:
\[\]
/* LEX STARTS */
\[\]
I = 0;
\[\]
/* EXTRACT TOKENS */
DO WHILE (VERIFY (STR, ",") = 0); /* DO TILL END OF STRING */
\[\]
LAST = INDEX (STR, ",") - 1; /* LOOKING FOR DELIMITER */
\[\]
IF LAST > -1 & LAST < LAST THEN LAST = LAST;
\[\]
ELSE IF LAST < 0 THEN LAST = LAST;
\[\]
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 */;
FILE: LEXT  PLIOPT  A1  VM/SP CONVERSATIONAL MONITOR SYSTEM

LEXT: PROC(N,TOKEN,T1,PN,PRED,MOP,RTN_CODE);

/*****
MODULE DESCRIPTION
/*****
/***** PURPOSE:
S-PROC AT USER INTERFACE LEVEL WHICH ACCEPTS TOKENS
IN A MATRIX AND IDENTIFIES THE TREE STRUCTURE AND THE PREDICATE
VALUES IN THESE TOKENS; IT ALSO DISCRIMINATES MODIFICATION OPERATORS
FROM REGULAR ATTRIBUTE NAMES AND STORE THEM IN STRUCTURE MOP */
/*****
/***** METHOD:
THE HIERARCHICAL STRUCTURE OF NAMES IS IDENTIFIED BY
NESTED PARENTHESES. 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-NAMED TOKENS: MODIFICATION OPERATOR
(MOP) AND PREDICATE TOKEN. MOP IS DISTINGUISHED FROM REGULAR
NAME TOKENS BY ITS SPECIAL STARTING CHAR 'C'. TOKEN THAT
CONTAIN SPECIAL COMPARISON 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 IDENTIFY MODIFICATION OP IDENTIFIED
FOR CORRESPONDING NAME IN T1.
RTN_CODE: 0 - 0.K.
/*****
1,2: MISMATCH OF PARENTHESES
3: ILLEGAL MODIFICATION OPERATOR
/*****
/*****
DCL (N,TN,RTN_CODE,I,J,K) FIXED BIN;
DCL 1 T1 (25), /* ENTITY AND ATTRIBUTE NAMES TREE STRUCTURE*/
2 PARENT FIXED BIN,
2 ANAME CHAR(8);
DCL 1 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 (26),
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) BUILDTIN;

/* BEGIN PROCESSING */

RTN_CODE=0;
TN=0; /* INIT TI 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; /* MISMATCH OF PARENTH */
            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;

/* *************************************************************************/
FILE: LEXT PLOOPT A1

VM/SP CONVERSATIONAL MONITOR SYSTEM

TIP: PROC(K); /*INTERNAL SUBROUTINE*/
DCL K FIXED BIN;
DCL OP(3) CHAR(1) INIT('>','<','*');
DCL VALUE CHAR(40) VARYING; DCL CMD CHAR(B);
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;
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;
FILE: LEXT  PLIOPT  A1  VM/SP CONVERSATIONAL MONITOR SYSTEM  PAGE 004

END;
ELSE VALUE='';
PREV.VALUE(TN)=VALUE;
PREV.VLEN(TN)=LENGTH(VALUE);
/* REGISTER NAME*/
T1.ANAME(TN)=DATA;
T1.PARENT(TN)=ST(STP);
MP(TN)=CURRENT_MOP;
RETURN;
END /*T1*/;
END /*LEX*/;
MINIT: PROC;

 meille 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 IS STORED AT THE
    FIRST PACKET OF SH IS INITIALIZED TO BE 16 BY CALLING PAKT.
 2. IF IT IS INIT REQUEST AND IT IS FILE INIT (I.E., SH ALREADY
    READY EXISTS WITH DATA) THE KEY FOR THE UPPER LEVEL
    (4 BYTES) IS READ IN FROM VIRTUAL MEMORY LOCATION 8
    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.

 INCLUDE DSEIU, EPACK, SERVICE, FDEBUG, AMINIT;
 DCL KEY BIT (32);
 DCL (P:RTOP) PTR, (NAME, DATA) CHAR (8), OP CHAR (1), DT4 CHAR (4);
 DCL (NULL, BIN, UNSPEC, SUBSTR) BUILTIN;
 CALL TBEG (Proc Address, P);
 ALLOCATE MINIT RTN SET (RTP);
 OP=MINIT_ARG.OP;
 NAME=MINIT_ARG.FNAME;
 SELECT (OP);
 WHEN ('1') 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');
         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 */
RTP->MINIT_RTN.KEY=KEY;
END;

WHEN ('S') DO; /*PROCESS STORE KEY REQUEST*/
CALL PAKT(0,DATA,'R'); /*GET 1ST PACKET IN THE SH*/
UNSPEC(DT4) = MINIT_ARG.KEY;
SUBSTR(DATA,5,4) = DT4; /*UPDATE IT WITH KEY*/
CALL PAKT(0,DATA,'W');
END /*WHEN S*/;

END /*SELECT*/;
RTN: CALL TRTN( PROC_ADDR,RTP);
RETURN;
END /*MINIT*/;
FILE: MODO
PLIOPT AI
VM/SP CONVERSATIONAL MONITOR SYSTEM

MODO: 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*/
%INCLUDE AUPDN;
%INCLUDE ECRTB,EDELB,EREP,ESECH;

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('MODO 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" 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 /*MODO*/; /*RTN_CODE IS SET TO RTN FROM SUBROUTINE CALLS*/
FILE: MPST  PLIOPT  A1

VM/SP CONVERSATIONAL MONITOR SYSTEM

PAGE 001

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, I) FIXED BIN;
DCL SUBSTR BUILTIN;

DG I=92 TO 1 BY -1;
IF SUBSTR(MAP, I, 1)='0'B THEN DO;
    SUBSTR(MAP, I, 1)='1'B;
    POS=I;
    RETURN;
END;

POS=0; /*NOT AVAILABLE*/
RETURN;
END /*MPST*/;
NINIT: PROC;

/* A T-PROC AT THE M-ARY LEVEL FOR INITIATING THE CATALOGUE STRUCTURE
** AT THE M-ARY LEVEL *********/

%INCLUDE ANIMI, SERVICE, FDEBUG, ANIMI, DNPKEY, DPCAT, DBEU, ADEFP;
%INCLUDE ESUBI, EMPST;
DCL (P, RP, TP, TP, MP, RP) PTR, (NULL, STRING, UNSPEC, ADDR) BUILTIN;
DCL AKY BIT (32) STATIC, NKEY(3) BIT (32);
DCL MAP BIT (32), POS FIXED BIN, (1, J) FIXED BIN, ID BIT (32);
DCL RN_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.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='(B)' THEN DO; /*FILE INIT*/
AKEY=NINIT_RTN.KEY; /*OBTAIN KEY FROM NINIT RETURN***/
CALL SVC3 (P);
CALL RETH(AKEY, P, RN_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 TRHN(PROC_ADDR, RTP); /*RETURNS*/
RETURN;
END; /*FILE INIT*/

IF MP->NINIT_ARG.FNAME='(B)' THEN DO; /*NEW INIT*/
/*INIT PCAT*/
CALL SVC3(P);
**INFO.BMPTR=4;
**INFO.PLEN=18; /*18 BYTES LONG*/
**INFO.PTYPE='B';
**INFO.LTYPE=INFO.LTYPE.LL; /*LINK LST*/
**INFO.L_ID=UNSPEC(NULL); /*INIT TO NULL*/
/**CALL UP WPST TO OBTAIN L.POS'S*/
MAP='00011111111111111111111111'B;
CALL WPST (MAP, POS);
INFO.L_POS=POS;
CALL WPST(MAP, POS);
INFO.L_POS2=POS;
INFO.MAP=MAP;
FILE: Minit  Pliopt  A1

VM/SP CONVERSATIONAL MONITOR SYSTEM

/*STREAMLINE PINFO INTO BIT STRING*/
/*CREATE THIS PCAT CAT ENTRY*/
I=4; /*4 IDS*/
J=18+B;
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=1;
CALL TCALL (DEFP,1, TP, TP);
BCATID=TP->DEFP_RTN.PSETID;
CALL SVC3 (TP);
CALL SVC3 (TP);

NKEY(1)=PCATID;
NKEY(2)=BCATID;
NKEY(3)=UNSPEC (NULL); /*RESERVED*/
I=1; J=96; ALLOCATE BEU:
BEU.DATA=STRING(NKEY);
CALL CRT1(AXEY,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 TRNN (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 REP1(AKEY,P,RTN_CODE);
ELSE CALL CRT1(AKEY,P,RTN_CODE);
FREE BEU;
CALL TRNN (PROC_ADDR, RTP);
END /*WHEN 'S'*/;

PAGE 002
PARK PROC(VA,DATA,RW);
/** THIS IS A SUBROUTINE AT THE MM LEVEL WHICH RETRIEVES AN 8-BYTE
 * PACKET ('\"ATA\"') 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 OSH STV ***************************/

/** DUE TO INCOMPATIBILITY OF FORMAT OF \"ARG\" TO THE SET OF SERVICE
 * ROUTINES COMMONLY USED BY OTHER PH MODULES, CONTROL STRUCTURE
 * IS DIRECTLY ACCESSED FROM PARK, AND TIMING CODE IS DUPLICATED
 * HERE TO ACCUMULATE STATISTICS OF MODULE L1 WHICH IS NORMALLY
 * PERFORMED BY SERVICE ROUTINE TCALL ******************************/

#include users,vpx,pfsvc,arg;

DCL VA FIXED BIN(31), DATA CHAR(8), RW CHAR(1);

J=8; ALLOC ARG; /* PREPARE ARG TO COMM. WITH OSH */
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;

ALLOC PF.SVC; /* PREPARE FOR SEND PROTOCOL */
PF.SVC Svc = \"L1\";
PF.SVC_PTR = ARG PTR;

/******************** TIMING REPORT OPTION *********************/
DCL FTIME BIT(1) EXT; /* FTIME IS TIMING REPORT OPTION */
DCL (BEGIN TIME,ENUM,LOTIME CTL) FIXED BIN(31);
DCL RTIMER ENTRY RETURNS(FIXED BIN(31));
DCL (E-COUNT(50), ELO(50),ETIME(5C)) FIXED BIN(31) EXT INIT((50)0);
IF FTIME THEN DO;
ALLOCATE LOTIME; LOTIME = O;
BEGIN TIME = RTIMER;
END;
/******************** TIMING REPORT OPTION *********************/
CALL SEND (1,0,1, \"S\",20,PT.SVC);
CALL WAIT(1);

/******************** TIMING REPORT OPTION *********************/
IF FTIME THEN DO;
BEGIN TIME = RTIMER - BEGIN TIME;
ENUM = 19; /* L1 IS THE 19TH PROC NAME TO BE TRACED */
ETIME(ENUM) =ETIME(ENUM) + BEGIN TIME; /* 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) = O THEN LOTIME = LOTIME + BEGIN TIME;
END;
/******************** TIMING REPORT OPTION *********************/
ARG PTR = VP.WAIT MSG; /* OBTAIN RETURN ARG */
IF RW = \"R\" THEN DATA = ARG.DATA;
FREE ARG;
/* 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 */

#include 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,11,12,13) 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=nil);
    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=nil);
    nn=rete_rtn1.node;
    t{mlen(nn)} = rete_rtn1.dlen; /* note in current version, mlen a
dolen 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:
        tok(j) = ' ';  
        select (t{parent(j)}):
            when (last) do:
                token = token||' ('||t{name(j)});
            else
                stp = stp + 1;
                st(stp) = t{parent(j)};
            end;
            when (st(stp)) token = token||')' '||t{name(j)};
            otherwise do:
            od 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.WLEN(J)+3;
        IF SUM > 80 THEN DO:
            L=L+1;
            T.POS(J) = 1;
            SUM = T.WLEN(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(I), 1, 3) = ' |
    DO P=P2->RETE_RTN_PTR->RETE_RTN1_PTR REPEAT REPEP_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(I), 1, 3) = ' |
            END;
            END
ELSE IF RETE_RTN1.DLEN ^= 0
       THEN DO:
           I1+T.LINE(NODE);
           I2+T.POS(NODE);
           I3+T.MLEN(NODE);
           SUBSTRSTR(I1),I2,I3)
           SUBSTR(RETE_RTN1.CO DATA,1,RETE_RTN1.DLEN);
           SUBSTRSTR(I1),I2+I3,3) = ' ;
       END;
END /* DO P */:
   DO I = 1 TO L; /* WRITE OUT THE LAST OCCURRENCE */
       PUT SKIP LIST STR1));
       END;
END /* PRNT */:
FILE: PSEU PLIOPT A1 VM/SP CONVERSATIONAL MONITOR SYSTEM PAGE 001

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 PACKETS AND CALLS PAKT TO WRITE THEM INTO DSH */

%INCLUDE DSEU1, EPAKT;

DCL DATA CHAR(8), VA FIXED BIN(31), (TP, BP) PTR;
DCL (1, 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'); /* WRITE FIRST 8 BYTES OF THIS SEU */
L = (SEU.N + 2)/8;
/* L = # OF PACKETS IN THIS SEU - 1 */
DO I = 1 TO L;
/* PUT ALL PACKETS */
L1 = 1*8-2; L2 = MIN(BK, SEU.N-L1+1);
DATA = SUBSTR (SEU.DATA, L1, L2);
CALL PAKT (VA+I*8, DATA, 'W');
END;

RETURN;
END;
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.RTN_CODE =0;
CALL GSEU (REP_ARG.ID, SEUPTR, RTN_CODE); /*GET SEU DESIGNATED BY ID*/
IF RTN_CODE ^= 0
   THEN RTP->REP_RTN.RTN_CODE = 1; /* ILLEGAL SEU */
ELSE DO;
   SEU.N=REP_ARG.N/8;
   UNSPEC(SEU.DATA) = REP_ARG.DATA;
   CALL PSEU (REP_ARG.ID, SEUPTR, RTN_CODE);
   END /*ELSE DO*/;
CALL TRTN(PROC_ADDR, RTP);
RETURN;
END /*REP*/;
FILE: REPB  PLIOPT  A1  VM/SP CONVERSATIONAL MONITOR SYSTEM  PAGE 001

REPB: PROC (PARENT,BSETID, ID1,DATA,MODE,RTN_CODE);

%INCLUDE DBCAT, DBEU;
%INCLUDE EBC,ESRC,ESRT,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; /* WHEN 'S' */;
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; /*RTN_CODE 1 IS TARGET OF ARC NOT FND*/
  END /*WHEN 'A'*/;
END /*SELECT*/;

/*ENTER ID2 INTO ID1 ID ARRAY*/
BEU.ID_ARRAY(BINFO.POS)=ID2;
CALL REPI (ID1,P,RTN_CODE);
RTN_CODE=0;
END /*REPB*/;
FILE: REPP  PLIOPT  AI  VM/SP CONVERSATIONAL MONITOR SYSTEM  PAGE 001

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 */

REPO00010
REPO00020
REPO00030
REPO00040
REPO00050
REPO00060
REPO00070
REPO00080
REPO00090
REPO0100
REPO0110
REPO0120
REPO0130
REPO0140
REPO0150
REPO0160
REPO0170
REPO0180
REPO0190
FILE: REP1  PLIOPT  A1  VM/SP CONVERSATIONAL MONITOR SYSTEM  PAGE 001

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*/

XINCLUDE 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 ('REPL',1,P1,BP);

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

RTN_CODE=BP->REP_RTN.RTN_CODE;
CALL SVC3(P1); CALL SVC3(BP);
RETURN;
END;

223
RET: PROC;

/* A T-PROC AT THE MM LEVEL FOR RETRIEVAL OF SEU GIVEN ITS ID */
/* FOR INTEGRATED VERSION: MAKE USE OF SUBROUTINE GSEU (GET SEU) */

%INCLUDE ARET, SERVICE, DSEU1, 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.RTN_CODE = 1;
ELSE DB;
   RTP->RET_RTN.N = SEU.N=0;
   RTP->RET_RTN.DATA = UNSPEC(SEU.DATA);
END;

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

RETO0010
RETO0020
RETO0030
RETO0040
RETO0050
RETO0060
RETO0070
RETO0080
RETO0090
RETO1000
RETO1010
RETO1020
RETO1030
RETO1040
RETO1050
RETO1060
RETO1070
RETO1080
RETO1090
RETO1100
RETO1110
RETO1120
RETO1130
RETO1140
RETO1150
RETO1160
RETO1170
RETO1180
RETO1190
RETO1200
RETO1210
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*/

%INCLUDE OBCAT,ESUB1,DEU;
%INCLUDE 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*/;
RETE:PROC(
    /*-PROC TO PROCESS RETE **/
    %INCLUDE ARETE,DEBUG,SERVICE,ARETN;
    %INCLUDE DECAT;
    %INCLUDE EGECT,EGACT,EVTPE;

    DCL (P,TP,RP,LP,TP1,RTP,PP) PTR, NULL BUILTIN;
    DCL(RTN_CODE,1,J) FIXED BIN, NUM F XED BIN (31);
    DCL BDATA BIT (320), CDATA CHAR (40), DLLEN FIXED BIN;
    DCL TO_BIT FIXED BIN INIT (1), TO_CHAR FIXED BIN INIT (2);
    DCL (ENAME,ANAME,ENAME1) CHAR (8);
    DCL 1 MAINFO (25), /*AN ARRAY OF AINFO TO STORE RETRIEVED ACAT EN*/
       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 IBEG (P,PROC ADDR,P);
    ALLOCATE RETE_RTN SET (RTP);
    RTP->RETE_RTN RTN_CODE=0; /*INIT TO O.K.*/
    ENAME->RETE_ARG_NAME;
    MAINFO.ENAME(1)=ENAME; /*INIT AINFO ARRAY'S FIRST TOKEN*/
    **ALLOCATE ROOT NODE**/
    CALL GECT ((ENAME,AINFO,RTN_CODE)); /*GECT GETS ENTITY CAT ENTRY*/
    IF RTN_CODE=0 THEN DO;
      RTP->RETE_RTN RTN_CODE=100; /*100 IS NO SUCH ENTITY*/
      CALL TRN (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=AINFO;
    /*FOLLOW RETE TREE **/
    DO P=RETE_ARG, PTR REPEAT P=RETE_ARG, PTR WHILE (P=NULL); RET00470
      NODE=RETE_ARG, NODE;
      ENAME=MAINF0, ENAME(RETE_ARG, PARENT);
      ANAME=RETE_ARG, NAME;
      CALL GACT ((ANAME,ENAME1, AINFO, RTN_CODE)); /*GACT GETS ATTR CAT EN*/
      IF RTN_CODE=0 THEN DO;
        RTP->RETE_RTN, RTN_CODE=NODE=100; /*100 IS NO SUCH ATTRIBUTE**/
        CALL SVC3 (RP); /*FREE UP RETN ALLOCATION UP TO NOW*/
        CALL TRN (PROC_ADDR, RTP); /***RETURNS*/
      END.
    END.
// VERIFY PREDICATE DATA ***
IF RETE_ARG.N<>0 THEN DO:
CDATA=RETRE_ARG.PRED.CDATA(1);
DLEN=RETRE_ARG.PRED.DLEN(1);
CALL VTEP (AINFO,TO_BIT,BDATA,CDATA,DLEN,RTN_CODE);
IF RTN_CODE<>0 THEN DO; /* PROBLEMS*/
RTP->RETRE_RTN.RTN_CODE=NODE; /*++ IS ILLEGAL NODE DATA*/
CALL SVC3(RP); /* FREE UP RETN ALLOCATION UP TO NOW*/
CALL TRTN (PROC_ADDR, RTP); /* RETURNS*/
RETURN;
END;

/* REPLACE ANAME WITH PSETID AND BSETID AND FIX UP DATA***/
I=RETRE_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=RETRE_ARG, BY NAME;
TP=RETN_ARG.CBTP=1; 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=RETRE_ARG.PRED.DLEN(1)=DLEN;
TP=RETRE_ARG.PRED.DATA(1)=DATA;
END; /*DO P=RETRE_ARG.PTR. . .*/;
/* CALL UP RETN ***/
CALL TCALL ('RETN',1,RP,TP);

/* CHECK RETURNS*/
I=TP->RETRE_RTN.RTN_CODE;
IF 1<>0 THEN DO:
RTP->RETRE_RTN.RTN_CODE=1+200; /*200+ IS RTN CODE FROM RETN*/
CALL SVC3 (RP);
CALL SVC3 (TP);
CALL TRTN (PROC_ADDR, RTP);
RETURN;
END;
RTP->RETRE_RTN.N=TP->RETRE_RTN.N; /*COPY*/
LP=RP;

/*COPY RETN_RTN INTO RETE_RTN WHILE FIXING UP DATA ***/
DO P=TP->RETRE_RTN.PTR REPEAT P=RETRE_RTN1.PTR WHILE (P<>NULL);
/* INCLUDE HEX;*/
/* PUT SKIP LIST (* (RETE) RETN_RTN1. RETN_RTN1.HEX(RETE_RTN1.PTR));*/
ALLOCATE RETE_RTN1 SET (TP1);
LP->RETE_RTN1.PTR=TP1; LP=TP1;
I=TP1->RETE_RTN1.CBTP; J=TP1->RETE_RTN1.LEN; PP=TP1->RETE_RTN1.PTR; RET01110
TP1->RETE_RTN1=RETE_RTN1, BY NAME;/*COPY*/
RE01120
TP1->RETE_RTN1.CBTP=I; TP1->RETE_RTN1.LEN=J; TP1->RETE_RTN1.PTR=PP; RET01140
/*FIX UP DATA*/
RE01150
NODE=RETN_RTN1.NODE;
RE01160
IF NODE = 1 THEN DO; /*ROOT HAS NO DATA*/
RE01170
TP1->RETE_RTN1.DLEN=0; GO TO END; END;
RE01180
AINFO=MAININFO(NODE);
RE01190
SELECT(AINFO.TYPE);
RE01200
WHEN ('V') DO;
RE01210
BDATA=RETN_RTN1.DATA;
RE01220
DLEN=RETN_RTN1.DLEN;
RE01230
call VTPE (AINFO, TO_CHAR, BDATA, CDATA, DLEN, RTN_CODE);
RE01240
TP1->RETE_RTN1.DLEN=DLEN;
RE01250
TP1->RETE_RTN1.CDATA=CDATA;
RE01260
END;
RE01270
WHEN ('E')
RE01280
TP1->RETE_RTN1.DLEN=0; /*IGNORE DATA*/
RE01290
END /*SELECT*/;
RE01300
PUT SKIP LIST('RETE_RTN1', TP1->RETE_RTN1, HEX(TP1->RETE_RTN1.PTR)); RET01310
/* END: END;
RE01320
CALL SVC3 (RP); /*FREE UP ARG AND RTN LIST FROM RETN*/
RE01330
CALL SVC3 (TP);
RE01340
CALL TRTN (PROC_ADDR, RTP); /*RETURNS WITH ARGUMENTS*/
RE01350
DO P=RTP REPEAT RETE_RTN.PTR WHILE (P=NULL);
RE01360
END /*RETE*/;
RE01380
FILE: RETN  PLOOPT A1  VM/SP CONVERSATIONAL MONITOR SYSTEM

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          RET00140
   THE DATA BASE; IT FIRST VISITS THE ROOT NODE AND           RET00150
   OBTAINS THE FIRST OCCURRENCE OF THE ROOT NODE; IT         RET00160
   THEN VERIFIES THE OCCURRENCE AGAINST THE PREDICATE        RET00170
   (IF ANY) TO DECIDE WHETHER THIS OCCURRENCE IS VALID;      RET00180
   IF YES, THEN CONTINUE; OTHERWISE DISCARD IT AND           RET00190
   GET THE NEXT ROOT OCCURRENCE UNTIL ALL ROOT OCCURRENCES   RET00200
   ARE EXAMINED.                                             RET00210

2. AFTER A ROOT OCCURRENCE IS ESTABLISHED, THIS ROUTINE        RET00220
   FOLLOWS THE RETN TREE TO ESTABLISH OCCURRENCE OF ITS       RET00230
   ASSOCIATED ELEMENTS AND VERIFY THEIR CONTENT AGAINST      RET00240
   THEIR PREDICATES. IF ANY MISMATCH IS ENCONTERED, THE       RET00250
   CURRENT ROOT NODE OCCURRENCE IS ABANDONED AND A NEW       RET00260
   ROOT NODE OCCURRENCE IS OBTAINED AS IN 1.                  RET00270

3. AFTER ONE OCCURRENCE OF THE ENTIRE RETN TREE IS ESTABLISH   RET00280
   ED, A SUBROUTINE BUILD IS CALLED TO GENERATE THE DATA       RET00290
   FROM THE CURSOR IDS; AFTER WHICH THE NEXT ROOT NODE        RET00300
   IS OBTAINED AS IN 1.                                       RET00310

INPUT PARAMETERS:
AS INDICATED IN RETN_ARG;                                          RET00330

OUTPUT PARAMETERS: AS INDICATED IN RETN_RTN;                    RET00360

CALLING ARGUMENTS:                                               RET00380
\* INCLUDE ARETN;                                                  RET00400
/**** CALLS PROCEDURES:                                          RET00410
\* \* INTER-LEVEL T-PROC: NONE                                     RET00420
\* \* INTRA-LEVEL T-PROC: NONE                                    RET00430
\* \* INTRA-LEVEL S-PROC: BUILD, VERIF, RETB, RETP;               RET00440
\* \* CONTROL STRUCTURE, PANEL MANAGER AND DEBUGGING FACILITIES: \* INCLUDE EBUI, EVER1, ERETB, ERETP; \*                   RET00450
\* \* INCLUDE SERVICE, FDEBUG;                                    RET00460
/**** \* INCLUDE EBCT, EPCT, DBE;                                 RET00470
\* DL CURSOR (25) PTR INIT((25) NULL); /* CURRENT PTR TO NODES*/  RET00480
\* DL PSET (25) BIT(32) ALIGNED; /* TEMP FOR PSETID OF RETN TREE NDS*/  RET00490
\* DL (P, MP, RIP, TP, RP, LP, ROOT_PTR) PTR, (NULL, SUBSTR, UNSPEC) BUILTIN;  RET00500
\* DL (PTR1, PTR2) PTR;                                          RET00510
FILE: RIN
PLIOPT A1
VM/SP CONVERSATIONAL MONITOR SYSTEM
PAGE 002

DCL (RNODE,RTN_CODE) FIXED BIN;
DCL BDATA BIT(320) VARYING ALIGNED;
DCL (OP BIT(B), YES BIT(I)) ALIGNED;
DCL (PARENT,BSETID) BIT(32) ALIGNED;
DCL (K,NODE,I) FIXED BIN;
/*DCL OF BCAT TEMPLATE*/
DCL 1 BINFO (25);
  2 PSETID(2) BIT (32),
  2 POS FIXED BIN, /*POS OF POINTER ARRAY OF PSETID! USED*/
  2 FUNC CHAR (1), /*'S' OR 'M'*/
/*DCLS OF PSET CATALOGUE AT THE N-ARY LEVEL*/
DCL 1 PINFO (25);
  3 NUNPTR FIXED BIN,
  3 PLEN /*IN NUMBER OF BYTES*/ FIXED BIN,
  3 PTYPE CHAR (1), /*'N' OR 'X' OR 'B' OR 'C'*/
  3 LTYPE BIT (B), /*'000000' IS LINK LIST*/
  3 L.POS FIXED BIN, /*POS IN ID ARRY 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 TBEQ (PROC ADDR,MP);
ALLOCATE RETN_RTN SET (RTP);
RTP->RETN_RTN.RTN_CODE=0; /*INIT RTN CODE TO D.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 RETNOBIT (1) EXT;
IF FFFN THEN DO:
  PUT SKIP EDIT('NOE *PSETID* PAR **GET** N *BSETID* ****OP*** DL DATA')
     RETNO910;
  /*RETNO900:
  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.'/')
     RETN_ARG.OP(1),RETN_ARG.DLEN(1),'')
     (F(3),X(2),A(8),F(3),X(2),B(8),F(2),A);/*PUT SKIP EDIT*/
    IF RETN_ARG.PARENT='O' THEN PUT EDIT(HEX4(ADDR(RETN_ARG.BSETID)->BP))(A(8));
    ELSE PUT EDIT('') (A);
    IF RETN_ARG.N='O'
       THEN DO:
         PUT EDIT(RETN_ARG.OP(1),RETN_ARG.DLEN(1),'')
            (x(1),b(8),f(3),a);/*PUT SKIP EDIT*/
         DO II=1 TO RETN_ARG.DLEN(1)/4;DCL II FIXED BIN;
         PUT EDIT (HEX4 (ADDR (RETN_ARG.DATA (II))->PAAT(II) )) (A (8));
         END;
   END;
   DCL PAAT (10) PTR BASED;
   END;
   DCL PAA (10) PTR BASED;
   END;
   /*INCLUDE HEX; DCL BP PTR BASED;*/
FILE: RETN PLODPT A1 VM/SP CONVERSATIONAL MONITOR SYSTEM

END;
END:

END:

DO P=MP REPEAT P->RETN_ARG_PTR WHILE (P=NULL);
IF RETN_ARG.NODE=25
THEN DO;
  RTP->RETN_RTN.RTN_CODE=1;
  IF FDEBUG.LEV2 THEN PUT SKIP LIST ('(RET) ERR RTN ',
  'PSET DATA STRUCTURE OVER FLOW (LIMIT 25)');
  GO TO RTN;
END;

CALL GPCT (RETN_ARG.PSETID, PINFO(RETN_ARG.NODE), RTN_CODE);
IF RETN_ARG.NODE = NODE
THEN CALL GBCT (RETN_ARG.BSETID, BINFO (RETN_ARG.NODE), RTN_CODE);
PSET(RETN_ARG.NODE)=RETN_ARG.PSETID;
END;

CALL RETP(PINFO(RNODE),NULL,RTN_CODE); /*1ST BEN*/
/*BEGIN TREE PROCESSING*/
OUTER_LOOP:
DO WHILE ("1'8);
/*ROOT NODE PROCESSING*/
P=MP;
IF RTN_PTR = NULL
THEN LEAVE OUTER_LOOP;
ELSE CURSOR(RNODE) = ROOT_PTR;

IF RETN_ARG.N = 0
THEN DO: /*VERIFY NODE DATA AGAINST PREDICATE*/
  IF RETN_ARG.PRED.OP(1); 
  BDATA=SUBSTR(RETN_ARG.PRED.DATA(1),1,
  RETN_ARG.PRED.DLEN(1)+1);
  CALL VERIF (PINFINODE),ROOT_PTR.OP,BDATA,RTN_CODE,YES);
  IF RTN_CODE=0
  THEN DO:
    RTP->RETN_RTN.RTN_CODE=3; /******** RTN CODE 3 ***********/
    IF FDEBUG.LEV2 THEN PUT SKIP LIST ('(RET) 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.TXT REPEAT P->RETN_ARG_PTR WHILE (P=NULL);
NODE=RETN_ARG.NODE;
PARENT =PSET(RETN_ARG.PARENT);
BSETID=RETN_ARG.BSETID;
PTR CURSOR(RETN_ARG.PARENT);
IF PTR1 = NULL
THEN DO;
  IF RETN_ARG.N = 0 THEN GO TO NEXT_ROOT;
  CURSOR(RNODE) = NULL;
  RETO1130
  RETO1140
  RETO1150
  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
END:  
ELSE DO:  
CALL RETB(PARENT.BINFO(NODE),PTR1,PTR2,RTN_CODE);  
IF RTN_CODE=0  
THEN DO:  
RTP->RTN_RTN.RTN_CODE=4;  
/****** RTN CODE 4 ***********/  
IF FDEBUG.LEV2 THEN PUT SKIP LIST ('(RTN) 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->RTN_RTN.RTN_CODE=5;  
/****** RTN CODE 5 *****/  
IF FDEBUG.LEV2 THEN PUT SKIP LIST ('(RTN) 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+1; /*INDEX TO OCCUR OF ROOT NODE*/  
CALL BUILD (CURSOR,PSIT,MP,K,LP,RTN_CODE);  
IF RTN_CODE=0  
THEN DO:  
RTP->RTN_RTN.RTN_CODE=6;  
/****** RTN CODE 6 ***********/  
IF FDEBUG.LEV2 THEN PUT SKIP LIST ('(RTN) 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->RTN_RTN.RTN_CODE=2;  
/******RTN CODE 2 ***********/  
RET01660
RET01670
RET01680
RET01690
RET01700
RET01710
RET01720
RET01730
RET01740
RET01750
RET01760
RET01770
RET01780
RET01790
RET01800
RET01810
RET01820
RET01830
RET01840
RET01850
RET01860
RET01870
RET01880
RET01890
RET01900
RET01910
RET01920
RET01930
RET01940
RET01950
RET01960
RET01970
RET01980
RET01990
RET02000
RET02010
RET02020
RET02030
RET02040
RET02050
RET02060
RET02070
RET02080
RET02090
RET02100
RET02110
RET02120
RET02130
RET02140
RET02150
RET02160
RET02170
RET02180
RET02190
RET02200
IF FDEBUG.LEV2 THEN PUT SKIP LIST ('(RETN) ERR RTN ').
    'AFTER CALLING RETP WHICH GIVES RTN CODE'.RTN_CODE);
    GO TO RTN;
    END;
    IF WP->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*/;
RETP: PROC(PINFO,PTR1,PTR2,RTN_CODE);

/* A S-PROC AT THE M-ARY LEVEL WHICH RETRIEVES THE ID OF THE P ELEM */
/* THATfollows ID1 IN PSETID. IF ID1 IS NULL THEN IT RETRIEVES THE */
/* THE FIRST OCCUR IN PSETID*/

%INCLUDE DEFU,DPCAT;
%INCLUDE 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:
RETI:PROC(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*/
INCLUDE SERVICE,ARET,DEBUG;
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);
*************************************************************************/
/* MEMORY REQUEST TRACE ON */
IF FFF
THEN CALL DUMPIT ('RETRIEVED',ID,NULL);
DCL FFF BIT (1) EXT;
*************************************************************************/
CALL SVC3 (RP);
IF AP->RTN.RTN_CODE=0 THEN DO; /*ID INVALID*/
    CALL SVC3(AP);
    P=NULL;
    RETURN;
END;
P=ADDR(AP->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/*RETI*/;
FILE: SETFH  PLIOPT  A1  VM/SP CONVERSATIONAL MONITOR SYSTEM  PAGE 001

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;  %INCLUDE 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.LEVO) (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;
FILE: SHWE  PLOPT  01  VM/SP CONVERSATIONAL MONITOR SYSTEM  PAGE 001

SHWE: PROC;
/* T-PROC AT THE ENTITY LEVEL, INVOKED IN RESPONSE TO A CALL FROM
MODULE DDO AT THE USER INTERFACE LEVEL TO RETRIEVE DATA DEFINITION
INFO */
SH000010
INCLUDE SERVICE, ASHWE, DECAT, EGACT, EGETC, ARETE, EALLA;
SH000020
DCL (P,LP,TP,PP,RP) PTR, NULL BUILTIN, (RTN_CODE,1,N) FIXED BIN;
SH000030
DCL MOD BUILTIN;
/* BEGIN */
CALL TBEG (PROC_ADDR, SHWE_ARGP);
CALL SHWE_RTN; LP=SHWE_RTN_P;
ALLOCATE SHWE_ARG; RP; LP=P;
RETE_ARG.NAME='E=ESET';
SH000040
RETE_ARG.GET=RETEGET.ALL;
SH000050
ALLOCATE REPE_ARG; LP->RETE_ARG.PTR; LP=P;
RETE_ARG.NODE=2;
RETE_ARG_NAME='E=NAME';
RETE_ARG.GET=RETEGET.ALL;
RETE_ARG.PARENT=1;
CALL TCALL ('RETE',1,RP,TP);
CALL SVC3 (RP);
/* EXAMINE RETURNS */
P=TP;
SH000060
IF RETE_RTN.RTN_CODE=0 THEN DO;
SH000070
SHWE_RTN.RTN_CODE=30+RETE_RTN.RTN_CODE; 30+ ************/;
SH000080
CALL SVC3 (P); GO TO RTN; END;
SH000090
IF RETE_RTN.N=0 THEN DO;
SH000010
SHWE_RTN.RTN_CODE=-2; 30+ -2 IS NO ENTITY SET DEFINED ******;
CALL SVC3 (P); GO TO RTN; END;
SH000030
/* COPY ENAMES RETRIEVED INTO SHWE_RTN2 */
N=RETE_RTN.N;
SH000040
ALLOCATE SHWE_RTN2 SET (PP); SHWE_RTN_PTR=PP;
SH000050
PP->SHWE_RTN2.NUMENAME=N;
SH000060
DO I = 1 TO N:
DO P= RETE_RTN.PTR REPEAT RETE_RTN1.PTR WHILE (P=NL);
SH000070
IF RETE_RTN1.NODE=2 /* ENAME NODE */
THEN DO: PP->SHWE_RTN2.ENAME(I)=RETE_RTN1.CODATA;
GO TO NEXTI;
SH000080
END; 30+GET OUT P LOOP */
END /* P LOOP */;
NEXTI: END /* DO I */;
SH000090
CALL SVC3 (TP); GO TO RTN; /* DONE */
SH000000
FILE: Srwe
PLOPT A1
VM/SP CONVERSATIONAL MONITOR SYSTEM

END /* IF ENAME = ' ' */;

/* CHECK ENTITY SET WHEN ENTITY SET NAME IS GIVEN */
CALL GECT (NAME1,EINFO,RTN_CODE);
IF RTN_CODE = 0 THEN DO;
   SHWE_RTN.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.RTN_CODE=500 + RTN_CODE; /* 500+ ***********/
      GO TO RTN: END;
   IF NUM_NAME2 = 0 THEN DO: /* NO ATTRIBUTES DEFINED */
      SHWE_RTN.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.RTN_CODE=1; /* 1 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 1 */;

RTN: CALL TRTN (PROC_ADDR, SHWE_RTN_P);
RETURN;
END /* SHWE */;
SRCH: PROC(PSETID,DATA,ID,RTN_CODE);
SRC00010
SRC00020
SRC00030
SRC00040
SRC00050
SRC00060
SRC00070
SRC00080
SRC00090
SRC01000
SRC01100
SRC01200
SRC01300
SRC01400
SRC01500
SRC01600
SRC01700
SRC01800
SRC01900
SRC02000
SRC02100
SRC02200
SRC02300
SRC02400
SRC02500
SRC02600
SRC02700

SRC00010
SRC00020
SRC00030
SRC00040
SRC00050
SRC00060
SRC00070
SRC00080
SRC00090
SRC01000
SRC01100
SRC01200
SRC01300
SRC01400
SRC01500
SRC01600
SRC01700
SRC01800
SRC01900
SRC02000
SRC02100
SRC02200
SRC02300
SRC02400
SRC02500
SRC02600
SRC02700
SVCS1: PROC (PROC_NAME, PTR, RTN_CODE);

******************************************************************************
*                         MODULE DESCRIPTION                              *
******************************************************************************
* PURPOSE: IT SERVES AS THE                                               *
* SVC SEND ROUTINE IN THE LOS, TO INSULATE APPLN PROGRAMS                 *
* FROM THE HARDWARE COMM PROTOCOL SEND; THIS ROUTINE                      *
* CREATES A NEW PROCESS AT THE PROPER LEVEL.                             *
******************************************************************************
* METHOD:                                                                *
* 1. DETERMINE THE LEVEL TO WHICH THE PROC BELONGS TO;                    *
* 2. DETERMINE THE TOTAL LENGTH OF MSG BY FOLLOWING CHAIN;               *
* 3. CALL SEND.                                                          *
******************************************************************************
* INPUT PARAMETERS:                                                      *
*  PROC_NAME: CHAR(7), NAME OF PROC TO BE CREATED;                        *
*  PTR: PTR, POINT TO MSG;                                               *
* OUTPUT PARAMETERS:                                                     *
*  RTN_CODE: FIXED BIN, 1=NO SUCH PROC;                                  *
* CALLS PROCEDURES:                                                     *
* SEND.                                                                 *
******************************************************************************
%
#include users, /*DCLS OF SEND PROTOCOLS OF THE CONTROL STRUCTURE*/
#include vp, /*DCLS OF VP DATA STRUCTURES IN THE LOS*/
#include psvc;
#include fdebug;
DCL PROC_NAME CHAR(7), (PTR,P) PTR,
  (RTN_CODE,I,J) FIXED BIN(15);
DCL 1 ARG_LIST BASED (P), /*TEMPLATE OF MSG CHAIN*/
  2 LEN FIXED BIN (15),
  2 CBTP FIXED BIN (15),
  2 PTR PTR;
DCL (LEVEL, VPID, BOXID, LEN) FIXED BIN, TYPE CHAR(1);
#include svctb; /*MACRO DEF OF PROC TABLE*/
#include proccore;
BEGIN PROCESSING+
  *DETERMINE THE LEVEL TO WHICH THE PROCESS TO BE CREATED BELONGS*
  J=0;
DO I=1 TO PROC_TBL_LEN;
  IF PROC_TBL.N(I)=PROC_NAME THEN J=I:
  IF J=0 THEN LEAVE;
END;
IF J=0 THEN DO; /*NO SUCH PROC_NAME*/
  RTN_CODE=1;
  RETURN;
END;
* DETERMINE THE TOTAL LENGTH OF MSG BY FOLLOWING MSG CHAIN*
LEN=0;
DO P=PTR REPEAT P->ARG_LIST.PTR WHILE (P=NULL());
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,TYP Len,PT_SVC);
RTN_CODE=0; /*OK*/
END /*SVCS1*/;
FILE: SVCS2  PLOOPT A1  VM/SP CONVERSATIONAL MONITOR SYSTEM

SVCS2: PROC (PROC_ADDR, PTR, RTN_CODE);

******************************************************************************

* MODULE DESCRIPTION

******************************************************************************

***** PURPOSE: IT SERVES AS THE

***** SVC SEND ROUTINE IN THE LOS, TO INSULATE APPLN PROGRAMS

***** FROM THE HARDWARE COMM PROTOCOL SEND; THIS ROUTINE

***** SENDS A MSG TO AN OLD PROCESS WHOSE ADDR IS PASSED AS

***** TO THIS ROUTINE AS PROC_ADDR.

***** METHOD:

1. DETERMINE THE TOTAL LENGTH OF MSG BY FOLLOWING CHAIN;

2. CALL SEND.

***** INPUT PARAMETERS:

***** PROC_ADDR: (3) FIXED BIN, THE ADDR OF THE OLD PROCESS;

***** PTR: PTR POINT TO MSG;

***** OUTPUT PARAMETERS: RTN_CODE: FIXED BIN, O=O.K.

***** CALLS PROCEDURES: SEND.

******************************************************************************

%INCLUDE FDEBUG;

%INCLUDE USERS: /* DCLS OF SEND PROTOCOLS OF THE CONTROL STRUCTURE */

%INCLUDE VFX: /* DCLS OF VP DATA STRUCTURES IN THE LOS */

DCL 1 PROC_ADDR,

(2 LEVEL,

2 VPID,

2 BOXID) FIXED BIN (15);

DCL (PTR,P) PTR.

RTN_CODE FIXED BIN (15);

DCL 1 ARG_LIST BASED (P). /* TEMPLATE OF MSG CHAIN */

2 LEN FIXED BIN (15),

2 DUM_CBTP FIXED BIN (15),

2 PTR PTR;

DCL (LEVEL,VPID, BOXID, LEN) FIXED BIN, TYPE CHAR(1);

/* DETERMINE THE TOTAL LENGTH OF MSG BY FOLLOWING MSG CHAIN */

LEN=O;

DO P=PTR REPEAT P->ARG_LIST.PTR WHILE (P=NULL());

IF FDEBUG.LEV0 THEN

LEN=LEN + P->ARG_LIST.LEN;

END:

/* SET UP SEND CALL */

LEVEL=PROC_ADDR.LEVEL;

VPID=PROC_ADDR.VPID; BOXID=PROC_ADDR.BOXID; TYPE="S";

CALL SEND (LEVEL, VPID, BOXID, TYPE, LEN, PTR);

RTN_CODE=O; /* O.K. */

END /* SVCS2 */;
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 REPREIVED */
DCL 1 TOKEN BASED (P),
   2 LEN  FIXED BIN(15),
   2 CBTP FIXED BIN(18),
   2 PTR PTR.
   2 DATA CHAR (I REFER (TOKEN,LEN)); /* */
DCL (P,P1) PTR, (1,LEN,CBTP) FIXED BIN(15),
   NULL BUILTIN;
%INCLUDE AREP,ACRT,AEDE,AEDEB,AEDEF,AEDEFP,AEDEL,AMINI,ANINI,ARET;
%INCLUDE ARETE,ARETN,AUDPDE,UDPDE,AVINI,AVNME,AISHNE;
%INCLUDE HEX;
%INCLUDE FODEBUG;
DO WHILE (P=NULL);
P1=TOKEN.PTR; /*SAVE NEXT PTR*/
CBTP=TOKEN.CBTP;
LEN=TOKEN.LEN;
/*
IF FODEBUG.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->DEF_A_ARG;
   WHEN (6) FREE P->DEF_A_RTN;
   WHEN (7) FREE P->DEF_B_ARG;
   WHEN (8) FREE P->DEF_B_RTN;
   WHEN (9) FREE P->DEF_E_ARG;
   WHEN(10) FREE P->DEF_E_RTN;
   WHEN(11) FREE P->DEF_F_ARG;
   WHEN(12) FREE P->DEF_F_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->RET_E_ARG;
   WHEN(22) FREE P->RET_E_RTN;
   WHEN(23) FREE P->RET_E_ARG1;
   WHEN(24) FREE P->RET_RTN1;
   WHEN(25) FREE P->RET_RTN_ARG;
   WHEN(26) FREE P->RET_RTN_RTN;
   WHEN(27) FREE P->RET_RTN_RTN1;
   WHEN(28) FREE P->UPDE_ARG;
   WHEN(29) FREE P->UPDE_RTN;

VC000010
VC000020
VC000030
VC000040
VC000050
VC000060
VC000070
VC000080
VC000090
VC00100
VC00110
VC00120
VC00130
VC00140
VC00150
VC00160
VC00170
VC00180
VC00190
VC00200
VC00210
VC00220
VC00230
VC00240
VC00250
VC00260
VC00270
VC00280
VC00290
VC00300
VC00310
VC00320
VC00330
VC00340
VC00350
VC00360
VC00370
VC00380
VC00390
VC00400
VC00410
VC00420
VC00430
VC00440
VC00450
VC00460
VC00470
VC00480
VC00490
VC00500
VC00510
VC00520
VC00530
VC00540
VC00550
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->SHME_ARG;
WHEN(37) FREE P->SHME_RTN;
WHEN(38) FREE P->SHME_RTN1;
WHEN(39) FREE P->SHME_RTN2;

OTHERWISE DO; PUT SKIP LIST ("****SVC3 WARNING",CBTP);
   SIGNAL ERROR; END;
   END;
   P=P1;
   END;
   */
IF FDEBUG.LEV0 THEN PUT EDIT(' **** SVC3: END') (A);
   */
RETURN;
END /*SVC3*/;

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;
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 FCALL(A) BIT(1) STATIC EXT;  /* FCALL IS DEBUG TRACE BIG */  
DCL FTIME BIT(1) EXT;  /* FTIME IS TIMING REPORT OPTION */  
%INCLUDE FSTRUC;

DCL (RP,P,T) PTR; DCL NULL BUILTN, I FIXED BIN;  
%INCLUDE USERS,VPX,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 FCALL(I)  /* IF TCALL TRACE BIT FOR THAT LEVEL IS ON*/  
THEN PUT SKIP EDIT ('TCALL: ',PROC_NAME)(I,1);
/******************** TTIME REPORT OPTION ***********************/
DCL (BEGTIME,ENUM,LOTIME,CTL) FIXED BIN(31);
DCL RTIMER ENTRY RETURNS(FIXED BIN(31));
DCL (ECOUNT50,EL050,ETIME50) FIXED BIN(31) EXT INIT((50));
IF FTIME THEN DO;
   ALLOCATE LOTIME; LOTIME=0;
   BEGTIME=RTIMER;
   END;
   /******************** TIMING REPORT OPTION ***********************/
   ALLOCATE MSG;
      MSG.PTR=RP; /*CHAIN TO ROOT PTR OF ARG LIST*/  
      MSG.RTN_ADDR.LEVTHSVP->VP.LEVEL;
      MSG.RTN_ADDR.VPID=THISVP->VP.VPID;
      MSG.RTN_ADDR.BOXID=1;
      CALL SVCS (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 */
         EL0D(ENUM)=EL0D(ENUM)+LOTIME; /* ACCUMULATE LOWER LEVEL PROC TIME */
         FREE LOTIME; IF ALLOCATION(LOTIME)=0 THEN LOTIME=LOTIME+BEGTIME;
      END;
       TCA00010  
       TCA00020  
       TCA00030  
       TCA00040  
       TCA00050  
       TCA00060  
       TCA00070  
       TCA00080  
       TCA00090  
       TCA01000  
       TCA01010  
       TCA01020  
       TCA01030  
       TCA01040  
       TCA01050  
       TCA01060  
       TCA01070  
       TCA01080  
       TCA01090  
       TCA01100  
       TCA01110  
       TCA01120  
       TCA01130  
       TCA01140  
       TCA01150  
       TCA01160  
       TCA01170  
       TCA01180  
       TCA01190  
       TCA01200  
       TCA01210  
       TCA01220  
       TCA01230  
       TCA01240  
       TCA01250  
       TCA01260  
       TCA01270  
       TCA01280  
       TCA01290  
       TCA01300  
       TCA01310  
       TCA01320  
       TCA01330  
       TCA01340  
       TCA01350  
       TCA01360  
       TCA01370  
       TCA01380  
       TCA01390  
       TCA01400  
       TCA01410  
       TCA01420  
       TCA01430  
       TCA01440  
       TCA01450  
       TCA01460  
       TCA01470  
       TCA01480  
       TCA01490  
       TCA01500  
       TCA01510  
       TCA01520  
       TCA01530  
       TCA01540  
       TCA01550  
       TCA01560
FILE: TCALL  PLIOPT  A1  VM/SP CONVERSATIONAL MONITOR SYSTEM

/*************************************************************
TCA00560
TCA00570
TCA00580
TCA00590
TCA00600
TCA00610
TCA00620
TCA00630
TCA00640

IF FTCALL (I) THEN PUT EDIT ('*** ',PROC_NAME,'RETURNS') (A,A,A);

FREE THISVP->VP.WAIT.MSG->MSG;
RETURN;
END /*TCALL*/:
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 SVC52
 **** 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 ***/

%INCLUDE USERS,VPX,ESVCS,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 SVC52 (PROC_ADDR,P,RTN_CODE);
CALL FINISH; /*COMMIT SUICIDE*/

END /*TRTN*/;

%INCLUDE HEX;
UPCT: PROC(PSETID,PINFO,RTN_CODE);
/* A SUBROUTINE WHICH ACCEPTS THE PSETID AND A DECODED PINFO 
AND REPLACES THE PCAT ENTRY */

%INCLUDE DPCAT,FODEBUG,OBUE;
%INCLUDE 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;
UPDE:PROC;

/*T-PROC TO PROCESS UPDE **/
/*INCLUDE UPDE,DEBUG,SERVICE,ADUPN,ARETE;*/
/*INCLUDE DECAT;*/
/*INCLUDE EGCT,EGACT,EVTPE;*/
/*INCLUDE 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,RP1,TP2,RP2) PTR, NULL BUILTIN;
DCL (RTN_CODE,1) FIXED BIN, NUM FIXED BIN (31);
DCL BDATA BIT (320), CDATA CHAR (40), DLLEN FIXED BIN;
DCL TO_BIT FIXED BIN INIT (1). TO_CHAR, FIXED BIN INIT (2);
DCL TO_BCK FIXED BIN INIT (3);
DCL (ENAME,ENAME1,ENAME2) 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 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.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.RTN_CODE = 100; /*100 IS NO SUCH ENTITY**/
  CALL TRN (PROC_ADDR,RTP); /*RETRUN**/
  RETURN;
END;
ALLOCATE UPDN_ARG SET (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 UPDN_ARG SET (TP); /*ALLOCATE UPDN TOKENS*/
  TP->UPDN_ARG.PTR=TP; LP=TP;
  TP->UPDN_ARG.UPDN_ARG, BY NAME;
  TP->UPDN_ARG.CBTP=RP->UPDN_ARG.CBTP; /*FIX UP CBTP*/
UPD00010
UPD00020
UPD00030
UPD00040
UPD00050
UPD00060
UPD00070
UPD00080
UPD00090
UPD00100
UPD00110
UPD00120
UPD00130
UPD00140
UPD00150
UPD00160
UPD00170
UPD00180
UPD00190
UPD00200
UPD00210
UPD00220
UPD00230
UPD00240
UPD00250
UPD00260
UPD00270
UPD00280
UPD00290
UPD00300
UPD00310
UPD00320
UPD00330
UPD00340
UPD00350
UPD00360
UPD00370
UPD00380
UPD00390
UPD00400
UPD00410
UPD00420
UPD00430
UPD00440
UPD00450
UPD00460
UPD00470
UPD00480
UPD00490
UPD00500
UPD00510
UPD00520
UPD00530
UPD00540
UPD00550
TP->UPDN_ARG_PTR = NULL;
NODE=UPDE_ARG.NODE;
ENAME=MAININFO.ENAME(UPDE_ARG.PARENT);
ANAME=UPDE_ARG.NAME;
CALL GACT (ANAME, ENAME1,AINFO,RTN_CODE); /*GACT GETS ATTR CAT E*/
IF RTN_CODE = "O"
THEN DO;
RTP=UPDE_RTN.RTN_CODE=NODE+100; /*IS NO SUCH ATTRIBUTE*/
CALC SVC3 (RP); /*FREE UP UPDN ALLOCATION UP TO NOW*/
CALL TRTN (PROC_ADDR, RTP); /*RETURNS*/
RETURN;
END; /*SELECT*/
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 (UPDE_ORD_ORD) TP->UPDN_ARG.OP=UPDN_OP.CI;
WHEN (UPDE_ORD_DEL) TP->UPDN_ARG.OP=UPDN_OP.DO;
WHEN (UPDE_ORD_REP) TP->UPDN_ARG.OP=UPDN_OP.REP;
OTHERWISE;
END; /*SELECT*/;
END; /*IF V*/
IF UPDE_ARG.PARENT = 1 THEN /*FOR ALL ATTR'S NOT DIRECTLY RELATED TO ROOT NODE*/
TP->UPDN_ARG.OP=UPDN_OP.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(UPDN_OP.CI,UPDN_OP.REP) DO;
ALLOCATE RETE_ARG SET (EP);
EP->RETE_ARG.NAME=ENAME; /*IN CURRENT VERSION*/
   PARENT MUST BE ROOT NODE*/
EP->RETE_ARG.GET = RETGET.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.NODE.DLEN=UPDE_ARG.DLEN;
EP1->RETE_ARG.PRED.OP=RETE.OP.EQ;
CALL TCALL ('RETEN',EP,EP1);
IF EP1->RETE_RTN.RTN_CODE = "O" | EP1->RETE_RTN.N = 0
THEN DO; /*PROBLEM DATA, KEY VIOLATION*/
IF FDEBUG.LEV1
UPD00900
UPD00910
UPD00920
UPD00930
UPD00940
UPD00950
UPD00960
UPD00970
UPD00980
UPD00990
UPD1000
UPD1001
UPD1002
UPD1003
UPD1004
UPD1005
UPD1006
UPD1007
UPD1008
UPD1009
UPD1010
UPD1011
THEN PUT SKIP LIST ('(UPDE) KEY VIOLATION');
RTP->UPDE_RTN.RTN_CODE=300+UPDE_ARG.NODE;
/**** RTN_CODE 300+ IS VIOLATION OF KEY CONSTRAINT*/
CALL SVC3 (EP); CALL SVC3 (EP1); CALL SVC3 (RP);
CALL TRN (PROC_ADDR, RTP);
RETURN;
END /*RTN_CODE CHK*/;
END /*IST OR REP*/;
OTHERWISE;
END /*SELECT*/;
END /*FUNC="K"*/;

/*CHECK DATA TYPE AND NUMERIC LIMITS*/
CDATA=UPDE_ARG.CDATA;
DLEN=UPDE_ARG.DLEN;
CALL VPTE (AINFO.TO_BIT_CHK, BDATA, CDATA, DLEN, RTN_CODE);
IF RTN_CODE<>0
THEN DO: /*PROBLEMS*/
RTP->UPDE_RTN.RTN_CODE=NODE; /*T IS ILLEGAL NODE DATA*/
CALL SVC3(RP); /*FREE UP UPDN ALLOCATION UP TO NOW*/
CALL TRN (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.BDATA=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*/
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 = O;
UPDN_ARG.PARENT = 1; /* ALWAYS ROOT NODE */
UPDN_ARG.NODE = NODE + I;
IF AINFO_TYPE = 'V'
THEN UPDN_ARG.OP = UPDNOP_DD;
ELSE UPDN_ARG.OP = UPDNOP_DEL;
END /* DO I */;
END /* IF DEE */;

/*CALL UP UPDN **/
CALL TCALL ('UPDN', I, RP, TP);

/*CHECK RETURNS*/
I = TP->UPDN_RTN.RTN_CODE;
IF I = 0 THEN
RTP->UPDN_RTN_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*/;
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 occurrences 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 occurrences
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:

INCLUDE AUPTDN;
INCLUDE AUPTDN PROCEDURES:
INTER-LEVEL T-PROC: NONE;
INTRA-LEVEL T-PROC: RETN;
INCLUDE ARETN;
INCLUDE S-PROC: MODN, CRTN, DELN;
INCLUDE EMODN, ECRTN, EDLN;
CONTROL STRUCTURE, PANEL MANAGER AND DEBUGGING FACILITIES:
INCLUDE SERVICE, FDEBUG;

DCL (P,TP,RP,RTP,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);
/*
FILE: UPDN   PLIOPT   A1   VM/SP CONVERSATIONAL MONITOR SYSTEM   PAGE 002

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 N;
    TEMP.IDN(N,J) = O;
    TEMP.IDP(N,J) = NULL;
  END;
END;

/*******************************************************************************/
DCL FFN BIT (1) EXT;
IF FFN 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(4),F(4),F(4),A);
  IF UPDN_ARG.PARENT=O THEN PUT EDIT(HEX4(ADDR(UPDN_ARG.BSETID)->BP),`')(A(8),A);
ELSE PUT EDIT(`')(A);
  DO II = 1 TO UPDN_ARG.DLEN/4; DCL II FIXED BIN;
    PUT EDIT(HEX4(ADDR(UPDN_ARG.DATA)->PA(II)))(A(8));
  END;
END;
DCL PA(A(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);
  NTRP(UUPDN_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 NODE 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;
/*******************************************************************************/
FILE: U-0N     PLOOPT A1     VM/SP CONVERSATIONAL MONITOR SYSTEM

END;
ELSE;
    IF LP=NUILL
        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;
    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 ORRURRENCE 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 = RETGET.ALL;
            RETN_ARG.PSETID = NTR(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(N) = IDP(N,J)->UPDN_ARG.DLEN;
                RETN_ARG.PRED.DATA(N) = IDP(N,J)->UPDN_ARG.DATA;
                RETN_ARG.PARENT = 1;
                RETN_ARG.N = 1;
                RETN_ARG.GET = RETGET.NO;
                RETN_ARG.PRED.CN(1) = 1;
                RETN_ARG.PRED.OP(1) = RETNOP.EQ;
                END /* J */;
            END /* CALL UP RETN */;
        CALL TCALL ('RETN',I,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 ***********
            UPD01150
            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*/
            UPD01640
            UPD01650
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*/
NPT(T)=UPDN_ARG.DLEN=P->RETN_RTN1.DLEN;
NPT(T)=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*****/
/*MP*/;
SELECT (UPDN_ARG.OP);
WHEN (UPDNOP.OFF) CALL CRTN (P.OFF_NODE,RTN_CODE);
WHEN (UPDNOP.DEL) CALL DELN (P.OFF_NODE,RTN_CODE);
WHEN (UPDNOP.MOD) CALL MOON (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;

/*SELECT****/;

/*CHECK SUBROUTINE RTN CODE ******/
IF RTN_CODE=0 THEN DO ; /*RTN CODE 0 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*/;
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 **/

%INCLUDE AUPE, 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 (OPE(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*/
DCL N FIXED BIN, /*SERVE AS TOKEN COUNT*/ I FIXED BIN;

/INITIALIZE NA STRUCTURE*/
DO I=1 TO 25;
   NA(I)='O'8;
   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;  /* LEGAL 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='O'
       THEN DO;

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
DUM00270
DUM00280
DUM00290
DUM00300
DUM00310
DUM00320
DUM00330
DUM00340
DUM00350
DUM00360
DUM00370
DUM00380
DUM00390
DUM00400
DUM00410
DUM00420
DUM00430
DUM00440
DUM00450
DUM00460
DUM00470
DUM00480
DUM00490
DUM00500
DUM00510
DUM00520
DUM00530
DUM00540
DUM00550
FILE: UPE1  PLOPT A1  VM/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=TI.PARENT(I)+1;
UPDE_ARG.NAME=TI.NAME(I);
SELECT(OP(I));
WHEN ('T') UPDE_ARG.OP=UPDEOP.ID;
WHEN ('N') UPDE_ARG.OP=UPDEOP.INS;
WHEN ('D') UPDE_ARG.OP=UPDEOP.DEL;
WHEN ('R') UPDE_ARG.OP=UPDEOP.REP;
OTHERWISE DO;
RTN_CODE=2;
PUT SKIP LIST ('ILLEGAL SEMANTICS OF MODIFICATION STMT.');
CALL SVC3(RP);
RETURN;
END; /*SELECT*/;

IF NA(I)=0 AND 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.DAT=UPDEOP.D(K);
UPDE_ARG.DLEN=UPDEOP.L(K);
END;
END /*DO1*/;

/*ALL NODES ARE ALLOCATED*/
CALL TCALL ('UPDE',1,RP,TP);
CALL SVC3 (RP); /*FREE ARG LIST*/
CODE=TP->UPDE_RTN.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 ('RECN RTN CODE ,CODE-200)(A,F(3));
ELSE PUT SKIP EDIT ('ILLEGAL DATA FOR ATTRIBUTE', 
T.TI.NAME(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: USER PLI0PT A1
VM/SP CONVERSATIONAL MONITOR SYSTEM

USER:PROC;
MISSIONARY DESCRIPTION

I PURPOSE:
***** A T-PROC FOR THE USER TO SELECT SUBSYSTEMS AFTER LOGGIN 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.

%INCLUDE EDBA, SERVICE; /* ENTRY DCLS OF SUBSYSTEM*/
DCL NAME CHAR(7), P PTR;
/*BEGIN PROCESSING*/
CALL TBEQ (PROC_ADDR, P); /* JUST FOR CONVENTION */

DO WHILE ('1B');
    PUT SKIP LIST ("SUBSYSTEMS: DBA, B(item2BASE VIEW) OR R/item2(RELATIONAL VIEW)" USER00230 ?");
    GET EDIT (NAME) (A(7));
    IF NAME(7) ' THEN LEAVE;
    SELECT (NAME);
    WHEN ("DBA",'D') CALL DBA;
    WHEN ("BNV", 'B') CALL BNV;
    WHEN ("RELV", '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:
/*USER*/;

CALL TRIN (PROC_ADDR, P);
END */USER*/;
FILE: VERIF  PLIOPT A1  VM/SP CONVERSATIONAL MONITOR SYSTEM  PAGE 001

VERIF: PROC(PINFO,P,VOP,BDATA,RTN_CODE,YES);

/* A S-PROC IN THE N-ARY LEVEL TO VERIFY AND CONVERT DATA */

%INCLUDE DPCAT,DEBU;
%INCLUDE EPCT,ESUB;
DCL BDATA BIT (*) VAR ALIGNED;
VOP BIT(0) ALIGNED, RTN_CODE FIXED BIN, YES BIT(1) ALIGNED;
DCL PTYP CHAR (1), BDATA1 BIT (32) VARYING;
DCL (NUM,NUM1) FIXED BIN (31);
DCL 1 OP ALIGNED, /* SAME AS RETNOP */
  2 EQ BIT (B) INIT ('00000000'B),
  2 GT BIT (B) INIT ('00000001'B),
  2 LT BIT (B) INIT ('00000000'B);
DCL P PTR, NULL BUILTIN;
DCL UNSPEC BUILTIN;

/*BEGIN PROCESSING*/
YES='O'; /*INIT TO NO*/

PTYPE=PINFO.PTYPE;

SELECT (PTYPE);
  WHEN('N') DO: /*CONVERSION*/
    /* PUT SKIP LIST ('(VERIF), WHEN N'); */
    UNSPEC(NUM)=BDATA;
    UNSPEC (NUM1)=BEU.DA;
    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 /*SELECT OP*/;
    END /*WHEN N*/;

  OTHERWISE DO: /*END CONVERSION*/
    BDATA1=BEU.DA;
    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*/;

VER000010
VER000020
VER000030
VER000040
VER000050
VER000060
VER000070
VER000080
VER000090
VER001000
VER001100
VER001200
VER001300
VER001400
VER001500
VER001600
VER001700
VER001800
VER001900
VER002000
VER002100
VER002200
VER002300
VER002400
VER002500
VER002600
VER002700
VER002800
VER002900
VER003000
VER003100
VER003200
VER003300
VER003400
VER003500
VER003600
VER003700
VER003800
VER003900
VER004000
VER004100
VER004200
VER004300
VER004400
VER004500
VER004600
VER004700
VER004800
VER004900
VER005000
VER005100
VER005200
VER005300
VER005400
VER005500
**FILE: VINIT  PLIOPT  A1  VM/SP CONVERSATIONAL MONITOR SYSTEM**

**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 DEFB (through DFB1) 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 upon; */
/****** 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 FROM_RTN and in turn retrieve all keys. */
/****** INPUT PARAMETERS: */
/****** AS INDICATED IN VINIT_ARG; */
/****** OUTPUT PARAMETERS: */
/****** AS INDICATED IN VINIT_RTN;  */
/****** CALLING ARGUMENTS:  */
/****** CALLS PROCEDURES: */
/****** inter-level t-proc: NINIT,RETN,UPDN;  */
/****** include ANNI,ARETN,UPDPN; */
/****** intra-level t-proc: none; */
/****** intra-level s-proc: DFP1,DFB1; */
/****** include EDP1, EDFB1; */
/****** control structure, panel manager and debugging facilities: */
/****** include SERVICE,FDDEBUG; */
/****** external static variables to hold static data: KEY */
/****** include DKEY;  */
/*************************************************************************************/

**DCL (P,TP,RP,MP,RT,P) PTR;**
VINO0430
**DCL NAME CHAR(8), 1 FIXED BIN;**
VINO0440
**DCL NULL BUILTIN;**
VINO0450
**DCL RTH_CODE FIXED BIN (15);**
VINO0460
**DCL PC(7), */DATA* FOR DEF OF 7 PSETS IN CATALOGUE; SEQUENCE STARTS WITH DATA FOR ESET, ENM,INFO,ASET,ANM,AINF,KEY:/**
VINO0470
**2 PTYPE CHAR (1) INIT ('X','C','B','X','C','B','B').**
VINO0480
**2 PLEN FIXED BIN INIT (0,8,4,0,8,29,4).**
VINO0500
**2 TYPE CHAR(1) INIT ('E','V','V','E','V','V','V').**
VINO0510
**2 PSETID BIT (32);**
VINO0520
**DCL BC(5), */DATA* FOR DEF OF 5 BSETS IN CATALOGUE; SEQUENCE IS EXPAND DựESET,ANAME,AINF.**
VINO0530
**2 FUNC CHAR(1) INIT ('S', 'S', 'W', 'S', 'S').**
VINO0550
FILE: VINIT PLIOPT A1 VM/SP CONVERSATIONAL MONITOR SYSTEM PAGE 002

2 PSETID (2) BIT(32),
2 RTN_CODE FIXED BIN,
2 BSETID BIT (32);
DCL AKY BIT(32);

/*BEGIN PROCESSING*/
CALL TBEG (PROC_ADDR,MP);
ALLOCATE VINIT_RTN_SET (RTP);
RTP->VINIT_RTN_RTN_CODE = 0; /*INITIALIZE RTN_CODE TO O.K.*/
/*SET UP MSG TO CALL NINIT*/
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.RTN_CODE ^=O THEN DO;
RTP->VINIT_RTN.RTN_CODE = 1; /*SET RTN CODE INVALID*/
CALL SVC2(TP); /*FREE RTN MSG*/
GO TO RTP; /*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 DFB1(PC.PTYPE(I), PC.PLEN(I), PC.TYPE(I), RTN_CODE,
PC.PSETID(I));
IF RTN_CODE ^=O THEN DO;
RTP->VINIT_RTN.RTN_CODE = 1;
IF FDEBUG.LEVEL THEN PUT SKIP LIST (*VINIT: DFB1 ABEND');
GO TO RTP;
END;

IF I=7 THEN AKY=PC.PSETID(I);
ELSE KEY(I) = PC.PSETID(I);
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(3);
BC.PSETID(2,2)=KEY(4);
BC.PSETID(3,1)=KEY(5);
BC.PSETID(3,2)=KEY(6);
BC.PSETID(4,1)=KEY(7);
BC.PSETID(4,2)=KEY(8);
BC.PSETID(5,1)=KEY(9);
BC.PSETID(5,2)=KEY(10);
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(I));

204
FILE: VINIT PLOPT A1

IF BC.RTN_CODE (1) ^=O THEN DO;
  RTP->VINIT_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=UPDN.OP.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 ^=O THEN DO;
  RTP->VINIT_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 ^=O THEN DO;
  RTP->VINIT_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.SET=RETN_ARG.ALL;

CALL TCALL ('RETN',1,P,TP);
CALL SVC3(P); /*FREE CALLING ARG*/

PAGE 003
FILE: V...IT  PLIOPT  A1  VM/SP CONVERSATIONAL MONITOR SYSTEM  PAGE 004

IF TP->RETN.RTN.RTN_CODE ^=0 THEN DO;
  IF FDEBUG.LEV1 THEN PUT SKIP LIST ("INSIDE FILE INIT, JUST
turned from RETN, and RTN_CODE is not O.K.");
  RTP->VINIT.RTN.RTN_CODE =1; CALL SVC3(TP);
  GO TO RTN;
  END;
  END;
  END;
  END;

/*GET KEYS*/
P=TP->RETN.RTN.PTR;
DO I=1 TO N:
  KEY(I-1)=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*/;
VM: PROC;
/* T-PROC AT THE ENTITY LEVEL. INVOKED BY DMB (BASE DATA MANIPULATION) VMM0030
SUBSYSTEM AT THE USER INTERFACE LEVEL, TO CHECK FOR LEGALITY OF NAMES VMM0040
STATE BY USER BEFORE ACCEPTING DATA */
VMM0050
VMM0060
VMM0060
VMM0070
VMM0080
VMM0090
DCL PARENT_NAME(25) CHAR (8); /* TEMP STRUCTURE */
DCL I FIXED'BIN (31);
/
BEGIN /*
CALL TBEG (PROC_ADDR,P);
ALLOCATE VMNE_RTN SET (RTN);
CALL GECT (T.ENAME,EINFO, RTN_CODE);
IF RTN_CODE ^= 0 THEN DO;
  RTN->VMNE_RTN.N=1; /* PROBLEM WITH ENTITY SET NAME */
  GO TO RTN;
END;
/
VALIDATE ATTRIBUTES /*
DO I = 1 TO T.N;
  IF T.PARENT(I)=0 THEN ENAME=T.ENAME;
  ELSE ENAME = PARENT.ENAME (T.PARENT (I));
  CALL GACT (T.NAME (I), ENAME, AINFO, RTN_CODE);
  IF RTN_CODE ^= 0 THEN DO;
    RTN->VMNE_RTN.N=1; /* 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 */
RTN->VMNE_RTN.N=0; /* NO PROBLEM */
RTN: CALL TRTN (PROC_ADDR, RTN);
RETURN;
END /* VMNE */;
FILE: WNM1   PL/1OPT A1   VM/SP CONVERSATIONAL MONITOR SYSTEM

WNMI: PROC (T, RTN_CODE, N);
/**SUBROUTINE TO SET UP CALL TO WNMME**/
%INCLUDE AVNME, SERVICE, FDEBUG;
DCL (RTN_CODE, N) FIXED BIN;
DCL T
  /*DATA STRUCTURE OF NAMES TO BE VERIFIED*/
  2 ENAME CHAR(8),
  2 TN FIXED BIN, /*NUMBER OF ATTR*/
  2 T1 (25),
  3 PARENT FIXED BIN,
  3 ANAME CHAR (8);
DCL (TP, P) PTR, NULL BUILTIN;
ALLOCATE VNAME_ARG;
VNAME_ARG.T=T;
CALL TCALL('WNME',1,P,TP);
CALL SVC3(P);
RTN_CODE= TP->VNAME_RTN.RTN_CODE; /* RTN_CODE NOT SIGNIFICANT*/
N=TP->VNAME_RTN.N;
IF N = 0 /* NAME ERROR*/
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
DUM0100
DUM0110
DUM0120
DUM0130
DUM0140
DUM0150
DUM0160
DUM0170
DUM0180
DUM0190
DUM0200
DUM0210
DUM0220
DUM0230
DUM0240
DUM0250
DUM0260
FILE: VTPE  PLOPT  A1  VM/SP CONVERSATIONAL MONITOR SYSTEM  PAGE 001

VTPE: PROC (AINFO, MODE, BDATA, CDATA, DLEN, RTN_CODE);
VTPO0010
VTPO0020
*/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*/
VTPO0030
VTPO0040
VTPO0050
VTPO0060
*/INCLUDE DECAT; /*FOR AINFO TEMPLATE*/
VTPO0070
*/INCLUDE FDEBUG;
VTPO0080
DCL MODE FIXED BIN;
VTPO0090
DCL TO_CHK FIXED BIN INIT (2); TO_BIT FIXED BIN INIT (1);
VTPO1000
VTPO1010
DCL TO.Bit_CHK FIXED BIN INIT (3);
VTPO1020
DCL CDATA CHAR (40), BDATA BIT (320), (DLEN, RTN_CODE) FIXED BIN;
VTPO1030
DCL NUM FIXED BIN (31); /*TO HOLD CONVERTED FIXED BIN NUMERIC DATA*/
VTPO1040
DCL VERIFY BUILTIN; DCL BUILTIN;
VTPO1050
DCL DATA CHAR (40) VARYING;
VTPO1060
DCL (UNSPEC, SUBSTR) BUILTIN, I FIXED BIN;
VTPO1070
/*BEGIN PROCESSING*/
VTPO1080
RTN_CODE=0; /*INIT TO O.K.*/
VTPO1090
SELECT (MODE);               
VTPO1100
WHEN (TO_BIT, TO_BIT_CHK) DO;
VTPO1110
/*FROM CHAR STRING TO BIT STRING*/
VTPO1120
IF AINFO.TYPE='V' THEN DO; /*FOR V TYPE NODE ONLY*/
VTPO1130
SELECT (AINFO.VTYPE);
VTPO1140
WHEN ('N') DO; /*NUMERIC DATA*/
VTPO1150
   DATA=SUBSTR(CDATA,1,DLEN);
VTPO1160
   I=VERIFY(DATA, '0123456789 ');      
VTPO1170
   IF I='0' THEN DO;
VTPO1180
      RTN_CODE=1; /*ILLEGAL NUM DATA*/
VTPO1190
      RETURN;
VTPO1200
   END;
VTPO1210
   IF DATA=(40) ' THEN NUM=0;
VTPO1220
   ELSE NUM=DATA; /* CONVERSION */
VTPO1230
   /*CHECK FOR LIMITS*/
VTPO1240
   IF MODE=TO_BIT_CHK THEN DO;
VTPO1250
      IF NUM<AINFO.MIN THEN LEAVE;
VTPO1260
      ELSE DO; /*VOLATIL LIMITS*/
VTPO1270
         RTN_CODE=5; /******5 IS LIMIT VIOLATIONS*********/
VTPO1280
         RETURN;
VTPO1290
      END;
VTPO1300
   END;
VTPO1310
   SUBSTR(BDATA,1,32)=UNSPEC (NUM); /*TO BIT STRING*/
VTPO1320
   DLEN=4; /*4 BYTES LONG*/
VTPO1330
   END /*WHEN 'N'*/;
VTPO1340
OTHERWISE DO;
VTPO1350
   BDATA=UNSPEC(SUBSTR(CDATA,1,AINFO.MLEN));
VTPO1360
   DLEN = AINFO.MLEN; END;
VTPO1370
VTPO1380
VTPO1390
VTPO1400
VTPO1410
VTPO1420
VTPO1430
VTPO1440
VTPO1450
VTPO1460
VTPO1470
VTPO1480
VTPO1490
VTPO1500
VTPO1510
VTPO1520
VTPO1530
VTPO1540
VTPO1550
FILE: VTPE  PLOOPT A1

VM/SP CONVERSATIONAL MONITOR SYSTEM

PAGE 002

END /*SELECT VTYPE*/;
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 VTYPE*/;
END /*TYPE*/;
END /*MODE TO_CHAR*/;
END /*SELECT MODE*/;
END /*VTPE*/;