REPORT ON COMPUTER USAGE OF ACCOUNTS UNDER THE IBM/M.I.T. JOINT STUDY AGREEMENT GMIS SOFTWARE DEVELOPMENT

NEEMIS Staff
Systems Programming Group

Energy Laboratory in Association with
the Alfred P. Sloan School of Management

Working Paper No. MIT-EL-76-018WP

July 1976

IBM/M.I.T. JOINT STUDY PROGRESS REPORT - PART I

GMIS SOFTWARE DEVELOPMENT

July 1976

CONTENTS

GMIS Progress Report:

Systems Programming Group

1. Introduction

This report will detail our progress to date on the systems programming and development effort relating to the IBM/MIT Joint Study. This systems effort has primarily focused on the development of the Generalized Management Information System (GMIS) and its many component software facilities.

Appendix A provides a list of each VM/370 account being used in the GMIS systems development effort, and describes how each account is being used.

Our last GMIS Progress Report detailing our software development efforts, dated August 22, 1975, is included with this report as Appendix B. We will begin this progress report with work done subsequent to the August 22, 1975 report.

There will be five major sections to this report:

1) A discussion of progress made in each area of GMIS software development.

2) A description of the computer technologies used in this software development effort.

3) A discussion of the advantages of developing this system in the software, hardware and administrative environment provided by the IBM/MIT Joint Study.

4) Some suggestions for technological improvements that would improve the development process for GMIS, and would improve the use of the system.

5) A statement of what we have learned during our system development efforts that would assist future design and implementation efforts of a system like GMIS.

2. Progress in Software Development

This section describes software development efforts that have taken place since the August 22, 1975 GMIS Progress Report (Appendix A). We will assume that the reader is familiar with the motivation for GMIS, the uses of GMIS and the GMIS architecture. One document which provides this information is "GMIS: An Experimental System for Data Management and Analysis", by John J. Donovan and Henry D.

Jacoby (MIT Energy Laboratory Working Paper MIT-EL-75-011WP, September, 1975).

2.1. SEQUEL

Last summer, Release 2 of SEQUEL from San Jose was tested extensively for performance characteristics. Our findings were relayed to San Jose, which are outlined in sections 3 and 5 of the discussion overview used there last summer (Appendix C). Their overall response was to use what we learned about the use of SEQUEL in the development of System R, but they were not enthusiastic about making changes to SEQUEL Release 2 to improve performance.

Several changes have been made to the SEQUEL, XRAM and multi-segment RAM code over the past year. These changes include:

1) The data type DEC (FIXED DECIMAL) was added to SEQUEL to supplement the existing NUM and CHAR data types.

2) The SEQUEL parser and transaction execution routines were modified to accept the unary + and - operators as prefixes to numeric literals.

3) A LOG facility was added to SEQUEL which, if activated, sends the following data items to the STATPRI file:

- the SEQUEL statement
- the time when the transaction was entered
- the transaction type (TRANTYP)
- a list of all columns and tables referenced in the statement.

4) Software "hooks" were installed in the SEQUEL declaration macros to allow security features to be installed at the Multi User Interface (MUI) level.

5) An EXTERNAL array called LPAD was added to the SEQUEL Transaction Interface (TRANS) to inform UFI programs of the number of blanks added to the left of columns in SEQUEL output rows for formatting purposes.

6) Several SEQUEL and XRAM routines were modified to handle a problem that developed when inversions were made on certain columns containing numeric data. When a column contained a numeric value that was either nonpositive or greater than the greatest value of a RAM ID (2**21-1), the inversion would apparently be successfully created, but would respond erroneously when used. The most

straightforward technique to prevent this from occurring was to install a checking mechanism that would permit inversions to be created and manintained only on columns with data in the range of 1 to 2**21-1. Extensions to XRAM were made by Ray Fessel to implement this mechanism, and SEQUEL was modified to activate this inversion suppression when necessary. The result of these modifications is that the performance benefits of SEQUEL/XRAM inversions of columns cannot be used in columns with data that is out of a relatively narrow numeric range.

7) The SEQUEL error messages were revised to make them easier to understand by users.

8) Various SEQUEL, XRAM and RAM bugs were identified and fixed.

More detailed documentation on these changes is included in the following documents:

1) Log of SEQUEL Modifications (Appendix D)

2) Limitations of SEQUEL Release 3 (Appendix E)

3) Transaction Interface Description - MIT/SEQUEL Release 3 (Appendix F)

4) XRAM Extensions (Appendix G).

2.2. TRANSACT

A version of TRANSACT similar to that described in the last GMIS Progress Report was debugged and has been operational since September, 1975. The following documents detail the TRANSACT syntax and explain the use of each command:

1) TRANSACT Commands (Appendix H)

2) Explanation of TRANSACT Commands (Appendix I).

A number of extensions to TRANSACT have been proposed, including:

1) The implementation of a SET LOG command, to activate the new SEQUEL LOG facility described in the previous section of this report.

2) The TRANSACT RUN command macro processor. This facility, similar to CMS EXEC, was designed and partially implemented last summer. It is described in the document "TRANSACT RUN Command Macro Processor User Guide" (Appendix J).

3) The implementation of a TRANSACT transaction editor. One proposal for this editor is included with this report as Appendix K.

2.3. Multi-User Interface (MUI)

The GMIS Multi - User Interface is the routine that permits the SEQUEL data base machine to communicate with other virtual machines containing applications programs. The MUI is actually a SEQUEL User Friendly Interface (UFI) residing on the GMIS account as the SEQ3 MODULE. Access control features were added to the original MUI, SEQ2, which can be invoked by specifying the option SECURE with the START exec when activating the Transaction Virtual Machine (TVM).

This part of the GMIS system is currently being replaced under the new GMIS inter-machine communications facility design. The new GMIS design is described in section 2.7.

2.4. SEQUEL Database Description and Backup Utilities

Several additional SEQUEL utility programs have been written which permit users to retrieve a description of a SEQUEL database and to create several different types of backup copies of all or part of a database. These utility routines include:

1) DBDESCR - describes SEQUEL tables and prints SEQUEL catalog information on the terminal or line printer (Appendix L).

2) LTDB - the standard GMIS bulk loader (Appendix M).

3) SEQDUMP - the first SEQUEL backup utility. An entire SEQUEL database is dumped to CMS files in bulk loader (LTDB) format (Appendix N).

3) SEQBACK and SEQREST - utilize the CMS DDR command to create and reload a tape copy of the disks containing the database. These routines are faster than SEQDUMP, but are less flexible in their use (Appendix O).

4) DBSAVE and DBSCAN - use SEQBACK and SEQREST to create multiple backup copies of SEQUEL databases on a single tape, and creates an identifying label for each saved database on the tape (Appendix P).

5) SELDUMP - a selective backup facility, which creates bulk loader CMS files for tables specified by the user (Appendix Q).

2.5. RAM Utilization Workspace

Ray Fessel has brought up an APL workspace that contains functions which measure the utilization of pages in a RAM (used by SEQUEL) database. Instructions for its use are in Appendix R.

2.6. Applications Packages

Several applications packages have been brought up under VM/370 this past year for use in GMIS. Each one of these systems can communicate with the GMIS database through the use of punched CMS files and the VM spooling facility. Specialized communications interfaces for each package will be built under the new GMIS design (Section 2.7).

The following applications packages and systems have been made operational:

1) TROLL

2) TSP

3) BMD

4) EPLAN

5) DYNAMO

6) STATPAC II

7) MPSX

The packages will be briefly described and referenced in the forthcoming GMIS User Guide.

2.7. New GMIS Design

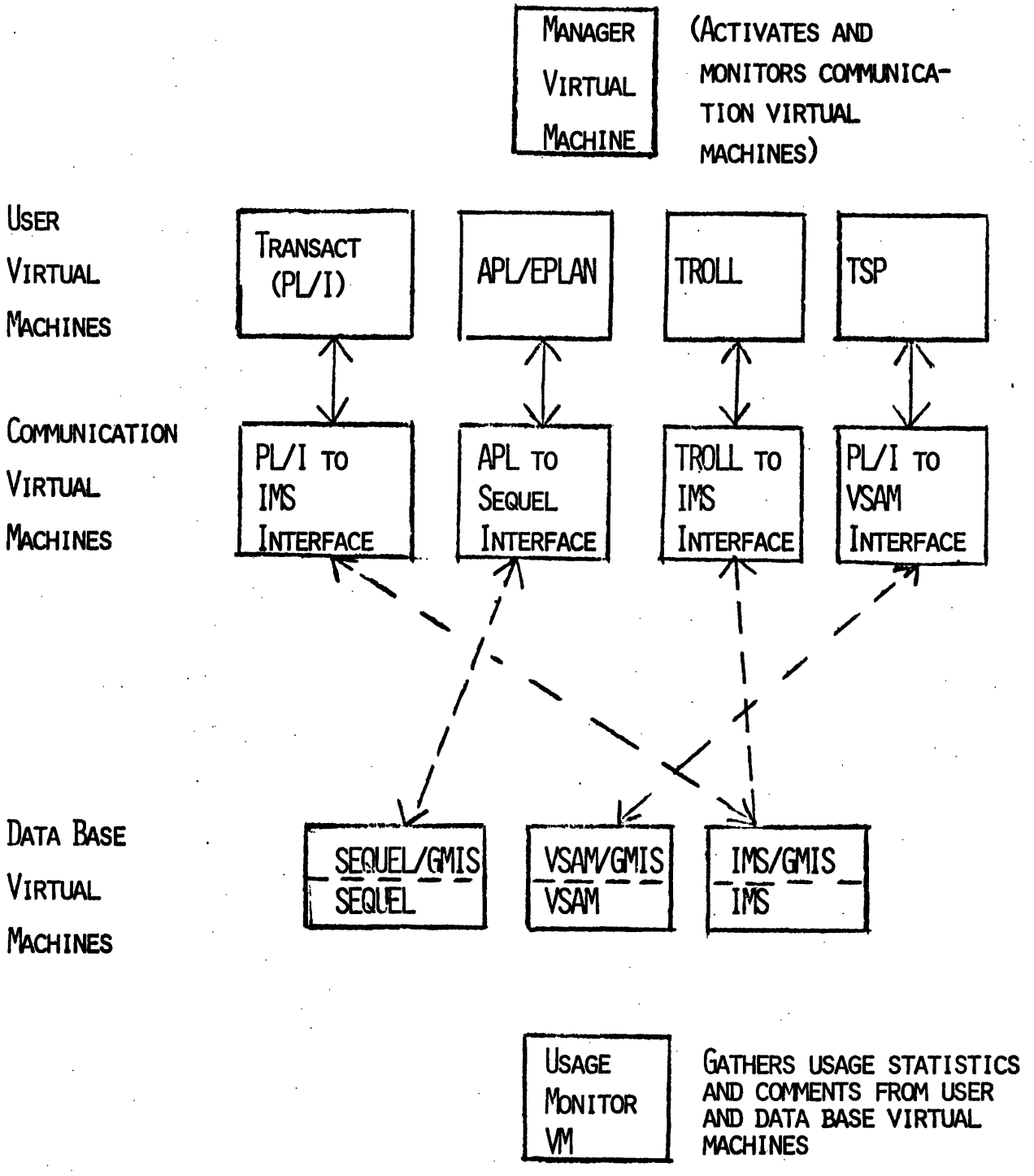
Beginning in June, we launched an effort to redesign the GMIS system to utilize some new VM inter-machine communications software, and to apply what we have learned from the GMIS prototype system to an improved design. In this section, a brief description of the new GMIS will be presented. More detailed system level and user level descriptions of the system are currently being written.

2.7.1. Explanation of New GMIS Configuration

Figure 1 presents an overview of the new GMIS. There are five types of virtual machines defined for this configuration, which allows any available user application packages to communicate with any available database systems. Note that the new GMIS allows for more than one database machine.

Figure 1

NEW GMIS IMPLEMENTATION OVERVIEW



Under this scheme, a user virtual machine (UVM) wishing to send a query to a particular database virtual machine (DBVM) must first signal the Manager Virtual Machine (MGR). The MGR is always logged in, and is activated by signals from UVM's. After the MGR has been signalled, it looks in its GMIS directory (a CMS file) for the ID of Communications Virtual Machine that is associated with the signalling UVM. When it finds the ID, the MGR uses the VM/370 Release 3 AUTOLOG facility to log in the appropriate CVM.

Assuming that there are n different types of UVM (e.g. TRANSACT, TROLL, APL), and m different types of DBVM (e.g. SEQUEL, QBX, IMS), then there are (n x m) possible ways a UVM's can communicate with a DBVM. For example, a TROLL user may want to query an IMS database, or an APL user may want to query a SEQUEL database. Each of these user/database combinations requires a communications interface, which will format user queries for the particular database system, signal the DBVM, format the reply from the DBVM for the UVM, and signal the UVM that the query has been processed. These functions were previously handled by the GMIS Multi-User Interface (MUI). Now, a separate interface program for each UVM/DBVM combination will be written that resides in the CVM. The UVM's and DBVM's will each have standard interfaces, and all query and reply formatting and intermachine communications will be handled by the CVM. (Although there could be as many as (n x m) interface programs, many of the interfaces under consideration will be the same). A UVM need only activate its CVM through the MGR with the appropriate interface program, and any time a UVM has a query to send, it simply signals its CVM, which does the rest of the work.

2.7.2. Inter-machine Communications Mechanism

Much of the spooled virtual card read/punch mechanism that was used for the inter-machine communications facility of the GMIS prototype will be replaced with the new SPY memory-to-memory inter-machine communications software (from IBM Yorktown). The SPY routines send messages much faster than the spooling mechanism.

2.7.3. GMIS Usage Monitor

A Usage Monitor VM (UMVM) will be installed that collects data from DBVM's (e.g. queries processed, execution statistics) and UVM's (e.g. descriptions of applications work being done on the CVM). It will generate a log file that can be retrieved later for performance and usage analysis.

2.7.4. Current Progress on Implementation of New GMIS

At this time, the new GMIS system has been partially implemented and is being tested in a simulated VM environment, which runs under the production VM/370 operating system at the Cambridge Scientific Center. The Manger Virtual Machine can AUTOLOG a communications Virtual Machine after being signalled by a User Virtual Machine, and the TRANSACT/SEQUEL interface for the CVM is almost ready for testing. Some of the additional CVM interfaces planned for the near future include TROLL/SEQUEL, APL/SEQUEL, and APL/GIS.

2.7.5. Query By Example Enhancements

We are investigating the possibility of rebuilding the front end of the Query By Example (QBX) database system to generate a less cryptic DDL/DML that can take advantage of the efficient relational operators that have been built into QBX. This front end could make use of a TRANSACT-like user interface.

3. Computer Technologies Used in the GMIS Effort

In developing GMIS, we have made extensive use of many software and hardware facilities made available through the IBM Cambridge Scientific Center. The software facilities used include:

1) Language processors

- Assembly Language
- PL/I Optimizing Compiler
- FORTRAN
- APLSV
- SCRIPT (for documentation).

2) VM/370 system software

- Modification of VM ASCII translate tables for adaptation of ASCII terminals to VM

- Use of many I/O and communications routines that call the CMS modules directly through BAL interfaces.

- Use of simulated VM/370 under production VM/370 to test new GMIS and SPY code.

3) IBM experimental code

- SEQUEL/XRAM/RAM

- QBX

- SPY

The hardware facilities used in GMIS development, in addition to the standard devices used (e.g. CPU, printers, disks, tapes) include:

1) Terminals

- IBM 2741

- IBM 3270

- Teletype

- TTY - compatible display devices (using light pens).

2) Telecommunications facilities

- IBM 3704 programmed for 2741 lines, 110 and 300 band TTY lines

- MIT extensions connected to VM/370 ports.

4. Advantages of IBM/MIT Joint Study Environment in GMIS Development

The technical and operational environment provided through the IBM/MIT Joint Study has made it possible for the GMIS prototype to be made operational, and for the development of the new GMIS to progress so rapidly. Some of the reasons for this rapid progress are due to the type of technical environment used; the VM/370 operating system is an excellent facility for system development, combining the advantages of interactive time sharing, availability of many language processors, ability to support different operating systems, a very flexible operating system command language (CMS) and many other more subtle features.

Beside these advantages, which could be accessed through many commercially available systems, were several opportunities unique to our relationship to IBM:

1) The opportunity to use and modify IBM experimental software, including RAM, XRAM, SEQUEL, FPLAN (before it was announced by IBM) and SPY.

2) The ability to use VM in unconventional ways (e.g. running VM under VM, accessing CMS routines directly, modifying VM for TROLL and for SPY), with the help of CSC

personnel. Outside vendors might not permit us to try some of these experiments, or our access to the vendor's systems people might be limited.

3) Our close rapport with CSC personnel for the solution of both technical (e.g. fixing bugs, systems design) and administrative (e.g. setting up accounts, obtaining manuals) problems.

4) The fact that we had direct access to the CSC Machine Room, so we could pick up our output quickly and make minor program changes immediately at the local terminals.

5. Suggestions for Technological Improvements

After using the facilities at IBM for over two years, there are some technical areas where improvements could be made, from our point of view. We will list three of those areas which come to mind here:

1) Interactive database management systems - After investigating the availability of interactive database management systems within IBM, we were disappointed to find no acceptable offerings beyond SEQUEL and IMS. SEQUEL is still very experimental, as we have pointed out many times. IMS is restricted to a hierarchial view of data and is not currently compatible with CMS. Much more product oriented work needs to be done in this area.

2) Better VM inter-machine communications - The SIGNAL&SHARE and SPY software facilities are initial efforts to improve communications and signalling between VM's. However, these facilities should be refined and incorporated into the VM/370 product, and eventually these intermachine signalling and shared virtual memory segment facilities should be built into the IBM hardware.

3) Remote printing and display terminals - We have found a need for faster and more sophisticated terminals that can be connected remotely to the computer through ordinary telephone lines. Specific suggestions for improvements in terminals that are possible today would be to build an asynchronous IBM 3270 display terminal so it can be used as a remote stand-alone terminal without a controller, and a 30 to 60 character per second version of an IBM 2741.

Appendix A

Current Uses of IBM Accounts for Systems Development

July 16, 1976

The following is a list of each ID being used for systems development, the name of the principal user of the ID, and the nature of the work being done on the account:

1) Account ID: NEEMIS6

Principal users:

Bryan Mau, Ray Fessel

Purpose of account:

All of the GMIS source code resides on the NEEMIS6 194 disk. The account is used to store the code, and to make minor changes and enhancements to the code.

2) Account ID: NEEMIS7

Principal users:

Clerical help, Louis Gutentag

Purpose of account:

All systems related documentation is entered on this account in SCRIPT. This account is also used for experimental applications of the relational database software, for possible uses in the old and new GMIS.

3) Account ID: NEEMIS8

Principal users:

Bob Selinger

Purpose of account:

Systems programming account. Bob is writing several parts of the new GMIS, and does much of his work on this account.

4) Account ID: GMIS

Principal users:

None (Louis Gutentag responsible)

Purpose of account:

This account contains all of the executable GMIS modules and EXEC files for users to access. It also contains all of the TXTLIB's and MACLIB's necessary to create GMIS modules and to compile GMIS programs, but does not contain the GMIS source code. Finished SCRIPT files are also placed on this account to make GMIS documentation available to users. All GMIS users link to the GMIS account (5/0) before they run any GMIS programs.

5-7) Account ID: GMIS1, GMIS2, GMIS3

Principal users:

Chat Lam, Bob Selinger, Tony Smith

Purpose of accounts:

The new GMIS is being developed and tested under these accounts. GMIS2 runs the simulated VM under VM for testing of the Manager Virtual Machine and the SPY code. The other accounts are used to develop the GMIS communications handlers, interface routines, etc.

Appendix B

Project GMIS Progress Report

August 22, 1975

This document will summarize the progress made on GMIS software develcpment to date.

SEQUEL

After receiving the code for SEQUEL, Release 1 from San Jose, we brought up the system using their 3270 User Friendly Interface (UFI70). Since then, we have written a number of our own UFI's, as described later in this paper. We refrained from making changes to the actual SEQUEL code until we received and brcught up Release 2 of SEQUEL. Our changes to SEQUEL are detailed in the SEQLOG SEQUEL file on NEEMIS6, but they fall into the following general categories:

1) Added DTYPE array to Transaction Interface to pass data types of transaction replies back to the UFI.

2) Added return CODE of 1 for DISPLAY ASSERTION command.

3) Modified SEQUEL/XRAM to handle problem of creating inversions on columns with numeric values greater than 2**21. These inversions will now be suppressed.

4) Fixed assorted XRAM and RAM bugs.

5) Extended SEQUEL implementation restrictions on maximum degree of a table, maximum length of an identifier, and maximum size of a character string constant. Output formatting of rows in TRANS and QUERY were rewritten for generality (for any "magic numbers").

Currently, we are modifying SEQUEL to accept the unary + and - operators as prefixes to numeric literals, and to handle unscaled FIXED DECIMAL constants. After unscaled FIXED DECIMAL is operational, we will extend this feature to include scaled decimal numbers.

The current documentation on these changes is included in these documents:

Log of SEQUEL Modifications
Limitations of SEQUEL Release 3
Transaction Interface Description - MIT/SEQUEL Rel. 3

TRANSACT

TRANSACT is a UFI designed to give users a facility to enter transactions and receive replies from any terminal device supported by VM/370. A description of the current features of TRANSACT are included in the documents:

TRANSACT Commands
Explanation of TRANSACT Commands

We are currently almost finished debugging a new version of TRANSACT that will support the following new features:

1) A DISPLAY option which, when set ON, will display the transaction which TRANSACT was processing before displaying the reply to that transaction. The DISPLAY is sent to the current OUTPUT device.

2) A RUN command macro processor has been written which will allow a user to write EXEC-like files containing TRANSACT and SEQUEL commands. Implemented features include variable substitution, looping, and conditional execution of command lines. A draft version of a document describing this facility is available (RUN Command Guide).

3) If a PROFILE TRANSACT file is on the TRANSACT user's A-disk, it will be RUN before execution of the first TRANSACT command.

We have also prepared a proposal for a TRANSACT interactive transaction editor, which will allow TRANSACT users to modify transactions after they have been entered (e.g. to correct errors). A selective dumping facility is also planned for TRANSACT. This facility will permit users to put the result of any query into a CMS file in bulk loader format. Using this facility, any selected portion of a database may be saved, deleted from the current database, and restored to the database at any future time.

Multi-User Interface (MUI)

A UFI named SEQ2 has been implemented which permits external virtual machines to send transactions to a comman Transaction Virtual Machine (TVM) with a SEQUEL database. Communications facilities are provided through the use of shared multi-write CMS transaction and reply files, and interprocessor signalling by means of virtual card readers and punches. This facility is described fully in Louis

Gutentag's Master's Thesis, Section 4.2.

There is some room for improvement in performance using this MUI implementation. However, we plan to bring up a modified version of San Jose's SIGNAL&SHARE to replace our slow system.

Transparent TRANS Interface for the MUI

A procedure called TRIFUFI has been written to simulate the SEQUEL Transaction Interface for the MUI. This means, for example, that a call to SEQUEL using TRIFUFI instead of TRANS will actually write QSTRING into a transaction file, signal the TVM to process the transaction, and wait for the TVM to signal that it has written the reply file.

This procedure permits us to use our existing UFI's (e.g. TRANSACT) in a multi-user environment without modifying their source code. An EXEC called REMGEN on GMIS 191 performs the same function as CRTMOD, but uses TRIFUFI, so resulting modules will run only with the MUI instead of running on a machine with its own database.

APL Interface

A series of assembly code routines and APL functions have been written to allow SEQUEL transactions to be passed to a TVM through the Multi-User Interface. Transactions are passed to the SEQUEL machine (TVM) through the use of the following APL statement:

```
Z <- QUERY '<SEQUEL transaction>'
```

where Z is the return code from SEQUEL. The results of the transaction are stored as vectors or matrices in APL variables, which are named after the columns that they represent.

APL / TRANSACT

A subset of TRANSACT has been implemented as an APL function to allow users in the APL environment to display the results of queries in a tabular format without having to leave APL. Complete documentation on the APL Interface and on APL TRANSACT must still be written.

Bulk Loaders

The original LSDB bulk loader received from San Jose has gone through several passes of revisions. First, the loader was enhanced to permit rows that spanned more than one 80 byte record to be loaded. This bulk loader was called LTDB.

The next series of changes to LTDB were designed to make it compatible with our new version of SEQUEL that permits identifiers of up to 16 characters. For generality, LTDB was extended in the following ways:

1) Free format control cards.

2) Multiple card primary key descriptions. A $ENDKEY control card was added to permit this extension. Previously, the $PRIKEY card only supported primary keys of seven columns or less.

SEQUEL Release 2 includes a facility for making integrity assertions on tables in the database. LTDB calls XRAM directly, and bypasses this integrity facility, which is implemented at the SEQUEL level. Therefore, we generated two new bulk loaders to replace LTDB. LTDBA is the same as LTDB and calls XRAM directly. LTDBB is functionally equivalent to LTDBA, but is really a SEQUEL UFI which bulk loads rows into tables by means of INSERT commands. The tradeoff between using the two loaders is this: LTDBA is fast and ignores integrity assertions, and LTDBB is very slow and checks each inserted row against the stored integrity assertions. The document "Bulkloading of a Database" describes the use of the current bulk loaders in greater detail.

SEQUEL Database Backup Utilities

There are currently two methods to create backup copies of SEQUEL databases: using the SEQDUMP stand-alone utility program or the CMS utility DDR.

SEQDUMP is a SEQUEL UFI which dumps a complete copy of the database to CMS files in bulk loader format. This permits users to add that data to their database at any time, using the bulk loader. The DDR program dumps a "snapshot" of the disks containing the database, and can be used only to restore the database to its state of integrity at the time it was dumped. The tradeoff here is that SEQDUMP creates a more flexible backup, but is slow when dumping because it has to issue many SEQUEL transactions to perform the dump. DDR, on the other hand, dumps the

database quickly, but any existing database is replaced at restoration time by the version that was dumped by DDR. SEQDUMP is described in the paper "Creating Backup Dumps of SEQUEL Databases", and information on the use of DDR can be obtained from Ray Fessel or Bryan Mau.

RAM Utilization Workspace

Ray Fessel has written an APL workspace that plots the amounts of disk space being utilized by a multi-segment RAM database (used for SEQUEL) at any given time. See Ray Fessel for further information.

TSP

A version of the Time Series Processor statistics package is currently being brought up under CMS by Tony Smith and Kai Wong. This is the version that is being used at the MIT Information Processing Center. After our CMS version of TSP is debugged, we will explore methods to make it an interactive system, and to add an interface to our GMIS Multi-User Interface.

TROLL

We are also in the process of bringing up a copy of the TROLL econometric modeling system under the VM/370 system at the Cambridge Scientific Center. There are several problems which are causing delays in our progress on this project. First, the VM version of TROLL from NBER requires modifications to the VM/370 operating system to support their shared memory segment and inter-processor signalling functions. These modifications must be made compatible with other existing versions of shared memory and signalling (e.g. San Jose, Yorktown, CSC). Also, TROLL is invoked under VM/370 as a named system (like CMS), and uses its own unique file system. This makes the process of building an efficient interface between TROLL and the GMIS Multi-User Interface a formidable task. However, as always, we can provide interim solutions by bringing up TROLL on a VM under VM, and we can use virtual card readers and punches to communicate with GMIS.

Appendix C

Visit to IBM San Jose Research Laboratory

August 27-29, 1975

Louis M. Gutentag

Overview of Topics for Discussion and Presentation

1. Progress update on IBM/MIT Joint Study software development.

2. Presentation of Project NEEMIS applications.

3. Performance analysis of SEQUEL, Release 2. Identification of areas where performance is especially weak, and could be improved in current release:

   3.1 SEQUEL level (objective in most cases is to reduce calls to XRAM).

      3.1.1. Absence of GROUP BY operator.

      3.1.2. Inefficient handling of joins through nested queries.

      3.1.3. Inefficient resolution of complex predicates (too many calls to TSTNODE).

      3.1.4. Creation (or non-creation) of temporary inversions on NUM and CHAR domains in SEQUEL predicates does not fully utilize inversion facility.

      3.1.5. Technique used to generate SEQUEL projections causes excessive time to be spent in XRAM (really an XRAM design problem).

   3.2. XRAM level

      3.2.1. Optimization of page fetches vs. memory resident page searches.

      3.2.2. Optimization of code in specific XRAM and multi-segment RAM routines.

4. Discussion of short and long term goals of San Jose Data Base Group.

   4.1. Applications orientation vs. research orientation.

4.2. Development of user interfaces.

4.3 Time frame for resolution of performance issues.

4.4 Exploration of alternate approaches to system development or incorporation of existing IBM products (e.g. IMS).

4.5. Plan for release of systems (e.g. System R) to wider user audience for testing and feedback.

5. Analysis of System R as a system to meet future needs of IBM/MIT Joint Study.

5.1 Performance analysis of System R.

5.1.1. Apply what we have learned from SEQUEL.

5.1.2. Handling of joins.

5.1.3. Reduction and optimization of transactions to some normal form (i.e. make DML truly non-procedural). Database accessing techniques.

5.1.4. Data clustering and garbage collection, other space utilization issues.

5.1.5. Automatic (user-transparent) reorganization of database structures based on types of user transactions applied to database over time.

5.1.6. Optimization and minimization of use of virtual storage in VM.

5.2 Feature analysis of System R.

5.2.1 Compatibility of SEQUEL vs. System R Transaction Interfaces for transfer of existing UFI's and replacement of XRAM with RSI (how much debugging will RSI need?).

5.2.2. Data types supported (FIXED DECIMAL, FIXED BINARY, FLOAT BINARY, CHARACTER).

5.2.3. Handling of NULL values, especially for returned values of statistical functions.

5.2.4. Extensibility of DML, especially for addition of statistical functions and possible macro definitions. General extension of System R computational capabilities to reduce dependence on APL and UFI's for numeric transformations.

5.2.5. Possible implementation restrictions (e.g. "magic numbers" problem in SEQUEL, maximum # of rows in table, maximum # of pages/segment, running out of ID's).

5.2.6. User views and accessing of system tables.

5.2.7. Dependencies on SIGNAL&SHARE modifications to VM.

5.2.7.1. Explore problems relating to acceptance of modifications by IBM or at customer installations.

5.2.7.2. Discuss NBER modifications to VM for TROLL - compatibility and efficiency issues. Need to agree on standard interface for different implementations of SIGNAL&SHARE.

6. Discussion of documentation problems.

6.1. SEQUEL Language Reference Manual.

6.2. Improved documentation of code and algorithms used.

Appendix D

Log of SEQUEL Modifications

4/30/75 L. Gutentag

Modified UFIDCL to include array called DTYPE which contains the data types of output columns, declared DTYPE (MAX_DEGREE) BIN FIXED EXT.

5/1/75 L. Gutentag

Modified QUERY to set DTYPE to the data types of output columns.

5/13/75 R. Fessel (logged by L. Gutentag)

Modified TRANS so that all calls to SEQUEL with FILESW='1'B will return through the interface. Previously, if FILESW='1'B and CODE=0, the next statement in the file would be read without a new call to SEQUEL by the UFI.

5/14/75 R. Fessel (logged by L. Gutentag)

Modified DDASERT so that a DISPLAY ASSERTION command will return with CODE=1 instead of CODE=0.

5/29/75 R. Fessel (logged by L. Gutentag)

Modified XPARSE so that the size of BUFFER is declared as CHAR (QSTRING_LENGTH+1) VAR.

6/24/75, R. Fessel

Modified XRAM routine XRINV so that it returns 9 in RCODE when an attempt is made to create an inversion but one of the domains contains numbers outside the range between 1 and 2**21. In this case, the inversion control tuple for the relation must be tested to see which domains failed to have inversions created because for those domains where inversion creation is possible, such inversions are actually created.

6/25/75 R. Fessel

Modified CRTINV so that it detects a return code of 9 from XRINV. In this case an error message is composed with a return code of 99 and the name of the failed column(s) is

appended to the message. Additionally the type 9 relations originally created for these inversions are dropped. Any inversions which succeed proceed normally.

6/30/75 R. Fessel

Modified GENINV so that if it gets an FCODE of 9 from XRINV when called to create a temporary inversion, it returns a special value in CODE to PASS3, namely 999.

7/1/75, R. Fessel

Modified XPARSE routine so that if the STATIST bit is on, and the FILESW bit is on, every time a record is read from the query file, it is also written out to the statistics file. Thus the statistics output will always contain the query, even if it is input from a file.

7/8/75, R. Fessel

Modified IXECLCK ASSEMBLE file so that virtual and total times are now returned in milliseconds rather than microseconds. This prevents fixed overflows in statistics package.

7/9/75 R. Fessel

Modified PASS3 routine in accordance with suggestion from Mort Astrahan so that it properly handles a node which it tries to create a temporary inversion for but for which GENINV returns a code of 999, indicating that the column contained invalid numeric data. In this case the node in question has to do multiple scans of the relation.

7/9/75 R. Fessel

Modified IXECLCK routine again so that times are returned in units of 10 microseconds. This will still prevent fixed overflows in statistics package while still retaining adequate resolution.

7/14/75 R. Fessel

Modified XRAM routine XRINV so that ZGDEL internal subroutine does not modify variables used elsewhere. This modification means that the type 9 XRAM relations are now correctly dropped when either XRDRINV is invoked or when an inversion relation is dropped internally because it contains

numeric data outside the range between 1 and 2**21.

7/15/75 R. Fessel

Modified XRAM routines XRFETCH and XRFTCHT so that the page pointed to by pointer P2 is now unlocked. This page contains the XRAM relation used in the arguments to these 2 entry points.

7/15/75 R. Fessel

Modified DRPINV so that it does not call XRDROP to drop the type 9 relation associated with the inversion since XRAM now drops this relation automatically.

7/15/75 R. Fessel

Modified CRTINV so that it no longer does an explicit call to XRDROP for the type 9 relation associated with an inversion that failed to be created.

7/15/75 R. Fessel

Modified GENINV so that it no longer does an explicit call to XRDROP for the type 9 relation associated with an inversion that failed to be created.

7/17/75 R. Fessel

Fixed bug in RAM routine ZRELID (in assembly ZGETID) so that control block is not clobbered when the last ID on a relation page is freed and the third page in the cell has not been allocated. Additionally, the page which had its last ID deleted was not being marked as empty and relation pages were not being reused.

7/23/75 R.Fessel

Modified RAM routines ZOVFLOW and ZRINGF. Reinstalled changes to correct bugs for very large relations which had been put in about 18 months ago but had got lost in the San Jose version.

7/24/75 L. Gutentag

Modified UFIDCL to include new "magic numbers":

```
MAX_DEGREE=32 (was 15)
MAX_NAME_LENGTH=16 (was 8)
MAX_STRING_LENGTH =128 (was 64)
QSTRING_LENGTH=4200 (was 2000)
```

Added new magic numbers for use in QCAT and UFI's:

```
SKIP_NAME=MAX_NAME_LENGTH+1;
NAME_SPACE=MAX_NAME_LENGTH-2;
DOMAIN_FORMAT=MAX_NAME_LENGTH+17;
TABLE_LENGTH=MAX_NAME_LENGTH+2;
DOMAIN_SPACE=MAX_NAME_LENGTH-4;
TABLE_FORMAT=2*MAX_NAME_LENGTH+21;
```

7/24/75 L. Gutentag

Modified TRANDCI to include new "magic numbers":

```
COLUMN_SIZE=40 (was 30)
DOMAIN_SIZE=40 (was 30)
NUMBERS_SIZE=256 (was 30)
STRINGS_SIZE=256 (was 30)
```

7/24/75 L. Gutentag

Modified all SEQUEL routines in which the "magic numbers" appeared as constants. The correct preprocessor variable was substituted for each constant. The greatest occurrences of these constants were in routines utilizing the shadow catalogs and in QCAT.

7/24/75 L. Gutentag

Modified TRANS starting on line 1460 to revise the technique used in GETROWS to format output rows in QSTRING. Now, all output columns will be centered under their column titles. This new code will not be affected by a change in MAX_NAME_LENGTH as was the previous code.

7/24/75 L. Gutentag

Modified QUERY starting at line 750 to format column headings in a general way. The previous code was written assuming MAX_NAME_LENGTH=8. Now, regardless of the value of MAX_NAME_LENGTH, all column titles will be centered over the column data formatted by GETROWS in TRANS.

7/24/75 L. Gutentag

Deleted the declaration for REPAREA ENTRY in TRANS; included that declaration in UTILDCL.

7/30/75 R. Fessel

Fixed RAM routine ZTRIAL in accordance with insructions from Raymond Lorie. Solved problem with retrieval from relation with a single B-ring on an A page and several C-rings on C pages. Symptom was that when NEXT was invoked until end of relation was returned, every entry was retrieved twice.

7/31/75 R. Fessel

Made 2 significant charges to SEQUEL routine XPARSE. All declarations generated by the parser generator are now brought in with a %INCLUDE statement from a library member called PARSDCL. Similarly, all the semantic routines in subroutine SYNTHESIZE (part of XPARSE) are now brought in with an %INCLUDE statement from a library member called SEMANTIC. The file PARSDCL COPY was generated from the parser generator using as input the grammar deck received from Jim Mehl which is the grammar for SEQUEL, Release 2. The file SEMANTIC COPY was created by editing it out of the old copy of XPARSE.

7/31/75 R. Fessel

The SEQUEL grammar was modified to include unary + and - operators prefixing numeric literals. This modified grammar was then run through the parse table generating procedure to produce a new version of PARSDCL COPY. Also the file SEMANTIC COPY was modified to include semantic routines for the new productions.

8/5/75 R. Fessel

GENCAT was modified to put the string 'DEC' in the SYSCHAR class relation. This constant is necessary so that various SEQUEL routines will be able to recognize decimal data type.

8/6/75 R. Fessel

The variable CDECTID was added to the catalog declarations in the file XRMDCL COPY. This variable will be initialized to the TID of the string 'DEC' in the SYSCHAR class relation.

8/6/75 R. Fessel

The SEQUEL grammar was modified to make DEC a valid domain type. The new grammar is in the file DECIMAL GRAMMAR. This file was used to generate a new version of PARSDCL COPY. Also the file SEMANTIC COPY was modified to include a semantic routine for the new production. These modified files were then used to generate a new object deck for XPARSE.

8/6/75 R. Fessel

SEQUEL routines INITCAT, QCAT, DEFDOM, DELDOM, and DEFTAB were all modified to hardle dcmains with a data type of DEC. The internal code for this data type was assigned as 3.

8/7/75 R. Fessel

Modified NUMDCL COPY file to include a preprocessor declaration for DECIMALTYPE which is set to 3.

8/7/75 R. Fessel

Modified TRANDCL COPY file to include a declaration for a DNUMBER array which holds the decimal representation of numbers in a SEQUEL statement.

8/7/75 R. Fessel

Modified SEMANTIC COPY file so that all numeric literals in a SEQUEL statement are put into the DNUMBER array as well as the NUMBERS array. The same index, NUMX, is used for both arrays. This modified file was used to generate a new copy of XPARSE TEXT.

8/7/75 R. Fessel

Modified ASSERT routine so that no special processing is done for a predicate with a BETWEEN ... AND clause. Instead, such predicates will be handled by SELECT, as are all other predicates currently.

9/3/75, R. Fessel

Modified XRAM routine XRINV in accordance with directions from Raymond Lorie, adding loop to lock buffers containing tuple id lists at top of main loop creating RAM relations for the inversions.

10/8/75 R. Fessel

Modified UFIDCL COPY, TRANS and XPARSE to implement the LOG facility. Added to UFIDCL a new single bit variable in static external called LCG. Modified TRANS and XPARSE so that if LOG is on, the SEQUEL statement is always sent to the STATPRI file, along with a timestamp. Also sent to the STATPRI file is the transaction type (TRANTYP) and a list of all columns and tables referenced in the statement.

R. Fessel 10/20/75

Modified RAM routine ZBHASH so that it no longer returns a hash value of 0 for data pages whose last 2 hex digits are X'7F' or relation pages which end in X'FF'.

R. Fessel 10/21/75

Modified routine TSTNODE in accordance with instructions from Mort Astrahan so that tuple predicates with multiple columns are handled properly.

R. Fessel 11/18/75

Corrected XRAM routine ZGFTF so that the upper limit of the DO loop for finding overflow cells with empty pages is now correct.

11/19/75 R. Fessel

Corrected SEQUEL routine TRANS for errors introduced in the output formatting when the name lengths were increased to 16 characters.

11/28/75 R. Fessel

Modified GENCAT so that LEN of entries for RELNAME,CNAME,COLNAME, and DOMNAME in both SYSTAB and DOMCAT are initialized to 16 (MAX_NAME_LENGTH). This was done to obviate necessity of updating these relations when domains, columns,assertions, and tables are created either using SEQUEL or the bulk loader.

12/1/75 R. Fessel

Modified RAM routine ZGFTF to correct error introduced while

making fix noted above. Upper limit of DO loop for scan of overflow cells was off by 1.

1/12/76 B. Mau

Installed security features for SEQUEL database. Modified UFIDCL to include external bit(1) variable SECURTY-indicates to SEQUEL whether or not to perform access rights checking. Modified routine TABINIT--inserted a check just before final return to test SECURTY and call the access checking routine (SECCHK) if indicated. Modified routine XWTAPL to signal condiation 'PLXXX' if error linking 340 or error accessing it; signal should be picked up by the UFI (SEQ3); this is to prevent error on part of calling machine from stopping the UFI. Modified routine TRANSACT to set SECURTY to '0'B--default is no security for single-user mode. Updated routine SEQ2 to implement security design; new version renamed SEQ3. SEQ3 to call three new routines: FILECHK, SECSIZE and SECCHK. Inserted ON-block into SEQ3 for condition 'PLXXX'--causes UFI to simply ignore present request and return to XW1API. Allowed for 2 new parameters to SEQIF: 'NODISC' suppresses the disconnect command; 'SECURE' invokes the security mechanism.

1/15/76 R. Fessel

Modified XRAM routines ZGETP and ZGETB to change algorithm used to find whether a page or block which is requested is already in a buffer. New algorithm is that, if it is not the most recently requested page or block, the page or block number is hashed and the corresponding hash table entry is tested. If the high order bit is off, the page or block in the buffer pointed to by the hash table entry is tested. If it is the right one, the requested page or block has been found. Otherwise the chain of buffer control blocks is scanned to find if the right page or block is in one of them. This solves the problem which occurred when an overflow page was emptied and its buffer was freed. When this occurs, the high order bit of its hash table entry is turned on. If a subsequent request was made for a page or block whose hash value conflicts with the released page, it would apper that that page was not in a buffer because the high order bit of its hash table entry was on even though that page might really be in a buffer.

1/20/76 B. Mau

Modified UFIDCL to include array LPAD (MAX_DEGREE)--to contain for each column (on return from GETROWS) the number of extra blanks added to the left of the column's field in QSTRING. Modified TRANS to set LPAD in GETROWS. Modified SEQDUMP to use LPAD to get the proper part of QSTRING into

its output file in these cases where the column name is longer than 8 characters or the LINE for the column.

2/10/76 B. Mau

Modified routine XPARSE. Alphabet extended to include several heretofore invalid special characters (eg ?, %, #). Also error messages revised.

2/10/76 B. Mau Modified routine GETROWS; replacd with new algorithm for stuffing CSTRING. Also modified routine SEQUEL to set TRANTYP to PARSNOTREC if response from routine PARSE indicates that it did not complete the parse (preciously, TRANTYP remained unchanged through a call to SEQUEL if the parser failed). Also modified macro NUMDCL to include new transaction type PARSNOTREC (999).

2/10/76 B. Mau

Modified routine QUERY to set LPAD external array. Previously, TRANS did this.

2/12/76 R. Fessel

Modified RAM routine ZREPLAT so that it checks each entry in a binary relation to make sure that it is less than 2**21. Previously, entries were checked and rejected only if they were zero or negative.

2/12/76 R. Fessel

Modified RAM routine RAMBOMB so that it prints the RAMBOMB code at the terminal whenever it is invoked.

2/18/76 R. Fessel

Modified XRAM routine XRINV so that when it is scanning a page for tuples in a particular relation, if it finds an entity that is not a tuple in the relation, it does not unlock that entity if it is the master control tuple for that relation.

3/11/76 R. Fessel

Modified RAM routine ZGETBIK so that when the first bit map for free pages is filled, i.e. the first 2048 physical pages are all in use, switching to the second bit map is handled correctly.

4/2/76 R. Fessel

Modified XRAM routine XRUPDT so that when a control tuple is being updated, no entry is made in any inversions which may exist on the domains being updated. This is true for all control tuples except for the inversion control tuple which cannot be updated by XRUPDT.

4/6/76 R. Fessel

Modified XRAM file NROPEN which contains entry points XROPEN, XRCLOSE, etc. with several modifications to bring module up to date with respect to modifications made in San Jose.

4/9/76 R. Fessel

Modified XRAM file XRAM which contains entry points NROPEN, XRDEF, XRMAKE, etc. with several modifications to bring module up to date with respect to modifications made in San Jose.

4/29/76 R. Fessel

Modified XRAM decks XRAM, NROPEN, and XRINV so that when a call is made to either XRMAKE or XRUPDT and the new or modified tuple would cause an invalid value to be put into any existing inversion, then an ECODE of 9 is returned and the inversions(s) in question is automatically dropped. The only way to find out which inversion(s) are dropped is to examine the inversion control tuple after the call to XRMAKE or XRUPDT.

Appendix E

Limitations of SEQUEL Release 3

1. Maximum degree of relation = 32 (was 15 in SEQUEL Release 2).

2. Maximum length of name of relation or column or domain = 16 characters (was 8 in Rel. 2).

3. Maximum length of character string data item = 128 (was 64).

4. Data types supported for database: integers (NUM) and varying length character strings (CHAR).

5. Constraints on Numeric Data:
   A) Only numbers in the following range can be entered into numeric columns:

      -2,147,483,648 to 2,147,483,647

   B) Inversions on numeric columns can only be created when all data in the columns is in the following more restrictive range:

      1 ! n ! 2,097,151

6. Additional constraints on table, column and domain names:
   A) The following are <u>invalid</u> as table, column and domain names:

AND	ARE	ASSERT	ASSERTION	AVG
BETWEEN	CHAR	COMPUTE	COUNT	CREATE
DELETE	DESCRIBE	DISPLAY	DOMAIN	DOMAINS
DROP	EMPTY	FROM	IN	INSERT
INTO	INVERSIONS	IS	KEY	KEYS
LIST	MAX	MIN	NUM	ON
OR	SELECT	SET	TABLE	TABLES
TOT	UNIQUE	UPDATE	WHERE	

B) The following characters may not be included in a table, column or domain name:

. (period)
, (comma)
; (semicolon)
: (colon)
< (less-than)
> (greater-than)
= (equals)
((left-paren)
) (right-paren)
+ (plus)
- (minus)
* (asterisk)
/ (left slash)
¬ (logical not)
| (logical or)
! (exclamation)

7. An input SEQUEL query cannot be longer than 4200 characters unless it is input via a file (was 2000 bytes in Rel. 2).

8. Maximum amount of nesting in a SEQUEL query: 3 levels (outer block plus 2 levels of nesting).

9. The maximum number of nodes in the parse tree of a SEQUEL query is 100.

10. The following features of the BNF syntax of SEQUEL are not implemented:

<SET-COMP> (production 69)

+ - * / may not be used in a <SET-CLAUSE> (production 33)

<UNIQUE-PRED> (production 71)

Appendix ?

Transaction Interface Description

MIT-SEQUEL Release 3

The UFI (User Friendly Interface) interfaces with the rest of the SEQUEL system by calling the procedure TRANS. The UFI shares the following external variables with TRANS:

```
QSTRING   CHAR(4200) VAR,
DTYPE(32), CCILEN(32), LPAD(32), DWIDTH(32)   BIN FIXED,
(TRANTYP,CODE,DEGREE,CARDIN,NUMROWS,DISPROW)  BIN FIXED,
(FILESW,REPORT,STATIST)  BIT(1);
```

TRANS has five entry points: TRANSIN, SEQUEL, GETROWS, SEQSTOP, and WINDUP, which are described below. The valid return codes for each entry point are listed. If CODE contains any other number after return from TRANS, the number is a code for some user or system error, and a message describing the error may be found in QSTRING.

TRANSIN: Must be called at beginning of user session to initialize the system. Normal return code = 0.

SEQUEL: If FILESW='0'B, a SEQUEL statement to be processed is in QSTRING. If FILESW='1'B, one or more SEQUEL statements are in the CMS file SEQSTAT SEQUEL. (In this latter case, successive calls to SEQUEL process the statements one at a time.) Normal return codes:

CODE=0: The SEQUEL statement was successfully executed. The type of the statement is indicated by TRANTYP:

- 102 Insert
- 103 Update
- 104 Delete
- 106 Create Domain
- 107 Drop Domain
- 108 Create Table
- 109 Drop Table
- 110 Create Inversion
- 111 Drop Inversion
- 112 Assert
- 113 Drop Assertion
- 114 Display Assertion

CODE=1: The SEQUEL statement was a DISPLAY ASSERTION command. The assertion statement is found in QSTRING as it was originally entered.

CODE=2: The SEQUEL statement was a query which evaluated to a table. The following information is available:

TRANTYP = 101 denotes query.
DEGREE = the degree of the query result.
CARDIN = the cardinality of the query result.
DTYPE(I) = the data type of column I of the query result, where
1 = NUM, and
2 = CHAR.
COLLEN(I) = the width (in characters) of the fields in QSTRING that correspond to column I
QSTRING contains the column-names of the query result, centered in consecutive fields of width COLLEN(I) characters.
(The actual rows of the query may be obtained by calling GETROWS.)

In addition, the two arrays IPAD and DWIDTH are available after the call to SEQUEL, although they are not useful until the call(s) to GETROWS.

CODE=3: SEQUEL was called with FILESW='1'B to process the next statement in the file SEQSTAT SEQUEL, but the last statement has already been processed (end of file reached).

CODE=4: The SEQUEL statement was a query which evaluated to a single constant. The query result, in character string form, is in QSTRING. DTYPE(1) contains the data type of the query result.

CODE=6: The SEQUEL statement was of type "LIST DOMAINS." The list of domains is in QSTRING, containing for each domain its name, type, usage count, and length of longest entry, in format A(16), X(2), A(4), X(2), F(3), X(2), F(4). The number of domains is in NUMROWS. TRANTYP = 105.

CODE=7: The SEQUEL statement was of type "LIST TABLES." (TRANTYP = 105. NUMROWS = the number of tables.) QSTRING contains all the table names, left-justified in 18-byte fields.

CODE=8: The SEQUEL statement was of type "DESCRIBE TABLE." TRANTYP = 105. NUMFCWS = the number of columns of the table described. QSTRING contains the name of the table (A(16)) followed by a description of each column: name, domain, type, encoding, part of key (YES/NO), inversion exists (YES/NO): A(16), X(2), A(16), X(2), A(4), X(2), F(1), X(2), A(3), X(2), A(3).

Other return codes indicate an error condition. If TRANTYP = 999, then the parser did not recognize the command; otherwise, TRANTYP should indicate what type of command was attempted.

GETROWS: Called repeatedly to obtain the actual rows of a query result after calling SEQUEL with CODE = 2. Caller must set DISPROW = the number of the first row he wishes returned. GETROWS will place as many full rows as possible in QSTRING, concatenating them. The format of each row is identical for each row that is put into QSTRING, and is determined by the arrays COLLEN, LPAD and DWIDTH. Within each row, there are (DEGREE) fields concatenated together--one for each column of the row. The total width of each column field is given by COLLEN--that is, column I has a field of width COLLEN(I). Within that field, there are LPAD(I) spaces on the left, added for centering purposes, then the data item itself, DWIDTH(I) characters wide, and then (possibly) some right-padding spaces (that is, the number of right-padding spaces is given by COLLEN(I) - LPAD(I) - DWIDTH(I)). (Note: either or both of LPAD(I) and DWIDTH(I) may be 0). The number of rows placed in QSTRING is returned in NUMROWS. Normal return code = 0.

SEQSTOP: SEQUEL has previously been called with FILESW='1'B. Some statements may remain in the file. SEQSTOP causes the system to ignore the remaining statements and close the file. Normal return code = 0.

WINDUP: Must be called to clean up the system at the end of a user session. Normal return code = 0.

Whenever a call is made to any of the entry points of TRANS, the following two bits govern the behavior of the system:

REPORT: If REPORT='1'B, the system types out on the

terminal a diagnostic trace of all routines entered.

STATIST: If STATIST='1'B, the system keeps statistics on time spent in various aspects of processing; before returning, it prints the statistics together with the current SEQUEL statement on the STATPRI LISTING file.

Appendix G

XRAM Extensions

F. Fessel

May 13, 1976

This document describes extensions which have been made to XRAM since the publication of the original XRAM documentation.

The following XRAM routines have been functionally modified:

XRINV

A code of 9 is now returned if at least one of the domains upon which an inversion is being created contains data which is not in the inclusive range of 1 to 2**21-1. The only way to tell which domain or domains contain such data is to examine the inversion control tuple upon return from XRINV. When a code of 9 is returned, the type 9 relation(s) corresponding to the offending domain or domains is automatically dropped.

XRMAKE

When a new tuple is added to a type 4 or 5 relation and the addition of that tuple would cause at least one domain of the relation which has an inversion defined on it to contain invalid data (not in the inclusive range between 1 and 2**21-1), then the inversion(s) in question are automatically dropped, the corresponding type 9 relation(s) are dropped, and a code of 9 is returned to the caller. Again, the only way to determine the offending domains is to examine the inversion control tuple.

XRUPDT

When a domain in a regular relation is updated and that domain has an inversion defined on it, and the updated value for the domain is invalid, then the corresponding inversion is dropped, the associated type 9 relation is dropped, and a code of 9 is returned to the calling program.

A new entry point has been added to XRAM. Its purpose is to count the number of unique entries in a domain of a regular relation which has an inversion defined on it. The format of this call is:

XRUCNT (rid,m,k)

rid is the relation id of the relation in question. m is the domain number for which a count of unique entries is desired. The count of unique entries is returned in k, the third argument.

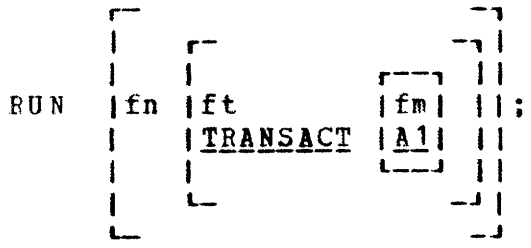
Return codes:

0: Action performed - no error.
3: Invalid relation id.
4: Not a regular relation or invalid domain number.
6: Specified domain does not have an inversion defined on it.
100: System error.

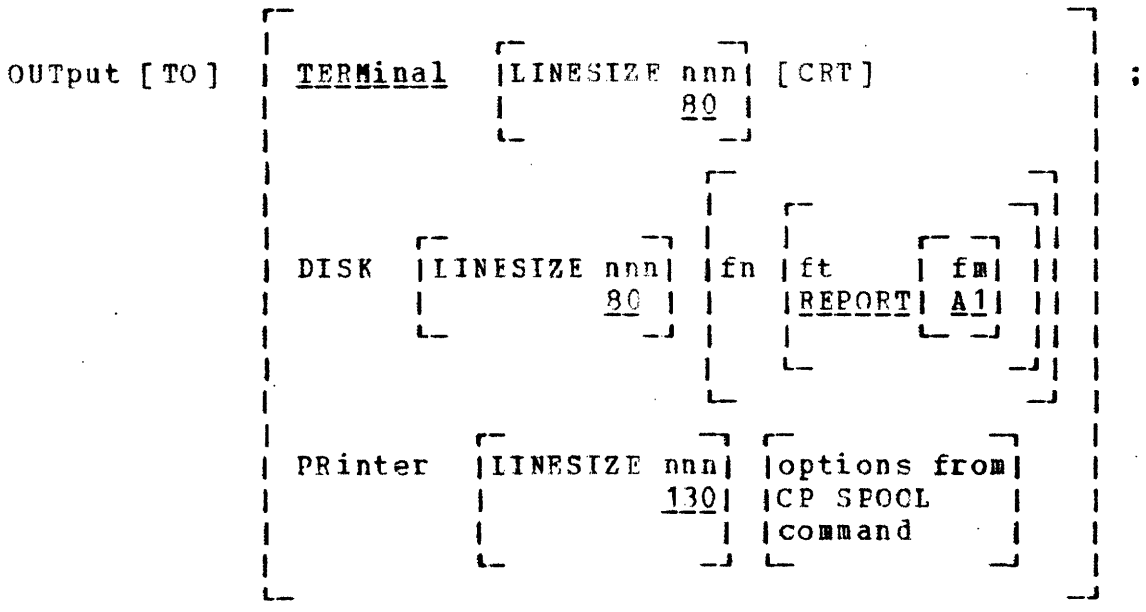
Appendix H

TRANSACT Commands

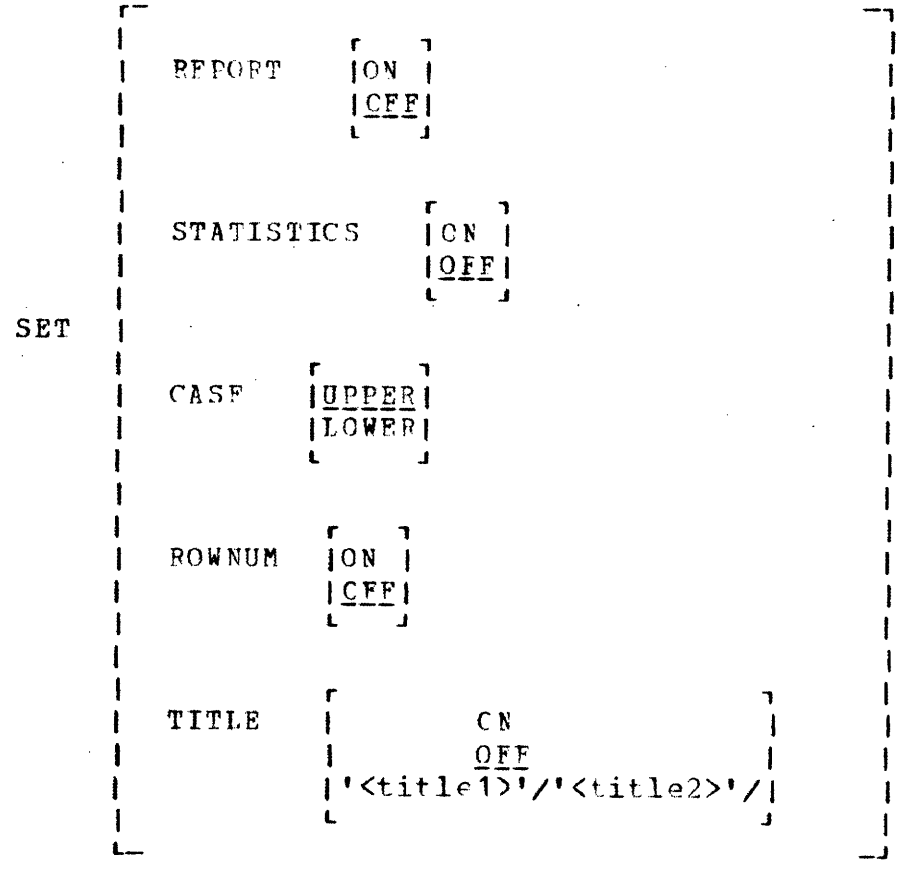
1.



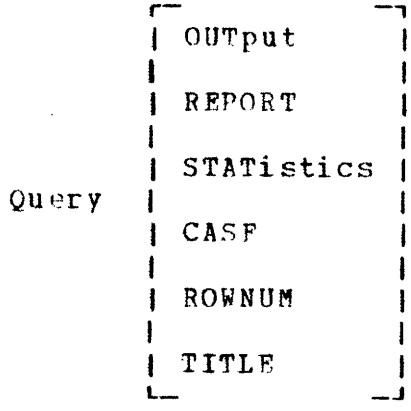
2.



3.



4.



5.

CP <command> ;

6.

CMS <command> ;

7. Any SEQUEL transaction.

8. QUIT;

Appendix I

Explanation of TRANSACT commands

1. RUN (not implemented)

Executes the transactions in the file specified by fn ft fm. Only the transactions described on the pages labelled SEQUEL SYNTAX should be included in a run file. Transactions will be processed sequentially and the system will respond with the message TRANSACTION COMPLETED after the last transaction has been processed. Errors may or may not cause processing to terminate, depending on the nature of the particular error involved. Run files must consist of variable length records (in CMS EDIT type RECFM V).

2. OUTPUT

Directs all output except for messages to the device specified.

OPTIONS

LINESIZE

The user may indicate a linesize for terminal, disk, or printer as long as it conforms to the limitations of the output device. Unless otherwise set the following LINESIZE values will be in effect:

for terminal - LINESIZE 80
for disk - LINESIZE 80
for printer - LINESIZE 130

TERMINAL

All output will be displayed at the terminal. If the CRT option is not specified output tables will be folded.

CRT

This option will enable the resultant tables from the SELECT command and the catalog queries LIST DOMAINS, DESCRIBE TABLE to be displayed in "window" format.
Commands which may be used with the window are:

Up [n]
┌─ ─┐
|Down [n] |
|Next |
└─ ─┘
Left [n]
Right [n]
Continue

UP n, DOWN n, LEFT n, RIGHT n will cause a displacement of n rows or columns in the corresponding direction. CONTINUE will allow the system to accept the next TRANSACT command.

DISK

Output will be directed to the file indicated by fn ft fm.

PRINTER

Output will be directed to the printer. Any arguments from the CP SPOOL PRT command may be used to modify the spooling control options. Standard CP SPOOL defaults will be used.

3. SET

REPORT

If REPORT is ON, the system types out on the terminal a diagnostic trace of all routines entered.

STATISTICS

If STATISTICS is ON, the system keeps statistics on time spent in processing. The statistics are printed, together with the current SEQUEL statement on the STATPRI LISTING file.

CASE

If case is set to LOWER, all quoted strings will be passed directly to SEQUEL without conversion to upper case. Thus, SET CASE LOWER will enable all literal character strings entered at the terminal to remain as they are typed in. SET CASE UPPER will cause character strings to be converted to upper case.

ROWNUM

If ROWNUM is ON, the rows generated by a SELECT command will be numbered for both folded and "window" output table format.

TITLE

This option provides an optional titling faculty for reports. When a title is specified it will be centered at the top of all output tables (except those output to a CRT). SET TITLE OFF stops the printing of the current title but allows the title to be retained in the system. SET TITLE ON causes the current title to be printed with the output tables as described above. Lines of a title may be specified by using SET TITLE '<title1>'/'<title2>'/... a maximum of 6 lines is allowed. Each line must be enclosed in single quotes and separated by slashes. A single quote may be used in the text of a title by entering it as two single quotes.

4. QUERY

Displays the options which are currently in effect for the OUTPUT and SET commands.

5. CP

Allows CP commands to be executed in the TRANSACT environment.

6. CMS

Allows the following subset of CMS commands to be executed in the TRANSACT environment:

ACCESS	LISTFILE	RENAME
CP	PRINT	RETURN
DISK	PUNCH	SET
ERASE	QUERY	STATE
EXEC	READCARD	TYPE

If CMS commands other than those listed are issued, an error message will be typed and the command will be rejected.

7. See SEQUEL BNF for valid queries.

8. QUIT

Ends the session.

NOTES:

1) All commands must be followed by a semicolon (nested

queries require more than one semicolon).

2) Transactions may require more than one input line. A transaction may be deleted by typing (%). If you wish to use (%) as a percent sign, enter (%%).

3) TRANSACT senses the end of a transaction if the last character entered on a line is a semicolon. Therefore, to avoid premature termination by TRANSACT of transaction input, make sure that a semicolon within a command (e.g. a COMPUTE clause) is not the last character entered on a line.

4) When TERMINAL LINESIZE is changed in TRANSACT, CP TERMINAL LINESIZE is also changed.

5) To enter the single-user TRANSACT environment from CMS type:

transact

and wait for the READY; message.

6) If you want to use TRANSACT to communicate with a disconnected Transaction Virtual Machine with an id of TVM, for example, type:

transact tvm

and wait for the READY; message.

Appendix J

TRANSACT RUN Command Macro Processor Proposal

DRAFT COPY

This publication is intended for those GMIS users who want to use the TRANSACT RUN facilities. It includes information on writing a RUN procedure, using the TRANSACT RUN facilities and RUN control statements, and building RUN procedures. The reader should have an understanding of elementary programming techniques such as branches, loops and loop control as well as an understanding of TRANSACT operating procedures, commands, and the CMS editor.

INTRODUCTION

The TRANSACT RUN facilities permit a user to define new TRANSACT commands that are combinations of existing TRANSACT and SEQUEL commands. The new commands, called RUN procedures, are usually created using the CMS Editor.

When a RUN procedure is invoked, it represents a sequence of commands that are executed according to the logic control statements defined in the RUN procedure. The TRANSACT user can create simple RUN procedures that execute several frequently used queries, or he can devise complex RUN procedures that etst several logical conditions before deciding whether or not to execute a query. The logical capabilities in the RUN processor are controlled with statements similar to the IF/THEN, GOTO, DO and LOOP statements familiar to high level language users.

A run procedure is created by placing a selected sequence of commands in a RUN file. A RUN file can have any valid CMS filename, filetype and filemode, but it is recommended that the filetype be TRANSACT so as to avoid confusion. RUN files are made up of varying length records up to 130 characters long Each record consists of one TRANSACT/SEQUEL command or RUN control statement.

Although RUN files are usually created by using the CMS Editor, they can also be created by reading a card file from the users virtual reader or by a user program.

INVOKING A RUN PROCEDURE

To invoke a RUN procedure, the user simply enters the TRANSACT command RUN, followed by the filename, filetype and filemode of the RUN file wanted, and optionally, a list of arguments. A RUN procedure can be invoked just by entering the filename if the filetype and filemode are 'TRANSACT' and 'A1' respectively. These are the default values taken for the RUN file.

When a RUN file is invoked, the RUN interpreter controls the execution of the procedure, substituting values for RUN variables where required, and passing control to TRANSACT for the execution of TRANSACT commands.

The RUN interpreter can manipulate parameter lists, thus allowing the user to pass arguments to the RUN procedure when it is invoked. Before a command in the RUN file is executed, each variable in it is temporarily replaced by the correspong argument from the parameter list that was specified when the RUN procedure was invoked. Use of these

variable arguments thus permits great flexibility in command execution within the RUN procedure.

The contents introduced in the preceding paragraphs are discussed in greater detail later in this document. At this point, however, the user can see that the RUN facilities provide him with a powerful tool that he can use to develop his own command language or set of operating procedures.

USING THE TRANSACT RUN FACILITIES

This section describes the three major parts of the TRANSACT RUN facility:

1. The RUN command which initiates TRANSACT execution of a RUN file.

2. The RUN files, which contain sequences of TRANSACT commands and RUN control statements.

3. The RUN interpreter, which analyses each statement in a RUN file before TRANSACT executes the procedure. Each of these items is described in greater detail under a separate heading.

RUN FILES

A RUN file is a CMS data file that can contain TRANSACT commands and RUN control statements.

RUN files can be created with the CMS editor, by punching cards or by a user program. The file should be of varying record length with a maximum record length not greater than 130.

RUN files consist of two types of statements: executable and nonexecutable. Each type is discussed below.

NONEXECUTABLE STATEMENTS

A nonexecutable statement in a RUN file is one that begins with an asterisk (*) and may or may not contain text. These statements are for use as comment statements and are ignored during RUN interpretation and processing.

EXECUTABLE STATEMENTS

An executable statement in a RUN file is any statement that does not begin with an asterisk. These statements consist of data items which are strings of contiguous nonblank characters separated by blanks. Three classes of executable statements are recognized by the RUN interpreter:

1. Null statements.
2. TRANSACT commands.
3. Control statements.

Each of these statement classes is discussed under a separate heading. In addition, a section on labels and comments is also given.

Null Statements

A null statement is an executable statement in which the number of data items is zero. A blank line is a null statement.

TRANSACT Commands

The RUN interpreter considers an executable statement as a TRANSACT command if the first data item does not start with an ampersand, an asterisk or a hyphen. Data item replacements are made on the statement and then TRANSACT executes the command immediately.

Any valid TRANSACT command may be included in a RUN file. Another RUN procedure may be invoked by simply entering another RUN command and setting the recursion nesting option to be described later.

Control Statements

An executable statement is a control statement if the first data item is a RUN control word. Examples of control words are:

```
&GOTO
&EXIT
&IF
```

Control satements begin with a control word, which is usually followed by a list of data items and, in some cases, by additional lines of data. Control statements provide the means by which the user can control the execution of his RUN procedure. The &IF control word, for example can establish a conditional test and a branch (&GOTO) can be taken if the condition is met.

Labels and Comments

A label in a RUN procedure begins with a hyphen (dash), and contains up to fifteen additional alphanumeric characters. A label can be placed in front of a TRANSACT command or RUN control statement. A label is often the object of a branching control statement, such as &GOTO or &LOOP. When searching for a label, the RUN interpreter examines only the first word on a line.

A comment in a RUN procedure begins with an asterisk (star). the remainder of the line may contain any combination of valid characters desired. A comment is included in the count of the number of lines read in from a file for execution (for use in references to line numbers or references to the scope of an &LOOP), but the text is ignored. See the section describing the &** control statement for alternate methods of entering comments.

THE RUN COMMAND

The RUN command gives one the ability to execute one or more TRANSACT commands or RUN control commands contained in a specified file, by issuing a single command. The format of the RUN command is:

```
+-------------------------------------------------------------+
|   RUN     |   FN  FT  FM  [NEST/NONEST]     (args...)     |
+-------------------------------------------------------------+
```

where:

FN
: is the filename of a file containing one or more TRANSACT commands to be executed.

FT
: is the filetype of that file. It can be any valid CMS filetype but if it is left out, it is assumed to be "TRANSACT".

FM
: is the filemode of the file. Again, it can be any valid CMS filemode but if it is left out, it is assumed to be "A1".

NEST/NONEST
: indicates how titles and other features are to be handled for recursive commands. That is, those RUN files which invoke other RUN files.

args
: are the arguments to replace the numeric variables in the RUN file specified. Within a RUN file, up to thirty symbolic variables may be used (each one indicated by an ampersand (&) followed by an integer ranging from one to thirty) to indicate values which are to be replaced when the RUN file is executed. The arguments are assigned to symbolic variables in the order in which they appear in the argument list. For example, each time an &1 appears in a RUN line, the first argument specified with the RUN command temporarily replaces the &1, the second argument specified within the RUN command replaces &2, and so on, to argument N of the RUN command.

  If the percent sign is used in place of an argument, the corresponding variable (&N) is ignored in all the commands which refer to that variable. If the specified RUN file contains more variables than

arguments given in the RUN command, the higher numbered variables are assumed to be missing, and are ignored when the commands are executed.

RUN CONTROL STATEMENTS

An executable statement is a control statement if the first data item is a RUN control word (the only exception are those control statements preceeded by labels). Control statements begin with a control word which is usually followed by a list of data items. Control statements provide the means by which the user can control the execution of his RUN procedure. Each RUN control statement is described below. (Note that all can be truncated down to a minimum length of 2 characters. Truncations starts from the rightmost character in the control verb.)

&ARGS CONTROL STATEMENT

```
| &ARGS        ( [arg1 [arg2 ...]] )                |
```

Redefines the arguments &1, &2, ... with the value of 'arg1', 'arg2' ...

NOTES:

List of arguments enclosed in parentheses, separated by blanks Strings with embedded blanks can be enclosed with single quotes. Quctes within a quoted string are represented by two adjacent single quotes. In a normal string however, they are represented by themselves. A per cent sign (%) is used to indicate a null value for a given argument in a list.

&BEGTYPE CONTROL STATEMENT

```
| &BEGTYPE                                           |
|----------------------------------------------------|
|   line1                                            |
|   line2                                            |
|   line3                                            |
|     .                                              |
|     .                                              |
|     .                                              |
|   linen                                            |
|----------------------------------------------------|
|   &END                                             |
```

Displays 'line1', 'line2', ... at the terminal. No argument replacement is perormed.

NOTES

&BEGTYPE must have a matching &END control statement for delimiting lines to be printed.

&CONTINUE CONTROL STATEMENT

```
┌──────────────────────────────────────────────────────┐
│  &CONTINUE                                           │
└──────────────────────────────────────────────────────┘
```

Provides a branch address for the &GOTO control statement and conditional braching statements.

&RETURN CONTROL STATEMENT

```
┌──────────────────────────────────────────────────────┐
│                  ┌           ┐                       │
│  &RETURN         │return_code│                       │
│                  │     0     │                       │
│                  └           ┘                       │
└──────────────────────────────────────────────────────┘
```

Exits from RUN file with the given return code.

NOTES

Return code must be between 0 and 9. If larger code is given, only the first digit of that number is used. If an invalid code is giver , or one is missing, 0 is assumed.

&ERROR CONTROL STATEMENT

```
+-------------------------------------------------------------+
|                                                             |
|  &ERROR      | executable statement |                       |
|              | blank line           |                       |
|                                                             |
+-------------------------------------------------------------+
```

Specifies a given action to be taken when an error occurs during execution. Entering &ERROR followed by a blank line resets the action to the normal system response. The normal response is tc print out the string being processed at the time of the error and then request a replacement line from the console.

&GOTO CONTROL STATEMENT

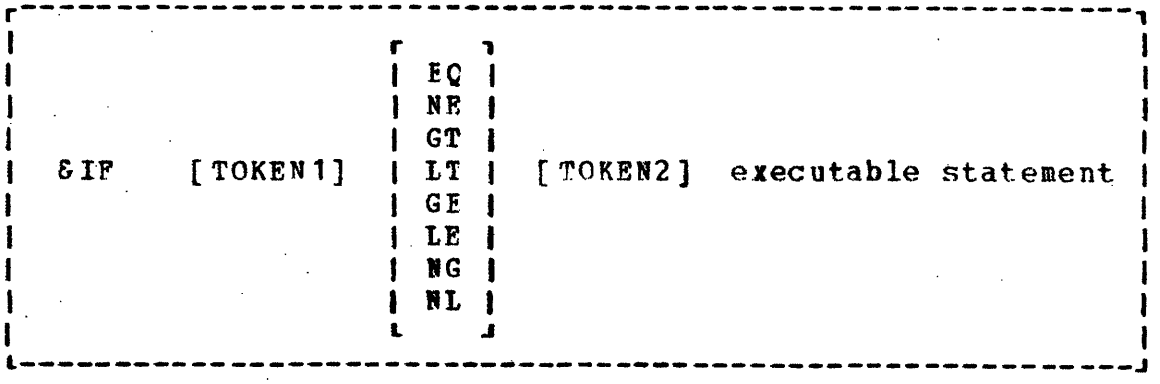
```
+-------------------------------------------------------------+
|                                                             |
|              | TOP         |                              |
|  &GOTO       | line_number |                              |
|              | label       |                              |
|                                                             |
+-------------------------------------------------------------+
```

Transfers control tc the top of the RUN file, to the given line number or tc the line starting with '-label'.

NOTES

The label specified should not have a leading hyphen, only the statement label in the text need have one to indicate that it is a statement label. If a transfer point is not found, a message is displayed and execution continues with the statement directly after the &GOTO just processed.

&IF CONTROL STATEMENT



Executes the 'executable statement' if the condition is satisfied.

NOTES

To compare for a null string, enter %% (double per cent sign) as the token tc be used for the comparison. Nesting of &IF statements is allowed so long as the entire statement is contained on one 130 character line.

&LOOP CONTROL STATEMENT

Loops through the fcllcwing n lines, or down to but not including the line indicated by '-label', for m times.

NOTES

The 'label' specified should not start with a hyphen as only the actual statement label needs one. Nested loop statements are not allowed. The statements passed over by the execution of an &SKIP control statement are not included in the ccunt for the number of statements executed in the loop.

WARNING Be careful that the scope of the statements inside a loop does not extend beyond the limits of the loop itself. (i.e. an &BEGTYPE control statement inside a loop with its associated &END

control statement outside the end of the loop.)

&READ CONTROL STATEMENT

```
+-------------------------------------------------------+
|  &READ                                                |
+-------------------------------------------------------+
```

Read the next line from the terminal and treats it as if it had been contained in the RUN file.

&QUIT CONTROL STATEMENT

```
+-------------------------------------------------------+
|  &QUIT                                                |
+-------------------------------------------------------+
```

Exits from the RUN file with a return code of 0. (i.e. Same as the statement - &RETURN 0)

&SKIP CONTROL STAEMENT

```
+-------------------------------------------------------+
|            +   +                                     |
|  &SKIP     | n |                                      |
|            | 1 |                                      |
|            +   +                                     |
+-------------------------------------------------------+
```

Skips over the next n lines in the RUN file. If no value is given, 1 is assumed for n.

&SPACE CONTROL STATEMENT

Types n blank lines at the terminal. Default value for n is 1.

&TYPE CONTROL STATEMENT

Prints the next n lines from the RUN file at the terminal. Default value for n is 1.

NOTES

Unlike the &BEGTYPE control statement, the &TYPE control statement does perform argument substitution before printing out a given line. It can therefore be used to check for correct redefinition of arguments when an &ARGS control statement is executed.

&** CONTROL STATEMENT

```
| &**      [ text ]                                                |
```

Indicates that the text on this line is to be ignored as a comment.

NOTES

Comments are also indicated by a '*' in column 1.

SPECIAL STATEMENT SYNTAX

'*' (in column 1) - Indicates that the present line is a comment and is to be ignored.

'-' (in column 1) - Indicates that this line conatins a statement label and the first token is therefore to be truncated from the rest of the line before processing.

WRITING A RUN PROCEDURE

Once a RUN procedure is designed, it can be entered into the VM/370 system in two primary ways.

1. By punching cards, which are then read via the real system card reader and the user's virtual reader.
2. By using the CMS Editor to enter input lines into a CMS file. The Editor normally truncates all input lines at 80 characters. This truncation can be avoided by specifying the LRECL option of the EDIT command with a record length of up to 130 characters. The Editor also normally assumes a file to have fixed length records. This can also be changed by specifying the RECFM option of the EDIT command to be varying.

   For example:

   ```
   EDIT filename TRANSACT (LRECL 130, RECFM V)
   ```

RUN files can also be created by a user program. Regradless of which method is used, the format of the entered statements is basically the same. Only one TRANSACT command or RUN control statement may be entered per card or card image. TRANSACT commands must be in the same format as they would be if entered from a terminal.

To use the CMS Editor to create a RUN file, simply enter the command:

```
EDIT filename TRANSACT (LRECL 130, RECFM V)
```

where the filename is any valid CMS filename, and the filetype of TRANSACT can be different. If the RUN file specified in the EDIT command is a new file, the message:

```
NEW FILE:
EDIT:
```

types out at the terminal. The user can then type in the INPUT subcommand and start entering input lines as soon as the Editor replies as follows:

```
input
INPUT:
( Begin entering input lines. )
```

Each input line is ended by pressing the return key. When input is complete, return to the EDIT mode by pressing the Return key again. If the file needs no corrections, simply type in the FILE subcommand. The data is stored and control is returned to the CMS environment. A RUN file created this way can be executed by typing in "RUN filename." Its contents can be examined either by typing it at the terminal using the CMS TYPE command, or by printing it on the system printer using the CMS PRINT command.

The preceding description of the CMS Editor identifies only a few of the Editor functions that may be useful in creating and maintaining RUN files. For a more complete discussion of the CMS Editor, refer to the VM/370: EDIT guide.

Appendix K

Proposal for TRANSACT Transaction Editor

This facility will permit a user of TRANSACT to edit any transaction that is typed in before actually executing it, and will allow that transaction to be executed and edited any number of times without need for reentry.

Conventions

Each command entered in TRANSACT will be saved just as it was entered, in multiple line format. The TRANSACT EDIT command will allow that command to be modified in a manner that parallels the CMS Editor (e.g. use of current line pointer). Until the TRANSACT editor subcommand QUIT is entered, TRANSACT will remain in EDIT mode, and the EDIT subcommand GO must be used to execute a transaction. When the TRANSACT user is not in EDIT mode, each transaction entered is executed in the normal manner, and if the first line entered after a READY; message is not EDIT;, then the buffer for the current transaction is cleared to accept a new transaction.

New TRANSACT Command

EDIT;

Puts the TRANSACT user into TRANSACT EDIT mode. Any (TRANSACT) EDIT subcommand may now be used to modify or input the current transaction. If the current transaction is empty, then the editor automatically enters INPUT mode.

TRANSACT EDIT Subcommands

Input

Puts the editor into INPUT mode. Successive lines are added or inserted into the current transaction in a manner similar to the CMS Editor. Entering a null line returns the editor to EDIT mode.

GO

Executes the current transaction. After the

transaction has been executed, TRANSACT returns to EDIT mode with the current line pointer (CLP) pointing to the top of the transaction. The transaction is not erased.

Type [n|*|1]

Types n lines of the curent transaction, starting with the line indicated by the CLP. If * is used, all lines from the CLP to the end of the transaction are typed.

Next [n|1]

Moves the CLP down n lines.

Up [n|1]

Moves the CLP up n lines.

DELete [n|*|1]

Deletes n lines of the current transaction, starting with the line indicated by the CLP. If * is used, then all lines from the CLP tc the end of the transaction are deleted.

DROP

The entire current transaction is deleted. This is equivalent to a TOP subcommand followed by a DELETE subcommand.

Top

Sets the CLP tc the top line of the current transaction.

Bottom

Sets the CIP to the last line of the current transaction.

Change /string1/string2/

Changes the first occurence of string1 to string2 on the line indicated by the CLP.

Quit

Leaves TRANSACT EDIT mode. Normal TRANSACT command processing continues.

NOTE: The EDIT; command is not recognized by the RUN command macro processor.

Appendix L

Using the DBDESCR Command

The DBDESCR command is used to get a general description of each table in a database. The user selects one of two options for the amount of information s/he wants printed:

(a) a column description ("DESCRIBE TABLE...") as well as a row count for each table,

or

(b) only a row count for each table.

USE

The user types DBDESCR followed by one of the following three things:

TERM
PRINTER
(nothing)

This directs the printed output either to the user's terminal, or to the printer; if nothing is typed after DBDESCR the program assumes that the output goes to the terminal. For example, the user might type

DBDESCR PRINTER

which invokes the program and sends the output to the printer.

The program will respond with

DO YOU WANT THE COLUMN DESCRIPTION FOR EACH TABLE? REPLY Y/N

The user now enters 'Y' if s/he wishes the column descriptions printed for each table (along with the row count), or 'N' if s/he wishes only to have the row count for each table. The program now prints out the database description.

Appendix M

Bulkloading of a Database (LTDB)

The program LTDBB loads a database from a CMS file, using free-format loader commands and user-specified format for the data records. The CMS file may have an arbitrary filename, filetype, and filemode. The program is in the form of a stand-alone module; that is, it cannot be invoked while using TRANSACT.

The loader can be used to:
(a) Add data to an existing table
(b) Create a new table
(c) Create a new domain.

One CMS file may contain a mix of all three loader commands.

When loading a SEQUEL table, the loader checks the data being entered against any assertions that were in effect at the time the loader was invoked (assertions cannot be set using the loader, but may be set using TRANSACT). When loading a table, the loader only adds data to the table, and does not affect data already present in the table.

The loader knows the input file as COMFILE in the program. Hence, the person using the loader to load from his or her input file called, for example, "INPUTDATA INPUTTYPE", on the A-disk, must link "COMFILE" to "INPUTDATA" by means of a FILEDEF:

```
FILEDEF COMFILE DISK INPUTDATA INPUTTYPE A1 (PERM)
```

(It is not necessary that the input file be on the A-disk.)

An easy way to invoke the loader without entering the FILEDEF is to use the EXEC command LOADDBB as follows:

```
LOADDBB  fn ft fm
```

This EXEC will issue the FILEDEF for the arbitrary file "fn ft fm", and will then invoke LTDBB automatically.

There is a default assumption that all character string data that is to be added to a table should be trimmed of any leading blanks; this is usually desireable. If the user does not want the leading blanks removed, eg. to be able to line up columns in the output of a SELECT command, he or she must use the 'NOLTRIM' option when invoking the loader; the user enters, instead of teh above command, a command of the form:

```
LOADDBB NOLTRIM fn ft fm
```

The INPUTDATA file is a regular CMS file of 80-character records (RECFM F). The loader commands consist of one or more records in the INPUTDATA file. Each command control card is read in a free format - each argument must be separated by one or more blanks. Note that this is not true of the data records themselves - these must be in a format specified in the $LOADTAB card. There are five major commands:

1. Define Domain Command

   Format: $DEFDOM <domain name> <domain type>

   <domain type> can be NUM (for numeric) or CHAR (for character).

   Each command creates one domain.

2. Define Table Command

   This command creates a table; it defines the table name, and for each column in the table, it identifies both the column name and domain name.

   Format:
   $DEFTAB <table name> <column name> <domain name> ...

   The order of the columns (left to right) in the table is the same as the order of the <column name> fields in the format description. The Define Table command must immediately be followed by a Define Primary Key command.

3. Define Primary Key Command

   This command must immediately follow a Define Table command. The format is:

   $PRIKEY <column name> ... $ENDKEY

   Note that though it is not necessary for a table to have a primary key, it is necessary to include a Define Primary Key command immediately after a Define Table command in INPUTDATA. If there is no primary key, there must be at least 1 space between $PRIKEY and $ENDKEY.

4. Load Table Command

   This command tells the system what table is being loaded, and in what format the data to be loaded is

found; the data--in that format--follow the command. Note that the table indicated in the command must have been previously created (either by a $DEFTAB loader command or a SEQUEL CREATE TABLE command).

The command has two parts: the format description and the data cards (records).

FORMAT DESCRIPTION

$LOADTAB <table name> <column name> <1st card #> <1st position> <last card#> <last position> ... $ENDCOL

Each column to be loaded has a group of five tokens associated with it in the format description:

column name
1st card # -- the index (relative to the next data card the loader will process) of the first card with data to be loaded into this column (this is normally 1--that is, the loader starts reading data off the first data card it finds--ie the next one).
1st position -- the leftmost card column (in the <1st card #>th card) where the data to be inserted into this column is to be found.
last card # -- the index, relative to the next data card (that is, the same data card to which <1st card #> is relative) of the last card where data for this column is located
last position -- the rightmost card column on this last card where data for this column is located.

The effect of each set of 5 tokens is that the loader inserts into the column <column name> all the data between <1st position> on <1st card> and <last position> on <last card>.

the order of the columns need not be the same as the left-to-right order that the table maintains; any order is acceptable, as long as the data are in the same order.

If some columns in the table being loaded do not appear in the format description the system assumes a default value of zero for that column. This is a potential problem since some columns have underlying domains of type CHAR. So, it is advised that all columns in the table have an

entry in the format description.

The records are processed one by one in the order as they appear in INPUTDATA. If a column name appears more than once in the format description, an error condition will result and the program will terminate.

DATA RECORDS

Data records are 80 byte card image records which follow their respective format description. The format of a data record is completely determined by the format description. A set of data records has the data for one row of a table (one tuple). Since one row of a table may require more than a single 80-byte record of the input file, the loader allows each input row of a table to extend over n data records. These data records are assigned relative numbers 1, 2, ...,n. The relative numbers are used in the <1st card#> and <last card #> fields of the format description to indicate the starting and ending points of a given column. It is therefore possible to have the data for a single column extend over more than one record.

If the data for a column, after being read from data card(s), contains nothing but blanks, the loader assumes a default of 0 for columns of data type NUM, and UNKNOWN for columns of data type CHAR.

End-Load Marker:

The last data card must be followed by

$ENDLOAD

5. End-Of-Input Command

The last record of INPUTDATA must be an end-of-input card whose format is:

$ENDINP

EXAMPLE:

```
$DEFDOM MODELDOM CHAR
$DEFDOM VOLDUM NUM
$DEFDOM MPGDOM NUM
$DEFDOM DATEDOM NUM
$DEFTAB CARSALES
     MODEL MODELDOM
     DATE  DATEDOM
     VOLUME VOLDOM
     MPG MPGDOM
$PRIKEY MODEL DATE $ENDKEY
$LOADTAB CARSALES
MODEL  1  1  1  15
DATE   1  20 1  23
VOLUME 1  28 1  34
MPG    1  17 1  19
$ENDCOL
CHEVROLET          1247401      33108
CORVETTE           1547401       2078
CHEVELLE           1797401      21175
CHEVY NOVA         1877401      21464
SPORTVAN           1527401       1370
MONTE CARLO        1497401      15668
CAMARO             1797401       8787
VEGA               3027401      38455
PONTIAC            1387401      10170
GRAND PRIX         1037401       4042
FIREBIRD           1797401       3666
VENTURA            1217401       4890
OLDSMOBILE         1107401      10533
$ENDLOAD
$ENDINP
```

Appendix N

Creating Backup Dumps of SEQUEL Databases

A need exists for creating backup dumps of SEQUEL data bases. A UFI has been written which fills this requirement. It is invoked by entering

```
SEQDUMP  filename
```

This will create at least one, and sometimes two files. The file which will always be created has the file name filename and filetype SEQDUMP. This file will be in bulk loader format and may be used to reload a reinitialized database. Using the distributed EXEC file for execution of the dump function, this file must be unpacked before it can be used by the bulk loader. This is accomplished by issuing the following command:

```
COPY filename SEQDUMP (UNPACK
```

Because each character domain is output into this file with a width equal to the maximum width for the domain, tuples which originally were contained in 80 characters will no longer be so contained. Hence to reload the database, the mulitple card version of the bulk loader must be used to reload the database.

The second file which may be created, depending on the contents of the database, will have a file name of filename and a file type of SEQUEL. It is in a format suitable for use as a SEQUEL query file using a UFI which allows query input from files. It will contain SEQUEL queries which will reestablish any inversions which were present in the original database. This file is necessary because inversions cannot be entered through the bulk loader. If there are no inversions in the database, this file will not be created.

If the disk space occupied by the file with filetype SEQDUMP becomes oppressive, by editing the file SEQDUMP EXEC, this file may be put on tape. This is done by modifying the FILEDEF for ddname DUMP to tape. If this is done, the COPY command which packs the SEQDUMP file should be removed from SEQDUMP EXEC and the FIIEDEF for ddname COMFILE in the bulk loader EXEC file should be modified for tape input to it.

Appendix O

Saving and Restcring a SEQUEL Database

The SEQBACK and SEQFEST functions are used to save and restore a SEQUEL database, respectively. They are very fast, taking only about 1 sec per cylinder; they use the DDR command to copy each cylinder of the database (including the non-335 disks) ontc tape (bit-by-bit), or to restore them from the tape in like manner.

USE:

SEQBACK nn -- saves a core image of the database

SEQREST nn -- reloads the core image

where nn is the number cf cylinders on the 335 disk (eg 20, 10).

ENVIRONMENT:

It is assumed that for either operation a tape is mounted at virtual address 181 and attached; for SEQBACK the tape is assumed to be positioned so that it can be properly written onto, and for SEQREST it is assumed that the tape is positioned at the beginning of the dump of the core image.

Appendix P

Saving and Restoring a Database

Database "saves", using SEQBACK, can be managed with a single tape, using the two (EXEC) routines:

DBSAVE
and DBSCAN

with the help of a private Database Log. The routines assume that the tape has already been mounted at virtual address 181 (though not necessarily rewound).

The basic organization of the tape is as follows: each database save is stored with a "stamp-file" inserted directly in front of it, which identifies the time and date of the "save", the number of cylinders in the 335 area involved, and the tables that were saved. The files on the tape alternate: stamp-file,save ---- stamp-file,save ---- stamp-file,save ---- etc. A new stamp-file/save pair is always put on the tape immediately after the most recent one (because it is assumed that they never get erased).

Saving a Database

If the tape is a clean tape (ie does not have any previous database saves on it), the user should simply rewind the tape, and invoke DBSAVE (see below).

If there are already saves on the tape, the overall strategy is to find the last save on the tape, using DBSCAN in conjuntion with the Database Log, then to use DBSAVE to store the database.

To start, the user invokes DBSCAN (no arguments), which will respond with

COMMAND?

The user should now type

FIRST

and DBSAVE will find the stamp-file for the first save on the tape, and print its name, eg.

DB_11/8 NEEMISA A1

This indicates to the user that the database save that

follows was from NEEMISA account, and was saved on November 8.

(The routine will now print out "COMMAND?" again, and wait for input.)

If the user wishes to know more about the save that follows, he enters

```
CONTENTS
```

The contents of the stamp-file will be loaded onto the A-disk, and the user may now examine them by entering

```
T fn ft fm
```

where 'fn ft fm' designates the stamp-file.

If this is not the last database save on the tape, which the user can verify by consulting his log, he may skip further down the list by entering either of the following two commands (in reponse to 'COMMAND?'):

```
NEXT
SKIP n
```

NEXT moves on to the next save on the tape, and prints out the name of its stamp-file (as above); SKIP moves over n saves (default is 1), and prints the file name of each corresponding stamp-file (so that the last stamp-file listed is the last one read, and the save corresponding to it is ready to be read in).

When the user has determined that he has found the last save on the tape, he should skip over the save itself by entering the command

```
LAST
```

which returns to CMS, and invoking the DBSAVE exec.

The DBSAVE exec will now save a copy of the database (preceded by the proper stamp-file). It takes two arguments: the first is the name of the account that the user is on, and the second is the number of cylinders in the 335-area that is to be saved (the same argument as to the SEQBACK exec). For example, the user might enter:

```
DBSAVE NEEMIS3 10
```

When the database has been saved, the user should note in his log the name of the stamp-file (to preserve the fact that it is now the last one on the tape) and may rewind it.

Restoring a Database

The overall strategy is to find the particular save that you wish to restore, using the DBSCAN exec, and then loading it by entering the 'SEQICAT' command.

To start, the user should invoke DBSCAN (no arguments), which will respond with

```
COMMAND?
```

Now the user types in "FIRST" and DBSCAN will print out the name of the first stamp-file on the tape, such as

```
DB_12/22 NEEMIS3 A1
```

This indicates to the user that the database save that follows was from NEEMIS3, and was saved on December 22.

If the user wishes more detailed information about this particular save, he should enter the command

```
CONTENTS
```

(in response to another 'COMMAND?'). The routine will load the stamp-file into the A-disk; the user may now examine it by entering

```
T fn ft fm
```

where 'fn ft fm' designates the stamp-file. A typical stamp-file would look like this:

```
20 CYLINDER DATABASE SAVED:
TIME: 22:13.05
DATE: 12/22/75

  5 TABLES:

INTEGRTY        DOMCAT          CATALOG         DOOBYDOO
BLOOPBLOOP
```

If the user does not wish to reload this particular save, he may scan down the tape to find the right save by using the two commands:

```
NEXT
SKIP n
```

NEXT moves on to the next save on the tape, and prints out the name of its stamp-file (as above); SKIP moves over n

(default is 1) saves, and prints the file name of each corresponding stamp-file (so that the last stamp-file listed is the last one read, and the save corresponding to it is ready to be read in).

Note: to simply list all the saves on a particular tape, the user may enter the 'FIRST' command followed by a 'SKIP 300' command (which will run off the end of the tape and flag an error, but a CMS REW command will fix that up).

When the user has found the save to be restored (ie. has just had the name of its stamp-file printed, or just looked at the contents of its stamp-file using the 'CONTENTS' command), he should enter the command

SEQLOAD nn

where nn is the number of cylinders on the 335-area to be restored (this is the same as the number of cylinders listed in the stamp-file itself). The database will be restored, and the user may rewind the tape. Summary of DBSCAN Commands:

In response to the question 'COMMAND?', the user may enter any of the following seven commands:

FIRST -- rewinds tape, finds first stamp-file on tape, and prints out its name

NEXT -- skips over one "save", and prints name of stamp-file corresponding to next save (if any)

SKIP n -- skips over the next n "saves" and prints the name of the stamp-file corresponding to the n-th save (if any)

CONTENTS--loads most recently found stamp-file onto A-disk, and responds by printing 'ENTER TYPE COMMAND (OR NULL LINE)'

SEQLOAD n --restores database pointed to by most recently found stamp-file; argument is number of cylinders on 335-disk

LAST -- skips over the next "save" (presumed to be the last one on the tape) and returns to CMS

QUIT--returns to CMS

In response to 'ENTER TYPE COMMAND (OR NULL LINE)', the user should:

--enter a type command (to type out a stamp-file that

has been loaded by the CONTENTS command), or

--enter a carriage return, to indicate no typeout.

Appendix O

Using SELDUMP

3/10/76

SELDUMP is a selective dump facility, that operates through the MultiUser Interface. It allows a user to dump selected table(s) from the DB Machine into an arbitrary file, in bulk loadable form (eg. LOADDBB). It optionally includes $DEFDOM and/or $DEFTAB records for the table(s).

USE:

SELDUMP db-accnt fn ft fm

db-accnt---the name of the database machine in which the table(s) to be dumped are located

fn ft fm---specify the file to receive the dump

Example:

SELDUMP NEEMIS ADIDATA LOADFILE A1

Routine will respond with: 'WHAT TABLE?'. Enter the name of the first table you wish to have dumped. Routine will then ask 'DO YOU WANT $DEFTABS, $DEFDOMS, BOTH OR NEITHER (T/D/B/N)?'. Enter one of the four letters 't', 'd', 'b' or 'n' to indicate your choice--this option will only remain in effect for this table.

The routine will now dump the table as directed. It will then ask you for the name of the next table, and for your $DEFTAB-etc. option, as above. You may dump as many tables as you want; when you are through, simply enter a carriage return (by itself) in response to the 'WHAT TABLE?' query.

Note: recall that you may not be able to dump extremely large tables with this routine, due to the limited size of the 340 area. If you need to, contact John Maglio or Bryan Mau.

Appendix B

Procedure for Analyzing Database Utilization

1. VMAPL

2.)LOAD 201 SEQUTIL (note: this is a LOAD, not a COPY)

3. PAGES_USED (this is the APL function that calculates utilization totals)

The terminal will print several "CONFILE" messages, of the form:

CON FILE nnnn TO xxxxxxxx COPY 01 NOHOLD

Then the program will output 5 numbers:

(a) total number of relation pages used
(b) total number of overflow pages used
(c) total number of data pages used
(d) total number of pages in use--the sum of (a), (b) and (c)
(e) the highest cylinder number in use

ROUGH ESTIMATE OF USAGE:

Type:

EXTENT(0;0) --using brackets instead of parantheses

Response will list total number of pages that are available to the database. Compare this number with the total given in (d) above--(d) will be some fraction of this number, representing percentage of total pages presently used up.

MORE DETAILED ANALYSIS:

The total number of cells allocated to each of the three types of pages (that is, relational, data and overflow) was determined when FORMSGT (eg SEQINIT) was last done on the database.

Each cell can accoumodate up to 48 pages, so that allocating 20 cells for data pages, for example, means that there cannot be more than 20 X 48 =960 data pages. Comparing this figure (that is, 48 X #_of_cells) to that given in (c) above will tell you how close you are to running out of storage for data pages (actually, to running out of storage for control blocks for data pages).

Likewise, the figures used with FORMSGT for the number of relation cells and overflow cells can be used, in conjunction with (a) and (b) above, to see how close you are to running out of those types of pages.