N-ARY LEVEL
IN THE SOFTWARE TEST VEHICLE
FOR THE INFOPLEX DATABASE COMPUTER

by

DAVID LUI

SUBMITTED TO THE DEPARTMENT OF
ELECTRICAL ENGINEERING AND
COMPUTER SCIENCE IN PARTIAL
FULFILLMENT OF THE REQUIREMENT
FOR THE DEGREE OF

BACHELOR OF SCIENCE

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

MAY, 1982

© MASSACHUSETTS INSTITUTE OF TECHNOLOGY, 1982

Signature of Author

Department of Electrical Engineering and Computer Science
May, 1982

Certified by Professor Stuart Madnick
Thesis Supervisor

Accepted by Professor David Adler
Chairman, Department Committee
CONTENTS

Chapter 1 General Overview..........................Page 1
Chapter 2 Data Organization in the Entity and
    the U-nary Levels..........................Page 6
Chapter 3 Functional Modules, Data Structures
    and Strategy...............................Page 12
Chapter 4 Description of Individual Functional
    Modules...................................Page 24
Chapter 5 Simulation and Simulated
    Procedures...............................Page 30
Bibliography....................................Page 33
Appendices.....................................Page 34
Appendix 1....................................Program Listings
Appendix 2....................................Data Structures
Appendix 3.................................Sample Terminal Session
Chapter One

General Overview

1.1 Introduction

Infoplex is a database computer architecture having as its objective the support of large-scale information management with high reliability. (Ref. 1) It aims to provide a solution to the problem of increasing loads, in terms of both throughput and volume of stored data, faced by today's and tomorrow's information processing needs.

Infoplex consists of a storage hierarchy, which supports a very large data storage system, and a functional hierarchy, which is responsible for providing all database management functions other than device management.

The Infoplex functional hierarchy is designed around a concept of hierarchical decomposition. (Ref. 2) It is a discipline that helps in identifying the key functional modules that have minimal interdependencies and can be combined hierarchically to form a software system, such as an operation system or a database management system.
FUNCTIONAL HIERARCHY

VIEW AUTHORIZATION LEVEL

VIEW TRANSLATION

VIEW ENFORCEMENT VALIDITY/INTEGRITY

VIRTUAL INFORMATION

N-ARY/BINARY

UNARY

DATA ENCODING/ MEMORY MANAGEMENT

↓ TO STORAGE HIERARCHY  Fig. 1
The idea of data abstraction is widely used in the functional hierarchy. The functional hierarchy takes care of accepting user's commands and doing the proper updating. According to the preliminary design, it is divided into ten levels each having its own data structures and its own functions. Communications between different levels take place in the form of control blocks. These blocks queue up and wait to be executed. The ten levels are i) View Authority Level ii) View Translation Level iii) View Enforcement Level iv) Validity/Integrity Level v) Virtual Information Level vi) N-ary Level vii) Binary Level viii) Unary Set Level ix) Data Encoding Level x) Memory Management Level. Refer to Fig.1.

Before a hardware prototype is commissioned, a software simulation is to be implemented as a vehicle for validating the communication and functional algorithms in the preliminary design. Previous efforts in this software test vehicle (STV) project include an early version (Ref.3) and a later version (Ref.4) of the functional hierarchy, and the implementation of a control structure which emulates the multi-level multi-processing environment. (Ref.5) The work described in this proposal is part of an integrated effort to improve on the more recent version of the functional hierarchy. In particular my work involves implementation of the N-ary level.

1.2 Proposed Work
All the data in the N-ary level are grouped into Primitive sets (Psets) and are made to relate to each other through Binary Association sets (Bsets). The level above the N-ary level will map logical constructs such as entity sets and attributes onto Psets and Bsets. The N-ary level will, in turn, map Psets and Bsets onto record sets, or U-nary sets. It will, according to some algorithm, group related Psets into record sets, and implement additional associations through linkage field(s) in the records. The U-nary sets are implemented at the Unary Level, which directly supports the N-ary level.

The primary tasks of the N-ary Level include maintaining a catalogue of all the Psets and Bsets defined, updating of the data, and retrieval of data. Updating data expands into creating new records, deleting existing records and modifying the contents of specified records. Fig.2 shows the outline of the three subsystems of the N-ary Level and the interactions among the three.

My work is to implement the creation of new records, deletion of existing records and modifying the contents of specified records.
ENTITY
N-ARY

RETRIEVE PROPER RECORD GIVEN ATTRIBUTES

CREATE NEW RECORDS
DELETE OLD RECORDS
MODIFY CONTENTS OF EXISTING RECORD

DEFINE NEW CATALOGUE ENTRY

N-ARY
U-NARY

CALL PROPER MODULES IN THE U-NARY LEVEL
2.1 Entity Level Concepts

At the Entity Level, data are organized by the binary network model. A visual presentation of our binary network model is shown in the following figure. There are four basic constructs. Primitive Elements represent some objects or facts in the real world. (Fig. 2.1a) A primitive Set is a group of primitive elements that have similar generic properties and therefore are given a common group name, called a Primitive Set Name, or Pset_name. (Fig. 2.1b) Binary associations are representations of some real world relationship among primitive elements from different primitive sets. (Fig. 2.1c) A binary set is a group of binary associations that have similar generic properties (i.e., the incident primitive elements belong to the same primitive sets, and the associations have the same meaning). It is designated a pair of primitive set names and a pair of association names. (Fig. 2.1d)
In Fig.2.1d, the upper portion (primitive elements and binary associations) represents the instance of the database, while the lower portion represents the schema of the database. Therefore, the schema of our Binary Network model is composed of primitive sets (also known as the 'nodes') and binary association sets (also known as the 'arcs').

Further classification of nodes and arcs: In a binary network schema graph, a node can be either an entity node or a value node. An entity node serves to tie all equally related value nodes or entity nodes together. By 'equally related' we mean that those nodes tied to this entity node are all direct attributes of the entity node, instead of 'derived' attributes. An entity node corresponds to any set of real world objects (tangible or intangible) that have some common set of attributes which are revealed by the node's binary connections to other value nodes or entity nodes. It's own identity is reflected by these associations; i.e., an instance of an entity node does not have any value or identity, and its designation is made through instances of its associated nodes. In a sense, the purpose of an entity node is to collect equally related binary associations to form a semantically clean N-ary association. Therefore an entity node is also referred to as an N-ary entity node. Those nodes that are not entity nodes are value nodes. Value nodes, in contrast to entity nodes, have values assigned to their instances.
Arches can be specified by several parameters. It can be many-to-1 (mtol), one-to-one (ltol), or one-to-many (ltom). A second parameter is 'E' or 'N' which means whether or not the child's recordid is embedded in the parent's record respectively. Fig. 2.2 shows a couple of examples on how information is moulded into a data tree.

2.2 U-nary Concepts

At the U-nary Level, the data are organized in files, records and fields. In principal, a Pset would be mapped to a file. A file is divided into records and fields. A record is
contain the actual data. Each file has a unique numerical ID. Within each file, each record has a unique recordid. Within each record, each field has a unique fieldid. Each field is designated to contain a pre-determined kind of data.

There are two kinds of data—actual numerical or character data, and recordid. In order to implement relation or Bset in the U-nary Level, recordids are used. If the relation between two Psets is 'E', then the recordid of the child Pset is included in a designated data field of the record of the parent to mimic the relationship.

To continue with the two examples presented earlier in the discussion of data organization in the Entity Level, here are the records that correspond to the information contained in the entity level:

<table>
<thead>
<tr>
<th>RECORDID</th>
<th>FIELDID</th>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>DAVID</td>
<td>21</td>
<td>9</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>MEI</td>
<td>22</td>
<td>1</td>
<td></td>
</tr>
</tbody>
</table>

FILEID = 1
The crux of the problem lies in how to map the data in the Entity Level to the data in the Unary Level. The work is done by the N-ary level and part of the work is accomplished by my thesis.
Chapter Three

Functional Modules, Data Structures
and Strategy

3.1 Examplary Update trees

Although all the three operations use the same data structures, they differ subtly in their contents. A discussion of an example for each is obviously warranted. As a preview, a discussion of the general structure of an update tree would precede that of any specific tree.

All the information that is required for a certain kind of update, whether Create, Delete or Modify are formulated by the level above in an update tree. The update tree contains five kinds of nodes: one rootnode, value nodes, many-to-l(mtol) entity nodes, one-to-one(ltol) entity nodes and one-to-many(ltom) entity nodes. The rootnode is the file which contains the record to be updated. All the attached nodes are either of two kinds—value or entity. A value node may contain only one data value, e.g., age. An entity node is another record. For example, a tree that models Mei's personal record to be updated is as follows:
Mei is of age 22, home state being Ma, whose job is student, office address being Bldg 53 and of salary $2000/mthly. The tree that contains all this information is as follows:

![Diagram](image)

Basically, there are three kinds of operations involved: create a new record in a certain file and, if necessary, insert this new recordid in existing record; delete an existent record; modify the value contents or recordid contents of an existing record.

For creation, three kinds of nodes are attached to the rootnode. All the values or recordid in these nodes are to be
filled in the new record to be created or if the new record is supposed to be contained in one of these nodes, the recordid of this newly created node is inserted into the appropriate record. The value nodes have their values contained in the UPDN_ARG which is the control block between the N-ary level and the previous level. The entity nodes have sufficient nodes attached to them so as to enable this level to retrieve the recordid.

For deletion, all the nodes that are attached to rootnode, whether it is a value or entity node, are contained within the rootnode to identify the record. The recordid retrieved is passed down to the next level for deletion.
For modification, there are four kinds of nodes as in creation: the value nodes, the mtol entity nodes, the ltol entity nodes, and the ltom entity nodes. The three kinds of operations are ID, REPLACE, ADD, REMOVE. The ID is to identify the record to be modified. The REPLACE changes the value contents of the value nodes. ADD and REMOVE simply add or remove a Bset relation.

3.2 Modules of N-ary Planned Implementation

As a preliminary consideration, the implementation consists of a main program which takes care of the control and calling of subroutines. The main program would be separated into two parts. The first part deals with mtol and embedded...
ltol relationship only while the second part deals with non-embedded ltol relationship and ltom relationship. It is useful to separate one from the other because in solving some non-embedded relationship issues it is necessary to obtain information from the embedded relationship. Also it is a good idea to maintain a clear-cut distinction between the two because the coding would have very little similarity and hence, it is not meaningful to merge them together.

Within the upper half of the main program, there is a further distinction between the part that deals with value nodes and the part that deals with entity nodes, all nodes being mtol or embedded ltol relation only. The reason for this separation is again the difference in implementation outweighs similarity in spite of a lot of conceptual similarity.

Information about the two kinds of nodes are extracted from the UPDN_ARG received from the upper level. For value nodes, the fieldid is to be retrieved from the catalogue and for entity nodes, the recordid is to be retrieved with the help of the retrieval subsystem. So there would be quite a number of interfaces with the retrieval subsystem and the catalogue.

The second half deals with non-embedded ltol and ltom relationship. Combined with information obtained from the first stage, it would formulate a request for each non-embedded
relationship. As a result, there would be an array of pointers each pointing to a control block forming a request.

At preliminary glance, there would be about 20-25 subroutines. They are grouped into modules calling each other in the same level. The following is a crude flowchart of calling modules.
3.3 Data Structures

Before an explanation of the strategy is discussed, it is useful to review what is submitted to the N-ary Level. The structure is called UPDN_ARG and it is declared as follows:

dcl 1 updn_arg(p),
  2 basic_op char(1), /* 'c', 'd', or 'm' */
  2 root_psetid bit(32),
    /* unique psetid to be mapped to a unique file */
  2 n fixed bin, /* total number of non-root nodes */
  2 node_descrip(ua_ctr refer(updn_arg.n))
    3 bsetid bit(32),
      /* unique bsetid to be mapped to a field */
    3 parent fixed bin, /* node no of parent */
    3 op char(1), /* meaningful only for modify */
      'p', 'r', or 'a' */
    3 data char(40); /* numerical, character or recordid */

Each node in the tree is numbered with differently. The whole tree in the entity level is mapped into this structure before it is submitted to the N-ary Level for further mapping. The rootnode is identified with a root psetid. All the other nodes are represented as an element of an array of node_descrip. Again, each node is identified with its unique
bsetid, its parent number in the tree, its operation code, if any, and the data if it is a value node.

What is returned to the N-ary Level is the UPDN_RTN with the following declaration:

```plaintext
dcl 1 updn_rtn based(p),
    2 rtn_code fixed bin(15), /* 0= success, */
    2 error_nodeno fixed;
    /* node number with invalid recordid */
```

The `rtn_code` is indicative of the success or failure of the basic operation, create, delete or modify after the retrieval of all the recordids are successfully completed. The `error_nodeno` is indicative of which recordid cannot be retrieved. It is the node number of the entity node of this subtree whose recordid is invalid.

The structure that is submitted to the U-nary Level after all the mappings are successfully completed is as follows:

```plaintext
dcl 1 twolev_tree based(p),
    2 basic_op char(1), /* 'c', 'd' or 'm' */
    2 b_op_rtncode fixed bin,
        /* to be filled by U-nary Level */
    2 root_fileid fixed bin, /* fileid of root node */
    2 recordid fixed bin, /* recordid of root node */
```

-19-
2 no_of_field, /* total no of fields */
2 id_arr(no_fdupd refer(twolev_tree.no_of_field),
   3 fieldid fixed bin(15),
   3 data char(40); /* actual data or recordid

The root_fileid identifies the file whose contents are to be updated. Each element of the id_arr indicates the particular field to be updated and the data to be used to replace the current content.

The structure that is submitted back to the N-ary Level from the U-nary Level is the same one that is passed down. However, there is a new piece of information- the b_op_rtncode which is filled by the U-nary Level as according to whether the basic operation is successful or not.

Finally, an important structure needs to be mentioned is the catalogue where all the mappings information are extracted. It is able to map a psetid to a fileid and a bsetid to a fieldid. Also information as to the type of the bset relation, e.g., 'E' or 'N' for 1tol relation, and finally the fileid of the target node for a non-embedded 1tol relation are obtained.

dcl 1 pcat based(p),
   2 psetid bit(32),
   2 fileid fixed bin(15),

-20-
A few points need be mentioned here. Type means whether the node is a value node or entity node. Func means whether the relation is an mtol, ltol or Itom. Itype means whether or not the data value or recordid is embedded in the current file. Fileid is non-zero, of course, only when the target node is a ltom entity node or a non-embedded ltol entity node. It is obvious from the previous layout of the structure of the catalogue that practically no mappings can be accomplished without access to the information to the appropriate catalogue which corresponds to the root_psetid in the UPDN_ARG.

3.4 Strategy

From the previous discussion, it is obvious that the mapping information comes from extracting the proper catalogue...
entry and deducing all the fields to be updated. However, due to the existence of non-embedded entity nodes, i.e., those with the parameter 'N', which imply that the file of the target node needs to be modified rather than that of the source node or the rootnode, it is necessary to construct an individual twolev_tree for each such non-embedded node. The strategy is to break down the update tree into two groups—(i) embedded nodes: value nodes, mtol entity nodes, embedded 1tol entity nodes (ii) non-embedded nodes: non-embedded 1tol entity nodes, 1tom entity nodes. The reason is obvious— for embedded nodes, all the data, whether it is actual numerical or character value, are contained in the same one file, that of the rootnode. However, for non-embedded nodes, the data are contained in the file of the target nodes. Hence there is more than one file to be updated. However, the twolev_tree can only accommodate all the update information for one file. It is therefore essential to construct one twolev_tree for each file to be modified. To summarize, if the tree has two non-embedded 1tol or 1tom entity child nodes, then three twolev_trees are passed down to the U-nary Level for complete updating.

3.5 Preview of Important Functional Modules

As mentioned before, update would consist of create, delete and modify. For create and modify, it is necessary to break down the tree into embedded and non-embedded nodes. The
modules, cr2, cr19, md2, md19 are responsible for constructing the twolev_tree for the root_node. Cr20 and md20 are responsible for constructing a twolev_tree for each non-embedded entity node. For delete, only the record of one file needs to be deleted. Hence, only de2 and de19 exist. A discussion of each functional module is presented in the following chapter.
4.1 Top Level Modules

The main program directs operation to cr2, cr19 etc., as according to the basic_op. Top level procedures are those that are visible in the main program. However, an important second level subroutine must be mentioned prior to the discussion of top level procedures, cr2, cr19, cr20, de2, de19, md2, md19 and md20. The subroutine is GNI.

GNI- in principal, it extracts all the information about each node from the catalogue and the UPDN_ARG. For value nodes, it stores the fieldid and data in a structure pointed to by MVNIP; for embedded entity nodes, it stores the first node of the sub-tree and the fieldid of the field that contains the recordid in a structure pointed to by MNNIP; for non-embedded entity nodes, it also stores the initial node no in a structure pointed to by OMNIP.
CR2- it invokes GNI to obtain information on value nodes with which it uses to fill the fieldid and data for all the value nodes in the twolev_tree of the rootnode.

CR19- it also employs the information on the initial node number of embedded entity nodes obtained from the CR2 invocation of GNI. It deals primarily with embedded entity nodes. It locates the number of the last node in the subtree. Then it passes the whole subtree to the retrieval subsystem to obtain the recordid defined by this subtree. It then fills the data part of the twolev_tree with this recordid. After all the recordids are retrieved, the twolev_tree for updating the file that corresponds to the rootnode is ready for export to the U-nary Level.

CR20- it also employs the information on the initial node number of non-embedded entity nodes obtained from the CR2 invocation of GNI. It deals primarily with non-embedded nodes. It first finds the fileid of the non-embedded node from the catalogue and then generate a twolev_tree for that non-embedded node. It is obvious that this twolev_tree has only one node and the data is the recordid of the newly created record.

DE2- it maps the root_psetid to the proper fileid.

DE19- it passes the whole tree to the retrieval subsystem and then inserts the returned recordid into the twolev_tree. MD2-
essentially the same as CR2 except that not all fields are updated. In CR2, if a certain field is not specified in the UPDN_ARG, it is still created with an initial value of NULL. However, in MD2, only specified fields are updated. MD19, MD20- same as their CREATE counterparts except that it is essential to fill the twolev_tree with the information on nodes that are to be removed before those that are to be added.

ZGTCT- it extracts the appropriate catalogue that corresponds to the given root_psetid in the UPDN_ARG.

TPROC- it is responsible for submitting a twolev_tree to the U-nary Level every time it is invoked. Hence it is an indispensable interface module.

CONTROL- this is the main program monitoring the flow of control. It directs execution to the proper create, delete or modify modules. It is also responsible for interfacing with the retrieval subsystem, the Entity Level and the U-nary Level.

USER- this is the options main program which enables the user to test all the modules with user-supplied data.

4.2 Second Level Subroutines
The aforementioned top level procedures call other subroutines that do not appear in the main control program. Hence they are called Second Level procedures. They can be separated into two main kinds—those that query the extracted catalogue entry and those that do not. This section deals with those that do not.

ILN- this boolean procedure test if a node is a second level node in an update tree, UPDN_ARG.

CLN- it counts the total number of second level nodes.

CMN- it counts the total number of second level value nodes and the total number of second level embedded nodes.

GTRCID- it is responsible for interfacing with the retrieval subsystem for extracting the recordid of embedded subtrees. It does so by passing the whole subtree to the retrieval subsystem.

4.3 Query Catalogue Subroutines

The following subroutines are mainly responsible for carrying out all the mapping functions. They are therefore very important and act as the bridge between the Entity Level and the U-nary Level.
QCEMB- this boolean procedure checks whether a given node is of type embedded, 'E', or non-embedded, 'N'.

QCFDID- this procedure returns the fieldid of a given node. If it is an embedded node, it returns the field in the record of the file of the rootnode that is to be updated. If it is a non-embedded node, it returns the field in the record of the file of the target that is to be updated.

QCFLDS- this procedure returns the fileid of a target node. Hence it is invoked only when the calling procedure deals with non-embedded nodes.

QCFLID- this procedure returns the fileid of the root node.

QCNTYP- this procedure returns the type of the node. It can be value node, 'V', or an entity node, 'N'.

4.4 Conclusion

All the procedures described above are actually to be integrated into the main program of the STV to constitute the N-ary Level. In the following and last chapter, we will discuss the simulation of the environment so that the above procedures can be deterministically tested against programming bugs.
Hence the procedures that will be mentioned are not part of the N-ary Level but merely written to ensure binding testings are carried out.
Chapter Five

Simulation and Simulated Procedures

5.1 Concepts

There is a fine distinction between simulation procedures and simulated procedures. Because of the incompletion of the whole N-ary Level, some procedures that ought to be present to enable full testing of the updat subsystem will have to be simulated. However, some structures like the database should never ought be created by the N-ary Level. Nevertheless, they are coded to enable the simulation of the actual environment to the fullest extent. These are simulation procedures. Simulated procedures that ought to be in the N-ary Level have been discussed in the previous chapter. These include GTRCID, TPROC and ZGTCT. Simulation procedures are discussed in the following section.

5.2 Simulation Subroutines

Simulation procedures include SBDUA and SFLAPC. The former is responsible for interacting with the user to receive
user-supplied update trees, UPDN_ARG. The latter is responsible for building up a mini database.

SBDUA- It fires requests to the user for all the necessary information to build up a full-bloom update tree. In order, it queries the total no of non-root nodes, the basic operation, the root_psetid, for each non-root node, the bsetid, the parent, the operation code and finally the data.

SFLAPC- It is a predetermined procedure in the sense that when the user calls the main program CONTROL, it merely creates the preassigned database. The whole database is as depicted pictorially at the end of this chapter.

5.3 Conclusion

With the presence of these simulated and simulation procedures, the user can fully test the update subsystem. A sample terminal session is provided in appendix three. All the user-supplied interfaces have been coded into exec files for simplicity. Notice the large number of exec files tested. Care has been taken so that each exec file tests for different characteristics of the subsystem and overlap has been avoided to the best knowledge of the author. The exec files from 30 onwards test the modules that handle update trees with CREATE as the basic_op. The exec files from 50 onwards test the mod-
ules that handle update trees with DELETE as the basic_op. The exec files from 70 onwards test the modules that handle update trees with MODIFY as the basic_op.
Bibliography

1. Chat-Yu Lam, Stuart E. Madnick
   'INFOPLEX Data Base Computer Architecture'
   Center for Information Systems Research, M.I.T.

2. Meichun Hsu
   'FSTV - The Software Test Vehicle for the Functional
   Hierarchy of the INFOPLEX Data Base Computer'
   Center for Information Systems Research, M.I.T.

3. C. J. Date
   'An Introduction to Database Management' Second Edition
   Addison-Wesley.

4. Bruce Blumberg
   'INFOSAM - A Sample Database Management System'
   Center for Information Systems Research, M.I.T.

5. Meichun Hsu
   'Draft of the FSTV'
   Center for Information Systems Research, M.I.T.
Appendix 1- Program Listings

The following P/L1 program modules are included in this appendix in order.

GNI
CR2
CR19
CR20
DE2
DE19
DE20
MD2
MD19
MD20
ZGTCT
TPROC
-CONTROL
USER
ILN
CLN
CMN
GTRCID
QCEMB
QCFDID
QCFLDS
*PROCESS NAME('GNI'), INCLUDE, F(I):

GNI: PROC (UAP, BASIC_OP, MVNIP, MNNIP, OMNIP,
NO_MTO1VNODE, NO_MTO1NNODE, NO_1TOMNODE, CP, CT_NO);

/* GNI= GET_NODEINFO */

%INCLUDE MVNI;
%INCLUDE MNNI;
%INCLUDE ONI;
%INCLUDE UA;

DCL P PTR;
DCL ILN ENTRY EXTERNAL RETURNS(Bit),
QCNTYP ENTRY EXTERNAL RETURNS(CHAR(1)),
QCFDID ENTRY EXTERNAL RETURNS(FIXED BIN(31)),
QCEMB ENTRY EXTERNAL RETURNS(Bit);

DCL UAP PTR,
BASIC_OP CHAR(1),
(MVNIP, MNNIP, OMNIP) PTR,
(NO_MTO1VNODE, NO_MTO1NNODE, NO_1TOMNODE) FIXED,
CP PTR,
CT_NO FIXED;

DCL (VIA_CTR, NIA_CTR, ONIA_CTR, NODE_NO) FIXED INIT(0),
BSETID FIXED BIN(15) INIT(0);

VIA_CTR= NO_MTO1VNODE;
NIA_CTR= NO_MTO1NNODE;
ONIA_CTR= NO_1TOMNODE;
IF NO_MTO1VNODE^= 0 THEN
DO;
ALLOCATE MTO1VNODE_INFO SET (MVNIP);
MVNIP-> MTO1VNODE_INFO.INFO_ARR(*).NODE_NO= 0;
MVNIP-> MTO1VNODE_INFO.INFO_ARR(*).FIELDID= 0;
MVNIP-> MTO1VNODE_INFO.INFO_ARR(*).DATA=' ';
END;
IF NO_MTO1NNODE^= 0 THEN
DO;
ALLOCATE MTO1NNODE_INFO SET (MNNIP);
MNNIP-> MTO1NNODE_INFO.INFO_ARR(*).INITIAL_NODE_NO= 0;
MNNIP-> MTO1NNODE_INFO.INFO_ARR(*).FINAL_NODE_NO= 0;
MNNIP-> MTO1NNODE_INFO.INFO_ARR(*).FIELDID= 0;
END;
IF NO_1TOMNODE^= 0 THEN
DO;
ALLOCATE OTOMNODE_INFO SET (OMNIP);
OMNIP-> OTOMNODE_INFO.INFO_ARR(*) = 0;
END;
VIA_CTR = 0;
NIA_CTR = 0;
ONIA_CTR = 0;

DO NODE_NO = 1 TO UAP-> UPDN_ARG.N;
  IF ILN(UAP, NODE_NO)
  THEN
    DO;
      IF BASIC_OP = 'M'
          & UAP-> UPDN_ARG.NODE_DESCRIP(NODE_NO)
          .OP = 'I'
      THEN GOTO EOL;
      BSETID = UAP-> UPDN_ARG.NODE_DESCRIP(NODE_NO).BSETID;
      IF QCNTYP(CP, CT_NO, BSETID) = 'V'
      THEN DO;
        VIA_CTR = VIA_CTR + 1;
        MVNIP-> MTO1VNODE_INFO.INFO_ARR(VIA_CTR).
        NODE_NO = NODE_NO;
        MVNIP-> MTO1VNODE_INFO.INFO_ARR(VIA_CTR).
        FIELDID = QCFDID(CP, CT_NO, BSETID);
        MVNIP-> MTO1VNODE_INFO.INFO_ARR(VIA_CTR).
        DATA = UAP-> UPDN_ARG.NODE_DESCRIP
                  (NODE_NO).DATA;
      END;
      ELSE;
      END;
  ELSE;
    IF QCEMB(CP, CT_NO, BSETID)
    THEN DO;
      NIA_CTR = NIA_CTR + 1;
      MNNIP-> MTO1NNODE_INFO.INFO_ARR(NIA_CTR).
      INITIAL_NODE_NO = NODE_NO;
      MNNIP-> MTO1NNODE_INFO.INFO_ARR(NIA_CTR).
      FIELDID = QCFDID(CP, CT_NO, BSETID);
    END;
    ELSE;
    END;
  END;
END;

EOL:
END;
END GNI;
FILE: CR2    PLIOPT A    VM/SP CONVERSATIONAL MONITOR SYSTEM

*PROCESS NAME('CR2'), INCLUDE, F(I);

CR2: PROC (UAP, BASIC_OP, MVNIP, MNNIP, OMNIP,
            PTLTP, NO_MTO1VNODE, NO_MTO1NNODE, NO_1TOMNODE,
            CP, CT_CTR);

/* CR2= CREATE2 */

%INCLUDE TWOLEV;
%INCLUDE MVNI;
%INCLUDE UA;
%INCLUDE SPCT;

DCL P PTR;
DCL CLN ENTRY EXTERNAL RETURNS(FIXED),
CMN ENTRY EXTERNAL,
QCFLID ENTRY EXTERNAL RETURNS(FIXED BIN(31)),
GNI ENTRY EXTERNAL,
ILN ENTRY EXTERNAL RETURNS(BIT);

DCL UAP PTR,
BASIC_OP CHAR(1),
(MVNIP, MNNIP, OMNIP, PTLTP) PTR,
(NO_MTO1VNODE, NO_MTO1NNODE, NO_1TOMNODE) FIXED,
CP PTR,
CT_CTR FIXED;

DCL (NO_LEVEL2NODE, NO_FDUPD, VIA_CTR, ND_CTR, FD_CTR,
     _IDARR_CTR) FIXED INIT(0);

NO_LEVEL2NODE= CLN(UAP);
CALL CMN(UAP, BASIC_OP, NO_MTO1VNODE, NO_MTO1NNODE, CP, CT_CTR);
DO FD_CTR= 1 TO CP-> PCAT(C_T_CTR).N;
   IF CP-> PCAT(C_T_CTR).BSET_DESCRIP(FD_CTR).IMP.ITYPE= 'E'
      THEN NO_FDUPD= NO_FDUPD + 1;
END;

NO_1TOMNODE= NO_LEVEL2NODE - NO_MTO1VNODE - NO_MTO1NNODE;
ALLOCATE TWOLEV_TREE SET (PTLTP);
PTLTP-> TWOLEV_TREE.BASIC_OP= ' ';  
PTLTP-> TWOLEV_TREE.B_OP_RTNCODE= 0;  
PTLTP-> TWOLEV_TREE.ROOT_FILEID= 0;  
PTLTP-> TWOLEV_TREE.RECORDID= 0;  
PTLTP-> TWOLEV_TREE.ID_ARR(*) .FIELDID= 0;  
PTLTP-> TWOLEV_TREE.ID_ARR(*) .DATA= ' ';  
PTLTP-> TWOLEV_TREE.BASIC_OP= BASIC_OP;
PTLTP-> TWOLEV_TREE.ROOT_FILEID= QCFLID(CP, CT_CTR);
DO FD_CTR= 1 TO CP-> PCAT(C_T_CTR).N;
FILE: CR2   PLIOPT A   VM/SP CONVERSATIONAL MONITOR SYSTEM

IF CP-> PCAT(CTCTR).BSET_DESCRIP(FDCTR).IMP.ITYPE= 'E'
  THEN DO; IDARR_CTR= IDARR_CTR + 1;
  PTLTP-> TWOLEV_TREE.ID_ARR(IDARR_CTR).FIELDID
    = CP-> PCAT(CTCTR).BSET_DESCRIP(IDARR_CTR).
      IMP.FIELDID;
  PTLTP-> TWOLEV_TREE.ID_ARR(IDARR_CTR).DATA
    = 'NULL';
  END;
END;

CALL GNI(UAP, BASIC_OP, MVNIP, MNIP, OMNIP,
  NO_MT01VNODE, NO_MT01NNODE, NO_ITOMNODE, CP, CTCTR);

DO VIA_CTR=1 TO NO_MT01VNODE;
  DO FD_CTR= 1 TO PTLTP-> TWOLEV_TREE.NO_OF_FIELD;
    IF PTLTP-> TWOLEV_TREE.ID_ARR(FD_CTR).FIELDID
      = MVNIP-> MT01VNODE_INFO.INFO_ARR(VIA_CTR).FIELDID
      THEN PTLTP-> TWOLEV_TREE.ID_ARR(FD_CTR).DATA
        = MVNIP-> MT01VNODE_INFO.INFO_ARR(VIA_CTR).DATA;
  END;
END;

END CR2;

FREE MVNIP-> MT01VNODE_INFO */

END CR2;
• PROCESS NAME('CR19'), INCLUDE, F(I);

CR19: PROC (UAP, MNNIP, PTLTP, ERR_NODE, CP, FP, CT_NO);

/* CR19= CREATE19 */

/* GTRCID CHECKS FP-> FILE(10).RECORD(2).FILE(3) FOR VALID RECORDID. HENCE TO TEST FOR INVALID RECORDID, ONE CAN CHANGE THIS FIELD ENTRY ONE HAS TO TEST FOR TWO CASES 1) RECORDID VALID 2) RECORDID INVALID */

FILE: CR19 PLIOPT A
RC_RTN_CODE = GTRCID(UAP, INITIAL_NODE_NO, FINAL_NODE_NO,
                    RECORDID, CP, FP, CT_NO);

IF RC_RTN_CODE ^= 0
   THEN DO:
       ERR_NODE = INITIAL_NODE_NO;
       GOTO RETN;
   END;
DO FD_CTR = 1 TO PTLP-> TWOLEV_TREE.NO_OF_FIELD;
   IF PTLP-> TWOLEV_TREE.ID_ARR(FD_CTR).FIELDID
      ^= MNNIP-> MTO1NNODE_INFO.INFO_ARR(NIA_CTR).FIELDID
      THEN PTLP-> TWOLEV_TREE.ID_ARR(FD_CTR).DATA
        ^= RECORDID;
   END;
RETN:
    /* FREE MNNIP-> MTO1NNODE_INFO */
END CR19:
*PROCESS NAME('CR20'), INCLUDE, F(I):
CR20: PROC (UAP, OMNIP, PTLTP, ERR_NODE, PAP, CP, FP, CT_NO);

/* CR20= CREATE20 */
/* 2 CASES TO TEST FOR:
  1) RECORDID OF THE 1TOMNODE VALID
  2) RECORDID OF THE 1TOMNODE INVALID */

%INCLUDE UA;
%INCLUDE TWOLEV;
%INCLUDE ONI;
%INCLUDE PA;

DCL NULL BUILTIN;
DCL QCFLDS ENTRY EXTERNAL RETURNS(FIXED BIN(31));
DCL ILN ENTRY EXTERNAL RETURNS(BIT);
DCL GTCCD ENTRY EXTERNAL RETURNS(FIXED);
DCL QCFCID ENTRY EXTERNAL RETURNS(FIXED BIN(31));

DCL P PTR;
DCL (UAP, OMNIP, PTLTP) PTR,
  ERR_NODE FIXED,
  (PAP, CP, FP) PTR,
  CT_NO FIXED;

DCL BASIC_OP CHAR(1) INIT(' '),
  NO_FDUPD FIXED INIT(0),
  ROOT_RECORDID FIXED BIN(31) INIT(0),
  (ONIA_CTR, NO_1TOMNODE, NODE_NO) FIXED INIT(0),
  BSETID FIXED BIN(15) INIT(0),
  STLTP PTR INIT(NULL()),
  (INITIAL_NODE_NO, RTNTND_CTR, FINAL_NODE_NO, RC_RTNCODE) FIXED INIT(0),
  RECORDID FIXED BIN(31) INIT(0),
  PA_CTR FIXED INIT(0);

BASIC_OP= 'M';
NO_FDUPD= 1;
DO ONIA_CTR= 1 TO OMNIP-> OTOMNODE_INFO_NO_1TOMNODE;
  NODE_NO= OMNIP-> OTOMNODE_INFO_INFO_ARR(ONIA_CTR);
  BSETID= UAP-> UPDN_ARG_NODE_DESCRIP(NODE_NO), BSETID;
  ALLOCATE TWOLEV_TREE SET (STLTP);
  STLTP-> TWOLEV_TREE.BASIC_OP= ' ';
  STLTP-> TWOLEV_TREE.B.OP_RTNCODE= 0;
  STLTP-> TWOLEV_TREE.ROOT_FILEID= 0;
  STLTP-> TWOLEV_TREE.RECORDID= 0;
  STLTP-> TWOLEV_TREE.ID_ARR.FIELDID= 0;
  STLTP-> TWOLEV_TREE.ID_ARR.DATA= ' ';
  STLTP-> TWOLEV_TREE.BASIC_OP= QCFLDS(CP, CT_NO, BSETID);
  INITIAL_NODE_NO= NODE_NO;
DO RTNTND_CTR= INITIAL_NODE_NO + 1 TO UAP->UPDN_ARG.N; CR200560
   IF RTNTND_CTR= UAP-> UPDN_ARG.N CR200570
      THEN FINAL_NODE_NO= RTNTND_CTR; CR200580
   ELSE IF ILN(UAP, RTNTND_CTR) CR200590
      THEN DO; CR200600
         FINAL_NODE_NO= RTNTND_CTR - 1; CR200610
         RTNTND_CTR= UAP-> UPDN_ARG.N + 1; CR200620
      END; CR200630
   END; CR200640
RC_RTNCODE= GTRCID(UAP, INITIAL_NODE_NO, FINAL_NODE_NO, CR200650
   RECORDID, CP, FP, CT_NO); CR200660
IF RC_RTNCODE^= 0 CR200670
   THEN DO;
      ERR_NODE= INITIAL_NODE_NO;
      GOTO RETN;
   END; CR200680
STLTP-> TWELEV_TREE.RECORDID= RECORDID; CR200720
STLTP-> TWELEV_TREE.ID_ARR(1).FIELDID= QCFDID(CP, CT_NO, CR200730
   BSETID);
PA_CTR= PA_CTR + 1; CR200740
PAP-> PTR_ARR.ARR(PA_CTR)= STLTP; CR200750
END; CR200760
RETN: CR200770
/* FREE OMNIP-> OTOMNODE_INFO */ CR200780
CR200790
FILE: DE2      PLOPT A
VM/SP CONVERSATIONAL MONITOR SYSTEM

*PROCESS NAME('DE2'), INCLUDE, F(I):
DE2: PROC (BASIC_OP, PTLTP, CP, CTCTR);

/* DE2= DELETE2 */

%INCLUDE TWOLEV;

DCL QCFLID ENTRY EXTERNAL RETURNS(FIXED BIN(31));

DCL P PTR;
DCL BASIC_OP CHAR(1),
(PTLTP, CP) PTR,
CTCTR FIXED;
DCL NOFDUPD FIXED INIT(0);

NOFDUPD= 1;
ALLOCATE TWOLEV_TREE SET (PTLTP);
PTLTP-> TWOLEV_TREE.BASIC_OP= ' ';
PTLTP-> TWOLEV_TREE.B_OP RTNCODE= 0;
PTLTP-> TWOLEV_TREE.ROOT_FILEID= 0;
PTLTP-> TWOLEV_TREE.RECORDID= 0;
PTLTP-> TWOLEV_TREE.ID_ARR(*).FIELDID= 0;
PTLTP-> TWOLEV_TREE.IDARR(*).DATA=' ';
PTLTP-> TWOLEV_TREE.BASIC_OP= BASIC_OP;
PTLTP-> TWOLEV_TREE.ROOT_FILEID= QCFLID(CP, CTCTR);

END DE2;
*PROCESS NAME('DE19'), INCLUDE, F(I);
DE19: PROC (UAP, PTLTP, ERR_NODE, CP, FP, CT_NO);
/* DE19= DELETE19 */
/ * AS A RESULT OF SETTING INITIAL_NODE NO= 1, GTRCID CHECKS
 FP-> FILE(9).RECORD(2).FIELD(2) FOR A VALID RECORD NO
 ONE HAS TO TEST FOR TWO CASES
 1) RECORDID FOR ROOT_NODE VALID
 2) RECORDID FOR ROOT NODE INVAlID  */

%INCLUDE UA;
%INCLUDE TWOLEV;

DCL P PTR;
DCL GTRCID ENTRY EXTERNAL RETURNS(FIXED);

DCL (UAP, PTLTP) PTR,
    ERR_NODE FIXED,
    (CP, FP) PTR,
    CT_NO FIXED;

DCL (INITIAL_NODE_NO, FINAL_NODE_NO, RC_RTNODE)
    FIXED INIT(O),
    RECORDID FIXED BIN(31) INIT(O);

INITIAL_NODE_NO= 1;
FINAL_NODE_NO= UAP-> UPDN_ARG.N;
RC_RTNODE= GTRCID(UAP, INITIAL_NODE_NO, FINAL_NODE_NO,
    RECORDID, CP, FP, CT_NO);
IF RC_RTNODE= 0
    THEN DO:
        ERR_NODE= -1;
        RETURN;
    END;
PTLTP-> TWOLEV_TREE.RECORDID= RECORDID;

END DE19;
*PROCESS NAME('MD2'), INCLUDE, F(I);

MD2: PROC (UAP, BASIC_OP, MVNIP, MNNIP, OMNIP,
PTLTTP, NO_MTO1VNODE, NO_MTO1INNODE, NO_1TOMNODE,
IDARR_CTR, CP, CT_CTR):

MD200010
MD200020
MD200030
MD200040
MD200050
MD200060
MD200070
MD200080
MD200090
MD200100
MD200110
MD200120
MD200130
MD200140
MD200150
MD200160
MD200170
MD200180
MD200190
MD200200
MD200210
MD200220
MD200230
MD200240
MD200250
MD200260
MD200270
MD200280
MD200290
MD200300
MD200310
MD200320
MD200330
MD200340
MD200350
MD200360
MD200370
MD200380
MD200390
MD200400
MD200410
MD200420
MD200430
MD200440
MD200450
MD200460
MD200470
MD200480
MD200490
MD200500
MD200510
MD200520
MD200530
MD200540
MD200550

/* MD2* MODIFY2 */

%INCLUDE TWOLEV;
%INCLUDE MVNI;
%INCLUDE UA:

DCL P PTR;
DCL CLN ENTRY EXTERNAL RETURNS(FIXED),
CMN ENTRY EXTERNAL,
OCFLID ENTRY EXTERNAL RETURNS(FIXED BIN(31)),
GNI ENTRY EXTERNAL,
ILN ENTRY EXTERNAL RETURNS(BIT);

DCL UAP PTR,
BASIC_OP CHAR(1),
(MVNIP, MNNIP, OMNIP, PTLTP) PTR,
(IDARR_CTR, NO_MTO1VNODE, NO_MTO1INNODE, NO_1TOMNODE) FIXED,
CP PTR,
CT_CTR FIXED;

DCL (NO_LEVEL2NODE, NO_FDUPD, VIA_CTR, NO_CTR) FIXED INIT(0);

NO_LEVEL2NODE= CLN(UAP);
CALL CMN(UAP, BASIC_OP, NO_MTO1VNODE, NO_MTO1INNODE, CP, CT_CTR);
NO_FDUPD= NO_MTO1VNODE + NO_MTO1INNODE;
NO_1TOMNODE= NO_LEVEL2NODE - NO_FDUPD;
DO NO_CTR= 1 TO UAP->UPDN_ARG.N;
   IF ILN(UAP, NO_CTR)
      & UAP->UPDN_ARG.NODE_DESCRIOP(NO_CTR).OP='I'
      THEN NO_1TOMNODE= NO_1TOMNODE - 1;
   END;

IF NO_FDUPD= 0
   THEN NO_FDUPD= 1;
ALLOCATE TWOLEV_TREE SET (PTLTTP);
PTLTTP-> TWOLEV_TREE.BASIC_OP= ' ';
PTLTP-> TWOLEV_TREE.ROOT_FILEID= OCFLID(CP, CT_CTR);

CALL GNI(UAP, BASIC_OP, MVNIP, MNNIP, OMNIP,
          NO_MTO1VNOD, NO_MTO1NNODE, NO_ITOMNODE, CP, CT_CTR);

DO VIA_CTR= 1 TO NO_MTO1VNOD;
   IDARR_CTR = IDARR_CTR + 1;
   PTLTP-> TWOLEV_TREE.ID_ARR(IDARR_CTR).FIELDID = 
          MVNIP-> MTO1VNOD_INFO.INFO_ARR(VIA_CTR).FIELDID;
   PTLTP-> TWOLEV_TREE.ID_ARR(IDARR_CTR).DATA = 
          MVNIP-> MTO1VNOD_INFO.INFO_ARR(VIA_CTR).DATA;
END;

END MD2;

FILE: MD2    PLIOPT    A    VM/SP CONVERSATIONAL MONITOR SYSTEM    PAGE 002

/* FREE MVNIP-> MTO1VNOD_INFO */

END MD2;
/* PROCESS NAME('MD19'), INCLUDE, F(I); */

MD19: PROC (UAP, MNNIP, PTLTP, NO_MTOINNODE,
               IDARR_CTR, ERR_NODE, CP, FP, CT_NO);

MD19 = MODIFY19 */

/* INITIAL_NODE_NO SHOULD BE SET TO 0 INSTEAD OF 1
   BUT FOR THE SAKE OF SIMPLICITY DURING RETRIEVAL
   OF RECORDID, IT HAS BEEN SET TO 1.
   DURING CHECKING OF VALID RECORDID, IT CHECKS
   FILE(ROOT_FILEID).RECORD(2).FIELD(FIELDID OF FIRST NODE) */

/* THERE ARE 7 CASES TO TEST FOR:
   1. RECORDID FOR ROOT_NODE NOT IN
   2. RCDID FOR R_N IN, RCDID FOR MTO1 NODE NOT IN
   3. RCDID FOR R_N IN, RCDID FOR MTO1 NODE IN, OP IS 'R'
   4. RCDID FOR R_N IN, RCDID FOR MTO1 NODE IN, OP IS NOT 'R'
      RECORD IN
   5. RCDID FOR R_N IN, RCDID FOR MTO1 NODE IN, OP IS NOT 'R',
      RECORD NOT IN
   6. MIXED ORDERED 'R' AND 'A' MTOINNODES */

%INCLUDE MNNI;
%INCLUDE UA;
%INCLUDE TWOLEV;

DCL ILN ENTRY EXTERNAL RETURNS(BIT);
DCL GTRCID ENTRY EXTERNAL RETURNS(FIXED);

DCL P PTR;
DCL (UAP, MNNIP, PTLTP) PTR,
     (NO_MTOINNODE, IDARR_CTR, ERR_NODE) FIXED,
     (CP, FP) PTR,
     CT_NO FIXED;

DCL (INITIAL_NODE_NO, RTNTND_CTR, FINAL_NODE_NO, NIA_CTR,
     RC_RTNCODE, RUN) FIXED INIT(0),
     RECORDID FIXED BIN(31) INIT(0);

INITIAL_NODE_NO = 1;
DO RTNTND_CTR = 1 TO UAP-> UPDN_ARG.N;
   IF UAP-> UPDN_ARG.NODE_DESCRIP(RTNTND_CTR).OP = 'I'
      THEN DO;
         FINAL_NODE_NO = RTNTND_CTR - 1;
         RTNTND_CTR = UAP-> UPDN_ARG.N + 1;
      END;
END;
RC_RTNCODE= GTRCID(UAP, INITIAL_NODE_NO, FINAL_NODE_NO, RECORDID,MD100560
   CP, FP, CT_NO);
IF RC_RTNCODE= O
   THEN DO:
      ERR_NODE= -1;
      GOTO RETN;
END;
PTLTP-> TOWLEV_TREE.RECORDID= RECORDID;
IF NO_MTO1NNODE= O
   THEN
   DO:
      NIA_CTR= 1 TO MNNIP-> MTO1NNODEINFO.NO_MTO1NNODE:
      INITIAL_NODE_NO= MNNIP-> MTO1NNODEINFO.INFO.ARR(NIA_CTR).
      INITIAL_NODE_NO;
      DO RTNTND_CTR= INITIAL_NODE_NO + 1 TO UAP-> UPDN_ARG.N:
         IF RTNTND_CTR= UAP-> UPDN_ARG.N
            THEN MNNIP-> MTO1NNODEINFO.INFO.ARR(NIA_CTR).
            FINAL_NODE_NO= RTNTND_CTR;
      ELSE IF ILN(UAP, RTNTND_CTR)
         THEN DO:
            MNNIP-> MTO1NNODEINFO.INFO.ARR(NIA_CTR).
            FINAL_NODE_NO= RTNTND_CTR - 1;
            RTNTND_CTR= UAP-> UPDN_ARG.N + 1;
         END;
      END;
   END;
   END:
   RUN= 1:
LOOP:
   DO NIA_CTR= 1 TO MNNIP-> MTO1NNODEINFO.NO_MTO1NNODE:
      INITIAL_NODE_NO= MNNIP-> MTO1NNODEINFO.INFO.ARR(NIA_CTR).
      INITIAL_NODE_NO;
      IF RUN= 1
         THEN IF UAP-> UPDN_ARG.NODE_DESCRIP(INITIAL_NODE_NO).
            OP= 'A'
            THEN GOTO EOL;
      IF RUN= 2
         THEN IF UAP-> UPDN_ARG.NODE_DESCRIP(INITIAL_NODE_NO).
            OP= 'R'
            THEN GOTO EOL;
      IF UAP-> UPDN_ARG.NODE_DESCRIP(INITIAL_NODE_NO).OP= 'R'
         THEN RECORDID= 0;
      ELSE DO:
         FINAL_NODE_NO= MNNIP-> MTO1NNODEINFO.INFO.ARR
            (NIA_CTR).FINAL_NODE_NO;
         RC_RTNCODE= GTRCID(UAP, INITIAL_NODE_NO,
            FINAL_NODE_NO, RECORDID,
            CP, FP, CT_NO);
         IF RC_RTNCODE= O
            THEN DO:
               ERR_NODE= INITIAL_NODE_NO;
               GOTO RETN;
            END;
      END;
IDARR_CTR= IDARR_CTR + 1;
PTLTP-> TWELEV_TREE.ID_ARR(IDARR_CTR).FIELDID
= MNNIP-> MTO1NNODE_INFO.INFO_ARR(NIA_CTR).
FIELDID;
PTLTP-> TWELEV_TREE.ID_ARR(IDARR_CTR).DATA
= RECORDID;
EOL:
END;
IF RUN= 1
THEN DO:
  RUN= 2;
  GOTO LOOP;
END;
RETN:
/* FREE MNNIP-> MTO1NNODE_INFO */
END MD19:
PROCESS NAME('MD20'), INCLUDE, F(I);

/* MD20 = MODIFY20 */
/* 2 CASES TO TEST FOR: VALID AND INVALID RECORDID */
/* ALSO TEST FOR PROPER REORDERING OF 'R' AND 'A' NODES 
THE CHANGES ARE SPCT, SFLAPC TO SFLAPC1, AND ZGTC */

MD20: PROC (UAP, OMNIP, PTLTP, ERR_NODE, PAP, CP, FP, CT_NO);

%INCLUDE UA;
%INCLUDE TWOLEV;
%INCLUDE ONI;
%INCLUDE PA;

DCL NULL BUILTIN;
DCL QCFLDS ENTRY EXTERNAL RETURNS(FIXED BIN(31));
DCL I1N ENTRY EXTERNAL RETURNS(BIT);
DCL GRTRCID ENTRY EXTERNAL RETURNS(FIXED);
DCL QCFCID ENTRY EXTERNAL RETURNS(FIXED BIN(31));

DCL P PTR;
DCL (UAP, OMNIP, PTLTP) PTR,
ERR_NODE FIXED,
(PAP, CP, FP) PTR,
CT_NO FIXED;

DCL BASIC_OP CHAR(1) INIT(' '),
NO_FDUPD FIXED INIT(0),
ROOT_RECORDID FIXED BIN(31) INIT(0),
RUN FIXED INIT(0),
(ONIA_CTR, NO_1TOMNODE, NODE_NO) FIXED INIT(0),
BSETID FIXED BIN(15) INIT(0),
STLTP PTR INIT(NULL()),
(INITIAL_NODE_NO, RTNMD_CTR, FINAL_NODE_NO, RC_RTNCODE)
FIXED INIT(0),
RECORDID FIXED BIN(31) INIT(0),
PA_CTR FIXED INIT(0);

BASIC_OP = 'M';
NO_FDUPD = 1;
ROOT_RECORDID = PTLTP-> TWOLEV_TREE.RECORDID;
RUN = 1;

LOOP:
DO ONIA_CTR = 1 TO OMNIP-> OTOMNODE_INFO.NO_1TOMNODE;
NODE_NO = OMNIP-> OTOMNODE_INFO.INFO_ARR(ONIA_CTR);
IF RUN = 1 
THEN IF UAP-> UPDN_ARG.NODE_DESCRIP(NODE_NO).OP = 'A'
THEN GOTO EOL;
IF RUN = 2 
THEN IF UAP-> UPDN_ARG.NODE_DESCRIP(NODE_NO).OP = 'R'

MD200010
MD200020
MD200030
MD200040
MD200050
MD200060
MD200070
MD200080
MD200090
MD200100
MD200110
MD200120
MD200130
MD200140
MD200150
MD200160
MD200170
MD200180
MD200190
MD200200
MD200210
MD200220
MD200230
MD200240
MD200250
MD200260
MD200270
MD200280
MD200290
MD200300
MD200310
MD200320
MD200330
MD200340
MD200350
MD200360
MD200370
MD200380
MD200390
MD200400
MD200410
MD200420
MD200430
MD200440
MD200450
MD200460
MD200470
MD200480
MD200490
MD200500
MD200510
MD200520
MD200530
MD200540
MD200550
THEN GOTO EOL;
BSETID= UAP-> UPDN_ARG.NODE_DESCRIP(NODE_NO).BSETID;
ALLOCATE TWOLEV_TREE SET (STLTP);
STLTP-> TWOLEV_TREE.BASIC_OP= '/';
STLTP-> TWOLEV_TREE.B_OP_RTNCODE= 0;
STLTP-> TWOLEV_TREE.ROOT_FILEID= 0;
STLTP-> TWOLEV_TREE.RECORDID= 0;
STLTP-> TWOLEV_TREE.ID_ARR.FIELDID= 0;
STLTP-> TWOLEV_TREE.ID_ARR.DATA= ' '; 
STLTP-> TWOLEV_TREE.BASIC_OP= BASIC_OP;
STLTP-> TWOLEV_TREE.ROOT_FILEID= QCFLDS(CP, CT_NO, BSETID);
INITIAL_NODE_NO= NODE_NO;
DO RTNTND_CTR= INITIAL_NODE_NO + 1 TO UAP-> UPDN_ARG.N;
IF RTNTND_CTR= UAP-> UPDN_ARG.N
THEN FINAL_NODE_NO= RTNTND_CTR;
ELSE IF ILN(UAP, RTNTND_CTR)
THEN DO:
    FINAL_NODE_NO= RTNTND_CTR - 1;
    RTNTND_CTR= UAP-> UPDN_ARG.N + 1;
END;
RC_RTNCODE= GTRCID(UAP, INITIAL_NODE_NO, FINAL_NODE_NO, 
RECORDID, CP, FP, CT_NO);
IF RC_RTNCODE** 0
THEN DO:
    ERR_NODE= INITIAL_NODE_NO;
    GOTO RETN;
END;
STLTP-> TWOLEV_TREE.RECORDID= RECORDID;
STLTP-> TWOLEV_TREE.ID_ARR(1).FIELDID= QCFDID(CP, CT_NO, 
BSETID);
STLTP-> TWOLEV_TREE.ID_ARR(1).DATA= ROOT_RECORDID;
PA_CTR= PA_CTR + 1;
PAP-> PTR_ARR.ARR(PA_CTR)= STLTP;
EOL:
END;
IF RUN= 1
THEN DO:
    RUN= 2;
    GOTO LOOP;
END;
RETN:
/* FREE OMNIP-> OTOMNODE_INFO */
END MD20:
*PROCESS NAME('ZGTCT'), INCLUDE, F(I);
ZGTCT: PROC(CP, ROOT_PSETID) RETURNS(FIXED);
/* ZGTCT= GET_CAT_ENTRY */

%INCLUDE SPCT;
DCL P PTR;
DCL CP PTR,
   ROOT_PSETID FIXED BIN(15);
DCL CT_NO FIXED INIT(0);

DO CT_NO= 1 TO 5;
   IF CP->PCAT(CT_NO).PSETID= ROOT_PSETID
      THEN RETURN(CT_NO);
END;

END ZGTCT;
*PROCESS NAME('TPROC'), INCLUDE, F(I);
TPROC: PROC(PTLTP):

/* TPROC= THE INTER-LEVEL CALLING PROCEDURE */

%INCLUDE TWOLEV;

DCL MOD BUILTIN;
DCL P PTR;
DCL PTLTP PTR;
DCL RANDOM1 FIXED INIT(0), RANDOM2 FIXED INIT(0), RANDOM3 FIXED INIT(0), RANDOM FIXED INIT(0);

IF PTLTP-> TWOLEV_TREE.RECORDID= 0
  & PTLTP-> TWOLEV_TREE.BASIC_OP= 'C'
  THEN PTLTP-> TWOLEV_TREE.RECORDID= 4;
SELECT(PTLTP-> TWOLEV_TREE.BASIC_OP);
  WHEN('C', ) RANDOM1= 0;
  WHEN('D', ) RANDOM1= 1;
  WHEN('M', ) RANDOM1= 2;
END;

RANDOM2= MOD(PTLTP-> TWOLEV_TREE.ROOT_FILEID, 3);
RANDOM3= PTLTP-> TWOLEV_TREE.NO_OF_FIELD;
RANDOM= MOD(RANDOM1 + RANDOM2 + RANDOM3, 5);
SELECT(RANDOM);
  WHEN(0, 1, 2, 3) PTLTP-> TWOLEV_TREE.B_OP_RTNCODE= 0;
  WHEN(4) PTLTP-> TWOLEV_TREE.B_OP_RTNCODE= 'O';
END;

END TPROC;
FILE: CONTROL PLOPT A

VM/SP CONVERSATIONAL MONITOR SYSTEM

FILE: CONTROL PLOPT A

*PROCESS NAME('CONTROL'), INCLUDE, F(I);
CONTROL: PROC(UAP) RETURNS(PTR);

%INCLUDE UA:
/* UP TO CMN */

%INCLUDE TWOLEV;
%INCLUDE MVNI;
%INCLUDE MNNI;
%INCLUDE ONI;
/* UP TO CRMD2 AND GNI */

%INCLUDE PA:
/* UP TO CR20 */

%INCLUDE UR:
/* FINISHING TOUCH */

DCL P PTR;
DCL SBDUA ENTRY EXTERNAL RETURNS(PTR);
DCL CLN ENTRY EXTERNAL RETURNS(FIXED);
DCL NULL BUILTIN;
DCL UAP PTR,
NO_LEVEL2NODE FIXED INIT(O);
/* UP TO CLN */

DCL SFAPC ENTRY EXTERNAL,
ZGCT ENTRY EXTERNAL RETURNS(FIXED),
CMN ENTRY EXTERNAL;

DCL (FP, CP) PTR INIT(NULL()),
CT_NO FIXED INIT(O),
(NO_MTO1VNODE, NO_MTO1NNODE) FIXED INIT(O);
/* UP TO CMN */

DCL (CR2, GNI) ENTRY EXTERNAL;
DCL (MVNI, MNP, OMNIP, PTLTP) PTR INIT(NULL()),
(IDARR_CTR, NO_1TOMNODE, PR_CTR) FIXED INIT(O);
/* UP TO CR2 AND GNI */

DCL DE2 ENTRY EXTERNAL;
/* UP TO DE2 */

DCL MD2 ENTRY EXTERNAL;
/* UP TO MD2 */

DCL CR19 ENTRY EXTERNAL;
DCL ERR_NODE FIXED INIT(O);
/* UP TO CR19 */
DCL DE19 ENTRY EXTERNAL;
/* UP TO DE19 */

DCL MD19 ENTRY EXTERNAL;
/* UP TO MD19 */

DCL CR20 ENTRY EXTERNAL;
DCL PA_CTR FIXED INIT(0),
   PAP_PTR INIT(NULL());
/* UP TO CR20 */

DCL MD20 ENTRY EXTERNAL;
/* UP TO MD20 */

DCL TPROC ENTRY EXTERNAL;
DCL URP_PTR INIT(NULL());
(/* FINISHING TOUCH */

UAP= SBDUA();
NO_LEVEL2NODE= CLN(UAP);
PUT SKIP LIST(‘NUMBER OF LEVEL 2 NODES IS’, NO_LEVEL2NODE);
/* UP TO CLN */

CALL SFLAPC(FP, CP);
CT_NO= ZGTCT(CP, UAP-> UPDN_ARG.ROOT_PSETID);
CALL CMN(UAP, UAP-> UPDN_ARG.BASIC_OP, NO_MTO1VNODE, NO_MTO1NNODE,
   CP, CT_NO);
PUT SKIP LIST(‘NUMBER OF MTO1 VALUE NODES IS’, NO_MTO1VNODE);
PUT SKIP LIST(‘NUMBER OF MTO1 ENTITY NODES IS’, NO_MTO1NNODE);
/* UP TO CMN */

SELECT(UAP-> UPDN_ARG.BASIC_OP);
   WHEN (‘C’, ‘M’)
      CALL CR2(UAP, UAP-> UPDN_ARG.BASIC_OP, MNNIP, MNNIP,
            OMNIP, PTLTP, NO_MTO1VNODE, NO_MTO1NNODE,
            NO_1TOMNODE, CP, CT_NO);
      WHEN (‘D’)
      CALL DE2(UAP-> UPDN_ARG.BASIC_OP, PTLTP, CP, CT_NO);
      WHEN (‘M’)
      CALL MD2(UAP, UAP-> UPDN_ARG.BASIC_OP, MNNIP, MNNIP,
            OMNIP, PTLTP, NO_MTO1VNODE, NO_MTO1NNODE,
            NO_1TOMNODE, IDARR_CTR, CP, CT_NO);
END;

SELECT(UAP-> UPDN_ARG.BASIC_OP);
   WHEN (‘C’, ‘M’)

DO;
PUT SKIP LIST('PRINTING CONTENTS OF TWOLEV_TREE');
PUT SKIP LIST('TLT.BASIC_OP', PTLTP->TWOLEV_TREE.BASIC_OP);
PUT SKIP LIST('TLT.ROOT_FILEID', PTLTP->TWOLEV_TREE.ROOT_FILEID);
DO PR_CTR= 1 TO PTLTP->TWOLEV_TREE.NO_OF_FIELD;
PUT SKIP LIST('ENTRY', PR_CTR,' :', 'FIELDID, DATA');
PUT SKIP LIST(PTLTP->TWOLEV_TREE.IDARR(PR_CTR).FIELDID,
PTLTP->TWOLEV_TREE.ID_ARR(PR_CTR).DATA);
END;
IF NO_MTO1VNODE= 0 THEN DO;
PUT SKIP LIST('PRINTING CONTENTS OF MVNI');
DO PR_CTR= 1 TO MVNIP->MTO1VNODE_INFO.NO_MTO1VNODE;
PUT SKIP LIST('ENTRY', PR_CTR,' :', 'NODE_NO, FIELDID, DATA');
PUT SKIP LIST(MVNIP->MTO1VNODE_INFO.INFOARR(PR_CTR).NODE_NO,
MVNIP->MTO1VNODE_INFO.INFO_ARR(PR_CTR).FIELDID,
MVNIP->MTO1VNODE_INFO.INFO_ARR(PR_CTR).DATA);
END;
END;
IF NO_MTO1NNODE= 0 THEN DO;
PUT SKIP LIST('PRINTING CONTENTS OF MNNI');
DO PR_CTR= 1 TO MNNIP->MTO1NNODE_INFO.NO_MTO1NNODE;
PUT SKIP LIST('ENTRY', PR_CTR,' :', 'INITIAL_NODE_NO',
MVNIP->MTO1NNODE_INFO.INFO_ARR(PR_CTR).
INITIAL.NODE.NO,
MVNIP->MTO1NNODE_INFO.INFO_ARR(PR_CTR).FIELDID);
END;
END;
IF NO_1TOMNODE= 0 THEN DO;
PUT SKIP LIST('PRINTING CONTENTS OF ONI');
DO PR_CTR= 1 TO OMNIP->OTOMNODE_INFO.NO_1TOMNODE;
PUT SKIP LIST('ENTRY', PR_CTR,' :', 'NODE_NO');
PUT SKIP LIST(OMNIP->OTOMNODE_INFO.INFO_ARR(PR_CTR));
END;
END;
WHEN ('D') DO;
PUT SKIP LIST('PRINTING CONTENTS OF TWOLEV_TREE');
PUT SKIP LIST('TLT.BASIC_OP', PTLTP->TWOLEV_TREE.BASIC_OP);
PUT SKIP LIST('TLT.ROOT_FILEID', PTLTP->TWOLEV_TREE.ROOT_FILEID);
WHEN('D')
IF NO_MTO1INNODE^= O
THEN
CALL CR19(UAP, MNNIP, PTLTP, ERR_NODE, CP, FP, CT_NO);
WHEN('D') CALL DE19(UAP, PTLTP, ERR_NODE, CP, FP, CT_NO);
WHEN('M') CALL MD19(UAP, MNNIP, PTLTP, NO_MTO1INNODE, IDARR_CTR, ERR_NODE, CP, FP, CT_NO);
END;

SELECT(UAP-> UPDN_ARG.BASIC_OP);
WHEN ('C')
DO;
IF ERR_NODE= O THEN DO;
PUT SKIP LIST('PRINTING CONTENTS OF TWOLEV_TREE');
DO PR_CTR= 1 TO PTLTP-> TWOLEV_TREE.NO_OF_FIELD;
PUT SKIP LIST('ENTRY', PR_CTR, ':', 'FIELDID AND DATA');
PUT SKIP LIST(PTLTP-> TWOLEV_TREE.ID_ARR(PR_CTR).FIELDID, PTLTP-> TWOLEV_TREE.ID_ARR(PR_CTR).DATA);
END;
IF NO_MTO1INNODE^= O THEN DO;
PUT SKIP LIST('PRINTING CONTENTS OF MNNI');
DO PR_CTR= 1 TO MNNIP-> MTO1INNODE_INFO.NO_MTO1INNODE;
PUT SKIP LIST('ENTRY', PR_CTR, ':', 'INITIAL_NODE_NO, FINAL_NODE_NO, FIELDID');
PUT SKIP LIST(MNNIP-> MTO1INNODE_INFO.INFO_ARR( PR_CTR).
INITIAL_NODE_NO,
MNNIP-> MTO1INNODE_INFO.INFO_ARR( PR_CTR).
FINAL_NODE_NO,
MNNIP-> MTO1INNODE_INFO.INFO_ARR( PR_CTR).FIELDID);
END;
END;
END;
END;
END;
END;

/* UP TO CR19 */

WHEN ('D')
DO;
IF ERR_NODE= O THEN DO;
PUT SKIP LIST('PRINTING CONTENTS OF TWOLEV_TREE');
PUT SKIP LIST('RECORDID OF ROOT NODE');
PUT SKIP LIST(PTLTP-> TWOLEV_TREE.RECORDID);
END;
END;
END;

/* UP TO DE19 */
FILE: CONTROL PLIOPT A . VM/SP CONVERSATIONAL MONITOR SYSTEM PAGE 005

WHEN('M')
DO;
IF ERR_NODE= 0
THEN DO;
PUT SKIP LIST('PRINTING CONTENTS OF TWOLEV_TREE');
PUT SKIP LIST('RECORDID OF ROOT_NODE');
DO PR_CTR= 1 TO PTLTP-> TWOLEV_TREE.NO_OF_FIELD;
   PUT SKIP LIST('ENTRY', PR_CTR, ':', 'FIELDID AND DATA');
   PUT SKIP LIST(PTLTP-> TWOLEV_TREE.ID_ARR(PR_CTR).FIELDID);
END;
IF NO_MTOINNODE^= 0
THEN DO;
PUT SKIP LIST('PRINTING CONTENTS OF MNNI');
DO PR_CTR= 1 TO MNNIP-> MTOINNODE_INFO.NO_MTOINNODE;
   PUT SKIP LIST('ENTRY', PR_CTR, ':', 'INITIALNODE_NO, FINALNODE_NO, FIELDID');
   PUT SKIP LIST(MNNIP-> MTOINNODE_INFO.INFO_ARR(PR_CTR).
      INITIAL_NODE_NO,
      MNNIP-> MTOINNODE_INFO.INFO_ARR(PR_CTR).
      FINAL_NODE_NO,
      MNNIP-> MTOINNODE_INFO.INFO_ARR(PR_CTR).
      FIELDID);
END;
END;
END;
END;
END;
END;
END:

/* UP TO MD19 */

ALLOCATE UPDN_RTN SET(URP);
URP-> UPDN_RTN.RTNCODE= 0;
URP-> UPDN_RTN.ERROR_NODE= 0;
IF ERR_NODE^= 0
THEN DO;
   URP-> UPDN_RTN.ERROR_NODE= ERR_NODE;
   PUT SKIP LIST('NODE THAT CAUSED BREAKDOWN IN P-TREE',
      URP-> UPDN_RTN.ERROR_NODE);
RETURN(URP);
END;

SELECT(UAP-> UPDN_ARG.BASIC_OP);
WHEN ('C', 'M')
FILE: CONTROL PlioT A    VM/SP CONVERSATIONAL MONITOR SYSTEM    PAGE 006

IF NO_1TOMNODE^= 0
THEN
DO:
/* PTLTP-> TOWLEV_TREE.RECORDID= 4*//* TO BE MODIFIED IN CONTROL */
PA_CTR= OMNIP-> OTOMNODE_INFO.NO_1TOMNODE;
ALLOCATE PTR_ARR SET(PAP);
PAP-> PTR_ARR.ARR(*)= NULL();
SELECT(UAP-> UPDN_ARG.BASIC_OP);
WHEN ('C')
    CALL CR20(UAP, OMNIP, PTLTP, ERR_NODE, PAP, CP, FP, CT_NO);
WHEN ('M')
    CALL MD20(UAP, OMNIP, PTLTP, ERR_NODE, PAP, CP, FP, CT_NO);
END;
END:
WHEN ('D')
GOTO SUCCESSFUL_UPDATE;
END;

SELECT(UAP-> UPDN_ARG.BASIC_OP);
    WHEN('C', 'M')
DO:
IF ERR_NODE= 0
THEN
    IF NO_1TOMNODE^= 0
THEN
        GOTO SUCCESSFUL_UPDATE;
END;

SELECT(UAP-> UPDN_ARG.BASIC_OP);
    WHEN('C', 'M')
DO:
IF ERR_NODE^= 0
THEN
    URP-> UPONRTN.ERROR_NODENO= ERR_NODE;
    PUT SKIP LIST('NODE THAT CAUSED BREAKDOWN IN S-TREE', URP-> UPONRTN.ERROR_NODENO);
    RETURN(URP);
END;
IF PTLTP-> TOWLEV_TREE.NO_OF_FIELD= 1
THEN IF PTLTP-> TWOLEV_TREE.BASIC_OP= 'M'
& PTLTP-> TWOLEV_TREE.ID_ARR(1).FIELDID= 0
THEN DO:
  PUT SKIP LIST('NO PRIMARY TREE PROCESSED');
  GOTO PROCESS_ITOM;
END;
ELSE DO:
  CALL TPROC(PTLTP);
  PUT SKIP LIST('PRIMARY TREE PROCESSED');
END;
ELSE DO:
  CALL TPROC(PTLTP);
  PUT SKIP LIST('PRIMARY TREE PROCESSED');
END;

PROCESS_ITOM:
IF PTLTP-> TWOLEV_TREE.B_OP_RTNCODE= 0
THEN DO:
  URP-> UPDN_RTN.RTNCODE= PTLTP-> TWOLEV_TREE.
  B_OP_RTNCODE;
  PUT SKIP LIST('OPERATION FOR MTOI V AND N NODES');
  PUT SKIP LIST('ABORTED, RETURN CODE IS',
    URP-> UPDN_RTN.RTNCODE);
  RETURN(URP);
END;

PUT SKIP LIST('OPERATION FOR MTOI V AND N NODES SUCCESSFUL');
PUT SKIP LIST('BASIC_OP RTNCODE AND ERROR NODENO');
PUT SKIP LIST(URP->UPDN_RTN.RTNCODE, URP->UPDN_RTN.ERROR_NODENO);

IF NO_1TOMNODE= 0
THEN
  DO:
    IF UAP-> UPDN_ARG.BASIC_OP= 'C'
    THEN DO
      PR_CTR= 1 TO OMNIP-> OTOMNODE_INFO.NO_1TOMNODE;
      PAP-> PTR_ARR.ARR(PR_CTR)-> TWOLEV_TREE.ID_ARR(1).DATA
      = PTLTP-> TWOLEV_TREE.RECORDID;
      PUT SKIP LIST('NEW RECORD CREATED.');
      PUT SKIP LIST('ENTRY', PR_CTR, ':', 'FIELDID, DATA');
      PUT SKIP LIST(PAP-> PTR_ARR.ARR(PR_CTR)->
        TWOLEV_TREE.ID_ARR(i).FIELDID,
      PAP-> PTR_ARR.ARR(PR_CTR)->
        TWOLEV_TREE.ID_ARR(1).DATA);
    END;
    DO PR_CTR= 1 TO OMNIP-> OTOMNODE_INFO.NO_1TOMNODE;
    CALL TPROC(PAP-> PTR_ARR.ARR(PR_CTR));
  IF PAP-> PTR_ARR.ARR(PR_CTR)-> TWOLEV_TREE.B_OP_RTNCODE
    ^= 0
  THEN DO:
    URP-> UPDN_RTN.RTNCODE
    = PAP-> PTR_ARR.ARR(PR_CTR)->
      TWOLEV_TREE.B_OP_RTNCODE;
    PUT SKIP LIST('OPERATION NUMBER', PR_CTR,
      'ABORTED, RETURN CODE IS',
    URP-> UPDN_RTN.RTNCODE);
  RETURN(URP);
END:
SUCCESSFUL UPDATE:

PUT SKIP LIST('OPERATION COMPLETELY SUCCESSFUL');

PUT SKIP LIST('BASIC_OP RETNCODE, ERROR_NODENO');

PUT SKIP LIST(URP->UPDN_RTN.RTNCODE,
               URP->UPDN_RTN.ERROR_NODENO);

RETURN(URP);

END CONTROL;
*PROCESS NAME('USER'), INCLUDE, F(I);
USER: PROC OPTIONS(MAIN);
DCL NULL BUILTIN;
DCL CONTROL ENTRY EXTERNAL RETURNS(PTR);
DCL RTN_PTR PTR INIT(NULL());
DCL UAP PTR INIT(NULL());

   RTN_PTR= CONTROL(UAP);
END USER;
*PROCESS NAME('ILN'). INCLUDE. F(I);
ILN: PROC (UAP, NODE_NO) RETURNS (BIT);
/* ILN= IS_LEVEL2NODE */

%INCLUDE UA;
DCL UAP PTR,
    NODE_NO FIXED;

IF UAP-> UPDN_ARG.NODE_DESCRIP (NODE_NO).PARENT= 0
    THEN RETURN ('1' B);
ELSE RETURN ('0' B);

.END ILN;
*PROCESS NAME('CLN'), INCLUDE, F(I);
CLN: PROC (UAP) RETURNS (FIXED);

/* CLN= COUNT_LEVEL2NODE */

%INCLUDE UA;

DCL (TRANSLATE, NULL) BUILTIN;
DCL P PTR;
DCL UAP PTR,
    (NO_LEVEL2NODE) FIXED INIT(0);

DO NODE_NO = 1 TO UAP->UPDN_ARG.N;
    IF UAP->UPDN_ARG.NODE_DESCRIP(NODE_NO).PARENT = 0
    THEN NO_LEVEL2NODE = NO_LEVEL2NODE + 1;
END;
RETURN (NO_LEVEL2NODE);

END CLN;
*PROCESS NAME('CMN'), INCLUDE F(I);  
/* CMN= COUNT_MTO1NODE */  

CMN: PROC (UAP, BASIC_OP, NO_MTO1VNODE, NO_MTO1NNODE,  
CP, CT_NO);  
/* CMN= COUNT_MTO1NODE */

/* INCLUDE UA; */
DCL P PTR;
DCL UAP PTR,
BASIC_OP CHAR(1),
(NO_MTO1VNODE, NO_MTO1NNODE) FIXED,
CP PTR,
CT_NO FIXED;
DCL ILN ENTRY EXTERNAL HANDS(BIT),
QCNTYP ENTRY EXTERNAL HANDS(CHAR(1)),
QCEMB ENTRY EXTERNAL HANDS(BIT);
DCL NODE_NO FIXED INIT(O),
BSETID FIXED BIN(15) INIT(O);
NO_MTO1VNODE = 0;
NO_MTO1NNODE = 0;
DO NODE_NO = 1 TO UAP-> UPDN_ARG.N;
IF ILN(UAP, NODE_NO)
THEN IF BASIC_OP = 'M'
& UAP-> UPDN_ARG.NODE_DESCRIP(NODE_NO).OP = 'I'
THEN GOTO EOL;
ELSE DO;
BSETID = UAP-> UPDN_ARG.NODE_DESCRIP
(NODE_NO).BSETID;
IF QCNTYP(CP, CT_NO, BSETID) = 'V'
THEN NO_MTO1VNODE = NO_MTO1VNODE + 1;
ELSE IF QCEMB(CP, CT_NO, BSETID)
THEN NO_MTO1NNODE =
NO_MTO1NNODE + 1;
END;
EOL:
END;
END CMN;
*PROCESS NAME('GTRCID'), INCLUDE, F(I);
GTRCID: PROC(UAP, INITIAL_NODE_NO, FINAL_NODE_NO, RECORDID,
        CP, FP, CT_NO)
        RETURNS(FIXED);
/*'GTRCID= GET_RECORDID */

%INCLUDE UA;
%INCLUDE SFL;

DCL P PTR;
DCL QCFLID ENTRY EXTERNAL RETURNS(FIXED BIN(31));
DCL QCFDID ENTRY EXTERNAL RETURNS(FIXED BIN(31));

DCL UAP PTR,
    (INITIAL_NODE_NO, FINAL_NODE_NO) FIXED,
    RECORDID FIXED BIN(31),
    (CP, FP) PTR,
    CT_NO FIXED;

DCL FILEID FIXED BIN(31) INIT(0),
    BSETID FIXED BIN(15) INIT(0),
    FIELDID FIXED BIN(31) INIT(0);

FILEID= QCFLID(CP, CT_NO);
BSETID= UAP-> UPDN_ARG.NODE_DESCRIP(INITIAL_NODE_NO).BSETID;
FIELDID= QCFDID(CP, CT_NO, BSETID);
RECORDID= FP-> FILE(FILEID).RECORD(2).FIELD(FIELDID);
IF RECORDID=O
    THEN RETURN(1);
ELSE RETURN(0);

END GTRCID;
*PROCESS NAME('QCEMB'), INCLUDE, F(I);
QCEMB: PROC(CP, CT_NO, BSETID) RETURNS(BIT);
/* QCEMB= QUERY EMBEDDED OR NON-EMBEDDED */

%INCLUDE SPCT;

DCL CP PTR,
   CT_NO FIXED,
   BSETID FIXED BIN(15);
DCL BD_CTR FIXED INIT(0);

DO BD_CTR= 1 TO CP-> PCAT(CT_NO).N;
   IF CP-> PCAT(CT_NO).BSET_DESCRIP(BD_CTR).BSETID= BSETID
      THEN IF CP-> PCAT(CT_NO).BSET_DESCRIP(BD_CTR).IMP.ITYPE= 'E'
      THEN RETURN('1'B);
      ELSE RETURN('0'B);
   END;
   RETURN('0'B);
END QCEMB:
*PROCESS NAME('QCFDID'), INCLUDE, F(I):
QCFDID: PROC(CP, CT_NO, BSETID) RETURNS(FIXED BIN(31));

/*/ QCFDID= QUERY_FIELDID */

%INCLUDE SPCT;

DCL P PTR;
DCL CP PTR,
   CT_NO FIXED,
   BSETID FIXED BIN(15);
DCL BD_CTR FIXED INIT(0);

DO BD_CTR= 1 TO CP-> PCAT(CT_NO).N;
   IF CP-> PCAT(CT_NO).BSET_DESCRIP(BD_CTR).BSETID = BSETID
      THEN RETURN(CP-> PCAT(CT_NO).BSET_DESCRIP(BD_CTR).
                  IMP.FIELDID);
END;
END QCFDID;
*PROCESS NAME('QCFLDS'), INCLUDE. F(I);
QCFLDS: PROC(CP, CT_NO, BSETID) RETURNS(FIXED BIN(31));

/* QCFLDS= QUERY_FILEID_FOR_SECONDARY_TWOLEV_TREE */

DCL P PTR;
DCL CP PTR,
  CT_NO FIXED,
  BSETID FIXED BIN(15);
DCL BD_CTR FIXED INIT(0);

DO BD_CTR= 1 TO CP-> PCAT(CT_NO).N;
  IF CP-> PCAT(CT_NO).BSET_DESCRIP(BD_CTR).BSETID= BSETID
     THEN RETURN(CP-> PCAT(CT_NO).BSET_DESCRIP(BD_CTR).
                      IMP.FILEID);
END;
RETURN(O);
END QCFLDS;
*PROCESS NAME('QCFLID'), INCLUDE, F(I);
QCFLID: PROC(CP, CT_NO) RETURNS(FIXED BIN(31));

/* QCFLID= QUERY_FILEID */

DCL P PTR;
DCL CP PTR,
CT_NO FIXED;
RETURN(CP-> PCAT(CT_NO).FILEID);

END QCFLID;
*PROCESS NAME('OCNTYP'), INCLUDE F(I);
QCNTYP: PROC(CP, CT_NO, BSETID) RETURNS(CHAR(1));

/* QCNTYP= QUERY_NODE_TYPE */

%INCLUDE SPCT;

DCL P PTR;
DCL CP PTR,
   CT_NO FIXED,
   BSETID FIXED BIN(15);
DCL BD_CTR FIXED INIT(0);

DO BD_CTR = 1 TO CP->PCAT(CT_NO).N;
   IF CP-> PCAT(CT_NO).BSET_DESCRIP(BD_CTR).BSETID= BSETID
      THEN RETURN(CP-> PCAT(CT_NO).BSET_DESCRIP(BD_CTR).TYPE);
END:
RETURN(' ');

END QCNTYP:
*PROCESS NAME('SBDUA'). INCLUDE, F(I);  
SBDUA: PROC RETURNS (PTR);  
/
SBDUA* BUILD_UPDN_ARG */  
/* SBDUA = 
/* INCLUDE UA; */  
DCL TRANSLATE BUILTIN;  
DCL NULL BUILTIN;  
DCL P PTR;  
DCL UA_CTR FIXED INIT(0), UAP PTR INIT(NULL()),  
ND_CTR FIXED INIT(0),  
(LOW_BASIC_OP, LOW_OP) CHAR(1);  
PUT SKIP(2) LIST('TOTAL NUMBER OF NON ROOT NODES, FIXED');  
GET LIST(UA_CTR);  
ALLOCATE UPDN_ARG SET (UAP);  
UAP-> UPDN_ARG.BASIC_OP = ' ';  
UAP-> UPDN_ARG.ROOT_PSETID = 0;  
UAP-> UPDN_ARG.NODE_DESCRIP(*) .BSETID = 0;  
UAP-> UPDN_ARG.NODE_DESCRIP(*) .PARENT = 0;  
UAP-> UPDN_ARG.NODE_DESCRIP(*) .OP = ' ';  
UAP-> UPDN_ARG.NODE_DESCRIP(*) .DATA = ' ';  
GET LIST(LOW_BASIC_OP, UAP-> UPDN_ARG.ROOT_PSETID);  
UAP-> UPDN_ARG.BASIC_OP = TRANSLATE(LOW_BASIC_OP, 'CDM', 'cdm');  
DO ND_CTR = 1 TO UA_CTR;  
GET LIST(UAP-> UPDN_ARG.NODE_DESCRIP(ND_CTR) .BSETID,  
UAP-> UPDN_ARG.NODE_DESCRIP(ND_CTR) .PARENT,  
LOW_OP,  
UAP-> UPDN_ARG.NODE_DESCRIP(ND_CTR) .DATA);  
UAP-> UPDN_ARG.NODE_DESCRIP(ND_CTR) .OP = TRANSLATE(LOW_OP, 'IPRA', 'ipra');  
END;  
RETURN (UAP);  
END SBDUA;
FILE: SFLAPC PLIOPT A VM/SP CONVERSATIONAL MONITOR SYSTEM PAGE 001

*PROCESS NAME('SFLAPC'), INCLUDE, F(I);
SFLAPC: PROC(FP, CP);

/* SFLAPC= FILES AND PSET_CATALOGUE */

%INCLUDE SFL;
%INCLUDE SPCT;

DCL P PTR;
DCL (FP, CP) PTR;

ALLOCATE FILE SET (FP);
ALLOCATE PCAT SET (CP);

CP-> PCAT(*).PSETID= 0;
CP-> PCAT(*).FILEID= 0;
CP-> PCAT(*).BSET_DESCRIP(*).BSETID= 0;
CP-> PCAT(*).BSET_DESCRIP(*).TYPE= 'I';
CP-> PCAT(*).BSET_DESCRIP(*).FUNC= 'I';
CP-> PCAT(*).BSET_DESCRIP(*).IMP.ITYPE= 'E';
CP-> PCAT(*).BSET_DESCRIP(*).IMP.FIELDID= 0;
CP-> PCAT(*).BSET_DESCRIP(*).IMP.FILEID= 0;

CP-> PCAT(3).PSETID= 10000;
CP-> PCAT(3).FILEID= 10;
CP-> PCAT(3).N= 5;
CP-> PCAT(3).BSET_DESCRIP(1).BSETID= 304;
CP-> PCAT(3).BSET_DESCRIP(1).TYPE= 'V';
CP-> PCAT(3).BSET_DESCRIP(1).FUNC= 'I';
CP-> PCAT(3).BSET_DESCRIP(1).IMP.ITYPE= 'E';
CP-> PCAT(3).BSET_DESCRIP(1).IMP.FIELDID= 1;
CP-> PCAT(3).BSET_DESCRIP(2).BSETID= 303;
CP-> PCAT(3).BSET_DESCRIP(2).TYPE= 'V';
CP-> PCAT(3).BSET_DESCRIP(2).FUNC= 'I';
CP-> PCAT(3).BSET_DESCRIP(2).IMP.ITYPE= 'E';
CP-> PCAT(3).BSET_DESCRIP(2).IMP.FIELDID= 2;
CP-> PCAT(3).BSET_DESCRIP(3).BSETID= 302;
CP-> PCAT(3).BSET_DESCRIP(3).TYPE= 'N';
CP-> PCAT(3).BSET_DESCRIP(3).FUNC= 'M';
CP-> PCAT(3).BSET_DESCRIP(3).IMP.ITYPE= 'E';
CP-> PCAT(3).BSET_DESCRIP(3).IMP.FIELDID= 3;
CP-> PCAT(3).BSET_DESCRIP(4).BSETID= 302;
CP-> PCAT(3).BSET_DESCRIP(4).TYPE= 'N';
CP-> PCAT(3).BSET_DESCRIP(4).FUNC= 'M';
CP-> PCAT(3).BSET_DESCRIP(4).IMP.ITYPE= 'E';
CP-> PCAT(3).BSET_DESCRIP(4).IMP.FIELDID= 3;
CP-> PCAT(3).BSET_DESCRIP(5).BSETID= 401;
CP-> PCAT(3).BSET_DESCRIP(5).TYPE= 'N';
CP-> PCAT(3).BSET_DESCRIP(5).FUNC= 'M';
CP-> PCAT(3).BSET_DESCRIP(5).IMP.ITYPE= 'E';
CP-> PCAT(3).BSET_DESCRIP(5).IMP.FIELDID= 401;
CP-> PCAT(3).BSET_DESCRIP(5).IMP.FILEID= 8;
CP-> PCAT(3).BSET_DESCRIP(6).BSETID= 401;
CP-> PCAT(3).BSET_DESCRIP(6).TYPE= 'N';
CP-> PCAT(3).BSET_DESCRIP(6).FUNC= 'M';
CP-> PCAT(3).BSET_DESCRIP(6).IMP.ITYPE= 'E';
CP-> PCAT(3).BSET_DESCRIP(5).IMP.ITYPE= 'N';
CP-> PCAT(3).BSET_DESCRIP(5).IMP.FIELDID= 2;
CP-> PCAT(3).BSET_DESCRIP(5).IMP.FIELDID= 7;
CP-> PCAT(2).PSETID= 20000;
CP-> PCAT(2).FILEID= 9;
CP-> PCAT(2).N= 5;
CP-> PCAT(2).SETID= 201;
CP-> PCAT(2).SETID= 202;
CP-> PCAT(2).SETID= 203;
CP-> PCAT(1).PSETID= 30000;
CP-> PCAT(1).FILEID= 8;
CP-> PCAT(1).N= 5;
CP-> PCAT(1).SETID= 101;
CP-> PCAT(1).SETID= 102;
CP-> PCAT(1).SETID= 103;
CP-> PCAT(1).SETID= 104;
CP-> PCAT(1).SETID= 105;
VM/SP CONVERSATIONAL MONITOR SYSTEM

CP-> PCAT(4).PSETID= 25000;
CP-> PCAT(4).FILEID= 7;
CP-> PCAT(4).N= 5;
CP-> PCAT(4).BSET_DESCRIP(1).BSETID= 401;
CP-> PCAT(4).BSET_DESCRIP(1).TYPE='N';
CP-> PCAT(4).BSET_DESCRIP(1).FUNC='M';
CP-> PCAT(4).BSET_DESCRIP(1).IMP.ITYPE='E';
CP-> PCAT(4).BSET_DESCRIP(1).IMP.FIELDID= 2;
CP-> PCAT(4).BSET_DESCRIP(2).BSETID= 402;
CP-> PCAT(4).BSET_DESCRIP(2).TYPE='V';
CP-> PCAT(4).BSET_DESCRIP(2).FUNC='i';
CP-> PCAT(4).BSET_DESCRIP(2).IMP.ITYPE='E';
CP-> PCAT(4).BSET_DESCRIP(2).IMP.FIELDID= 1;
CP-> PCAT(4).BSET_DESCRIP(3).BSETID= 403;
CP-> PCAT(4).BSET_DESCRIP(3).TYPE='N';
CP-> PCAT(4).BSET_DESCRIP(3).FUNC='M';
CP-> PCAT(4).BSET_DESCRIP(3).IMP.ITYPE='E';
CP-> PCAT(4).BSET_DESCRIP(3).IMP.FIELDID= 3;
CP-> PCAT(5).PSETID= 15000;
CP-> PCAT(5).FILEID= 6;
CP-> PCAT(5).N= 5;
CP-> PCAT(5).BSET_DESCRIP(1).BSETID= 403;
CP-> PCAT(5).BSET_DESCRIP(1).TYPE='N';
CP-> PCAT(5).BSET_DESCRIP(1).FUNC='O';
CP-> PCAT(5).BSET_DESCRIP(1).IMP.ITYPE='N';
CP-> PCAT(5).BSET_DESCRIP(1).IMP.FIELDID= 3;
CP-> PCAT(5).BSET_DESCRIP(1).IMP.FILEID= 7;
CP-> PCAT(5).BSET_DESCRIP(2).BSETID= 501;
CP-> PCAT(5).BSET_DESCRIP(2).TYPE='V';
CP-> PCAT(5).BSET_DESCRIP(2).FUNC='i';
CP-> PCAT(5).BSET_DESCRIP(2).IMP.ITYPE='E';
CP-> PCAT(5).BSET_DESCRIP(2).IMP.FIELDID= 1;
CP-> PCAT(5).BSET_DESCRIP(3).BSETID= 502;
CP-> PCAT(5).BSET_DESCRIP(3).TYPE='V';
CP-> PCAT(5).BSET_DESCRIP(3).FUNC='O';
CP-> PCAT(5).BSET_DESCRIP(3).IMP.ITYPE='E';
CP-> PCAT(5).BSET_DESCRIP(3).IMP.FIELDID= 3;

FP-> FILE(*).RECORD(*).FIELD(*)= 0;
FP-> FILE(10).RECORD(2).FIELD(1)= 1000;
FP-> FILE(10).RECORD(2).FIELD(2)= 21;
FP-> FILE(10).RECORD(2).FIELD(3)= 2;
FP-> FILE(9).RECORD(2).FIELD(1)= 15;
FP-> FILE(9).RECORD(2).FIELD(2)= 300;
FP-> FILE(9).RECORD(2).FIELD(3)= 30;
FP-> FILE(8).RECORD(2).FIELD(1)= 2;
FP-> FILE(8).RECORD(2).FIELD(2)= 1;
FP-> FILE(8).RECORD(2).FIELD(3)= 3;
FP-> FILE(8).RECORD(2).FIELD(4)= 2000;
FP-> FILE(7).RECORD(2).FIELD(1)= 37;
FP-> FILE(7).RECORD(2).FIELD(2)= 2;
FP-> FILE(7).RECORD(2).FIELD(3)= 2;
FP-> FILE(6).RECORD(2).FIELD(1)= 1369;
FP-> FILE(6).RECORD(2).FIELD(3)= 41;

END SFLAPC;
Appendix 2- Data Structures

The following PL/1 data structures are included in this appendix in order.

UPDN_ARG
UPDN_RTN
TWOLEV_TREE
MT01VNODE_INFO
MT01NNODE_INFO
OTOMNODE_INFO
PTR_ARR
PCAT
FILE
DCL 1 UPDN_ARG BASED (P),
 2 BASIC_OP CHAR(1),
 2 ROOT_PSETID FIXED BIN(15),
 2 N FIXED,
 2 NODE_DESCRIP(UA_CTR REFER (UPDN_ARG.N)),
 3 BSÈTID FIXED BIN(15),
 3 PARENT FIXED,
 3 OP CHAR(1),
 3 DATA CHAR(40) VAR;
DCL 1 UPDN_RTN BASED(P),
2 RTN_CODE FIXED,
2 ERROR_NODENO FIXED;

UR 00010
UR 00020
UR 00030
DCL 1 TWOLEV_TREE BASED (P),
  2 BASIC_OP CHAR (1),
  2 B_OPRTNCODE FIXED,
  2 ROOT_FILEID FIXED BIN (31),
  2 RECORDID FIXED BIN (31),
  2 NO_OF_FIELD FIXED,
  2 ID_ARR (NO_FDUPD REFER (TWOLEV_TREE:NO_OF_FIELD)),
  3 FIELDID FIXED BIN (31),
  3 DATA CHAR (40) VAR;
DCL 1 MT01VNODE_INFO BASED (P),
2 NO_MT01VNODE FIXED,
2 INFO_ARR (VIA_CTR REFER (MT01VNODE_INFO.NO_MT01VNODE)),
3 NODE NO FIXED,
3 FIELDID FIXED BIN(31),
3 DATA CHAR (40) VAR;

MVNO0010
MVNO0020
MVNO0030
MVNO0040
MVNO0050
MVNO0060
DCL 1 MTO1NNODE_INFO BASED (P),
   2 NO_MTO1NNODE FIXED,
   2 INFO_ARR (NIA_CTR REFERENCE (MTO1NNODE_INFO.NO_MTO1NNODE)),
   3 INITIAL_NODE_NO FIXED,
   3 FINAL_NODE_NO FIXED,
   3 FIELDID FIXED BIN(31):
DCL 1 OTOMNODE_INFO BASED (P),
  2 NO_1TOMNODE FIXED,
  2 INFO_ARR (ONIA_CTR REFER (OTOMNODE_INFO.NO_1TOMNODE)) FIXED;

---

PAGE 001
DCL 1 PTR_ARR BASED (P).
   2 NO_PTR FIXED.
   2 ARR(PA_CTR REFER(PTR_ARR.NO_PTR)) PTR;

PA 00010
PA 00020
PA 00030
DCL 1 PCAT(5) BASED (P).
  2 PSETID FIXED BIN(15),
  2 FILEID FIXED BIN(31),
  2 N FIXED,
  2 BSET_DESCRIP (5),
  3 BSETID FIXED BIN(15),
  3 TYPE CHAR(1),
  3 FUNC CHAR(1),
  3 IMP,
  4 ITEP CHAR(1),
  4 FIELDID FIXED BIN(31),
  4 FILEID FIXED BIN(31);
DCL 1 FILE (10) BASED (P),
  2 RECORD (2),
  3 FIELD (4) FIXED;

SFLO0010
SFLO0020
SFLO0030
The following are a few testings with all the three types of trees—create, delete and modify. The complete testing can be accomplished by running through the following exec files:

- ex31 to ex40 for testing create
- ex51 to ex52 for testing delete
- ex71 to ex92 for testing modify
EXECUTION BEGINS...

TOTAL NUMBER OF NON ROOT NODES, FIXED

NUMBER OF LEVEL 2 NODES IS 2
NUMBER OF MT01 VALUE NODES IS 2
NUMBER OF MT01 ENTITY NODES IS 0
PRINTING CONTENTS OF TWOLEV_TREE

TLT.BASIC_OP C
TLT.ROOT_FILEID
ENTRY 2  300
ENTRY 1  15
ENTRY 3 NULL

PRINTING CONTENTS OF MVNI
ENTRY 1  1  15
ENTRY 2  2  300

PRINTING CONTENTS OF TWOLEV_TREE
ENTRY 1  300
ENTRY 1  15
ENTRY 3 NULL

PRIMARY TREE PROCESSED
OPERATION FOR MT01 V AND N NODES SUCCESSFUL

BASIC_OP RTNCODE AND ERROR_NODENO
O 0
OPERATION COMPLETELY SUCCESSFUL
BASIC_OP RTNCODE, ERROR_NODENO
O 0

EXECUTION BEGINS...

TOTAL NUMBER OF NON ROOT NODES, FIXED

NUMBER OF LEVEL 2 NODES IS 2
NUMBER OF MT01 VALUE NODES IS 1
NUMBER OF MT01 ENTITY NODES IS 1
PRINTING CONTENTS OF TWOLEV_TREE

TLT.BASIC_OP C
TLT.ROOT_FILEID
ENTRY 2  NULL
ENTRY 1 NULL 2
ENTRY 3 3
ENTRY 4 NULL
PRINTING CONTENTS OF MNNI ENTRY 1
ENTRY 3 3
PRINTING CONTENTS OF MNNI ENTRY 1
ENTRY 1
PRINTING CONTENTS OF TWOLEV_TREE ENTRY 1
ENTRY 2 NULL
ENTRY 1
ENTRY 3 3
ENTRY 4 NULL
PRINTING CONTENTS OF MNNI ENTRY 1
ENTRY 1

PRIMARY TREE PROCESSED
OPERATION FOR MTO1 V AND N NODES SUCCESSFUL
BASIC_OP RNICODE AND ERROR_NODENO
0 0
OPERATION COMPLETELY SUCCESSFUL
BASIC_OP RNICODE, ERROR_NODENO
0 0
R; T=0.68/1.38 12:08:06
ex51
R; T=0.01/0.01 12:08:10
EXECUTION BEGINS...

TOTAL NUMBER OF NON ROOT NODES, FIXED

NUMBER OF LEVEL 2 NODES IS 3
NUMBER OF MTO1 VALUE NODES IS 2
NUMBER OF MTO1 ENTITY NODES IS 0
PRINTING CONTENTS OF TWOLEV_TREE
TLT.BASIC_OP D
TLT.ROOT_FILEID
PRINTING CONTENTS OF TWOLEV_TREE
RECORDID OF ROOT_NODE 300
OPERATION COMPLETELY SUCCESSFUL
BASIC_OP RNICODE, ERROR_NODENO
0 0
R; T=0.64/1.26 12:08:28
ex51
R; T=0.01/0.01 12:08:36
EXECUTION BEGINS...
TOTAL NUMBER OF NON ROOT NODES, FIXED

NUMBER OF LEVEL 2 NODES IS 3
NUMBER OF MTO1 VALUE NODES IS 3
NUMBER OF MTO1 ENTITY NODES IS 0
PRINTING CONTENTS OF TWOLEV_TREE
TLT.BASIC_OP 0
TLT.ROOT_FILEID 8
PRINTING CONTENTS OF TWOLEV_TREE
RECORDID OF ROOT NODE
ENTRY 3
PRINTING CONTENTS OF TWOLEV_TREE
ENTRY 1 15
ENTRY 2 300
ENTRY 2 300
PRINTING CONTENTS OF MVNI
ENTRY 2 1 1 15
ENTRY 3 2 300
ENTRY 3 2 300
PRINTING CONTENTS OF TWOLEV_TREE
RECORDID OF ROOT NODE
ENTRY 30
ENTRY 1 15
ENTRY 2 300
ENTRY 2 300
PRIMARY TREE PROCESSED
OPERATION FOR MTO1 V AND N NODES SUCCESSFUL
BASIC_OP RETNCODE AND ERROR_NODENO 0 0
OPERATION COMPLETELY SUCCESSFUL
BASIC_OP RETNCODE, ERROR_NODENO 0 0
R; T=0.64/1.29 12:08:51
R; T=0.01/0.02 12:09:09
EXECUTION BEGINS...

TOTAL NUMBER OF NON ROOT NODES, FIXED

NUMBER OF LEVEL 2 NODES IS 3
NUMBER OF MTO1 VALUE NODES IS 2
NUMBER OF MTO1 ENTITY NODES IS 0
PRINTING CONTENTS OF TWOLEV_TREE
TLT.BASIC_OP M
TLT.ROOT_FILEID 9
ENTRY 1 15
ENTRY 2 300
ENTRY 2 300
PRINTING CONTENTS OF MVNI
ENTRY 2 1 1 15
ENTRY 3 2 300
ENTRY 3 2 300
PRINTING CONTENTS OF TWOLEV_TREE
RECORDID OF ROOT NODE
ENTRY 30
ENTRY 1 15
ENTRY 2 300
ENTRY 2 300
PRIMARY TREE PROCESSED
OPERATION FOR MTO1 V AND N NODES SUCCESSFUL
BASIC_OP RETNCODE AND ERROR_NODENO 0 0
OPERATION COMPLETELY SUCCESSFUL
BASIC_OP RETNCODE, ERROR_NODENO 0 0
R; T=0.67/1.39 12:09:27
R; T=0.01/0.02 12:09:09
TOTAL NUMBER OF NON ROOT NODES, FIXED:

NUMBER OF LEVEL 2 NODES IS 3
NUMBER OF MTO1 VALUE NODES IS 0
NUMBER OF MTO1 ENTITY NODES IS 0
PRINTING CONTENTS OF TWELEV_TREE
  TLT.BASIC_OP M
  TLT.ROOT_FILEID ENTRY
  PRINTING CONTENTS OF ONI
  ENTRY 1
  ENTRY 3
  ENTRY 2
  PRINTING CONTENTS OF TWOLEV_TREE
  RECORDID OF ROOT_NODE 2
  ENTRY 0
  PRINTING CONTENTS OF PTR_ARR
  ENTRY M
  RECORDID, NO_FDUPD 21 1
  FIELDID, DATA 2
  ENTRY M
  RECORDID, NO_FDUPD 1000 1
  FIELDID, DATA

NO PRIMARY TREE PROCESSED
OPERATION FOR MTO1 V AND N NODES SUCCESSFUL
  BASIC_OP RTNCODE AND ERROR_NODENO 0 0
OPERATION COMPLETELY SUCCESSFUL
  BASIC_OP RETNCODE, ERROR_NODENO 0 0

R; T=0.69/1.42 12:09:53
cp sp cons stop close