MIT/LCS/TM-53

MDC-PROGRAMMER: A MUDDLE-TO-DATALANGUAGE TRANSLATOR FOR INFORMATION RETRIEVAL

Safwan A. Bengelloun

October 1974

TM-53

MDC-PROGRAMMER:

A MUDDLE-TO-DATALANGUAGE TRANSLATOR FOR INFORMATION RETRIEVAL

Technical Memorandum

by

Safwan A. Bengelloun

June 1974

This research was supported by the Advanced Research Projects Agency of the Department of Defense, under ARPA Order No. 2095, which was monitored by Office of Naval Research Contract No. N00014-70-A-0362-0006.

PROJECT MAC

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

Cambridge Massachusetts 02139

ABSTRACT

This memo describes a practical application within the framework of the ARPA computer network of the philosophy that a fully developed computer network should appear as a virtual extension of the user's own software environment. The application involves the design and implementation of a software facility that will permit users at MIT's Dynamic Modeling System to consider the retrieval component of the Datacomputer (developed and run by the Computer Corporation of America) as an extension of the Muddle environment. This facility generates efficient Datalanguage retrieval code, handles inter-process control of the Datacomputer, and manages all the necessary network connections.

This memo reproduces a thesis of the same title submitted to the Department of Electrical Engineering, M.I.T., on May 20, 1974, in partial fulfillment of the requirements for the Degree of Bachelor of Science.

ACKNOWLEDGEMENTS

The author would like to thank Mr. Albert Vezza for encouragement and guidance in writing this memo. Thanks are also due to Hal Murray et al. at CCA for many discussions about the Datacomputer, and to the members of the Programming Technology Division for their support and suggestions.

TABLE OF CONTENTS

INTRODUCTION

The efficient use of computing resources has been historically one of the primary concerns of computer scientists. The efforts in this direction have run along two parallel paths, the one being the optimization of the sharing of hardware resources (computing power), and the other the increasingly more important area of the sharing of software resources. The former problem forced the trend away from dedicated systems and towards the multi-programmed, multi-access computer that is now in common use in virtually every application branch of data processing. But multi-programmed systems per se do not provide an appropriate framework within which to approach the software problem. In particular:

(a) Because the number of users that have access to a single multiprogrammed facility is limited in size, and because the demand for different applications is varied, the degree of specialization of any single facility remains restricted within well-defined boundaries.

(b) Because of this size limit, the access to the number of software resources is again limited. Computer manufacturers alleviate this problem by providing a set of "software packages" that are found to be useful in many applications. But this approach not only limits the user to the facilities of a single manufacturer, but often to the type of hardware available to him from that manufacturer.

(c) It is important that the sharing of resources not be encumbered with time delays (including those associated with distance). Users separated by great distances but having similar goals in common need to stay directly abreast with the developments in their particular areas of interest.

It was such considerations as the above that have led to the interest in and the development of computer networks. By linking together many computers so that the resources of each are readily available from every other site, the number of resources available to any individual user is immediately increased. By making each resource potentially available to this widely expanded set of users, the economies of scale begin to make specialization within reach. Finally the linkage of computers provides a communication medium among users that will allow the necessary interaction that forms the basis of software sharing.

A primitive computer network appears to the user as a collection of distinct computing resources (each with its own set of nuances) linked together through some communication medium. Use of foreign resources requires familiarity with foreign software procedures as well as with the mechanics of gaining access to these resources. By contrast, it is desirable that foreign resources appear as a logical extension of each user's computing environment. This memo describes the design and implementation of a software facility that will allow the users of the MIT Project MAC Dynamic Modeling System [1] to consider the retrieval component of the Computer Corporation of America Datacomputer [2, 3] as a logical extension of the DMS Muddle environment [4]. In particular it will allow the users at MIT-DMS to:

(a) ignore the management of network connections to the Datacomputer;

(b) ignore the management of inter-process control of the Datacomputer;

(c) form retrieval requests from any database at the Datacomputer using Muddle syntax, translating from that syntax into the Datalanguage code that the Datacomputer requires for retrieval.

This facility follows upon similar efforts in the past, but it is far superior to them in terms of generality, programming consistency, and adequacy of Datalanguage code generated.

This incorporation of the Datacomputer into the Muddle environment not only is desirable from the point of view of the user at MIT-DMS (since it accepts syntax that he is familar with), but also for the general network user: it is more convenient to program the Datacomputer with this facility; furthermore the addition of the "front-end" processing power of

Muddle to the Datacomputer provides to the network user a facility in its own right that is unique to the network.

OUTLINE

Chapter I describes the characteristics of the ARPA network and the type of problems that one faces when attempting to link resources within that network. Chapter II discusses the two resources that are to be linked together (Muddle and the Datacomputer) and the syntactic and semantic characteristics of each. The translation system is then discussed in full in the third chapter, with examples and scenarios included; comparison with previous systems is made. Finally Chapter IV concludes the memo and gives some suggestions for further development of such a facility.

I. THE ARPANET

Undoubtedly the most ambitious effort being made in the area of computer networks is with the Advanced Research Projects Agency Computer Communication Network (ARPANET). This geographically distributed network, originally linking some twenty computers when it first became operational in 1970, currently supports well over 40 such hosts spanning the distance from Hawaii, across continental United States, to London, England. Its chief characteristic (aside from its geographical distribution) is that it connects widely differing computer systems which share in common only the capability to support local multiprogramming.

The ARPANET interconnects a multitude of sites, each site consisting of a maximum of four computer systems and one communication computer (IMP). The IMPs of the network (each linked to a maximum of five others) provide the necessary standardization and in general are responsible for message traffic control across the network. An IMP also forms the interface between each of the computer systems and the network as a whole.

Standardization throughout the network is also provided by a series of layered protocols, i.e. a set of agreements between different processes as to the format through which they are to locate, synchronize, and exchange information. Currently the ARPANET supports three levels of protocols:

(a) bottom-level IMP-IMP and HOST-IMP protocols to manage the flow of information across the network (i.e. control, routing, etc.) [5, 6];

(b) a HOST-HOST protocol for the creation of virtual process-process connections between processes residing on different hosts [7];

(c) function-oriented process-process protocols to support specific tasks (such as the "Telnet protocol," [8] "file transfer protocol (FTP)," [9] and the "remote-job entry protocol" [10]).

It is the problems associated with the last set of protocols that concern us here. In particular, if remote facilities are ever to be viewed as a virtual extension of each host on the network, this set of protocols will have to be handled automatically by communicating processes at the two remote sites, with the user unaware of their existence. The implementation of such a facility, in which a program as opposed to a human user is directly manipulating a foreign service, brings with it its own particular problems. Specifically, for most time-shared computers, the set of responses generated by the system assume a human user at the other end who will use his own intelligence to act upon the responses he has received. Responses usually are coded in natural language (e.g. "xyz loading"), are often unsynchronized (e.g. "system going down"), and are often associated with the nuances of particular hosts (prompt characters, etc.).

The approach to the solution of this problem has been the adoption of what might be termed the user-server paradigm: the server site passively accepting commands from the user and type-coding its responses to inform the user of the state that it is currently in, to provide him with necessary informational messages that he may wish to act upon, and to inform him when errors have occurred (and if possible their severity so that appropriate action might be taken). The FTP protocol is an example of the application of such a paradigm, and we shall see that the Datacomputer, by using this paradigm, allows us to incorporate it as a virtual extension of a Muddle process.

II. THE TWO SYSTEMS

THE DATACOMPUTER

The Datacomputer is an information storage and retrieval service operated by the Computer Corporation of America to serve the general ARPANET community. Designed eventually to provide over a trillion bits of storage online, this developing system is planned to become one of the major stores of information for the network.

The Datacomputer is particularly suited for the management of highly structured information such as that found in transportation reservation schedules or weather databases, where the structure of the information is equally as important as its content. To the staggering amount of storage capacity is added the capability to rapidly retrieve subsets of the stored data according to either its structural organization or its content.

The Datacomputer is purely a storage device with no "front-end" processing capabilities (beyond the retrieval of information). This is a disadvantage, since interest in structured data often goes beyond the individual items and to the aggregate; we speak of "batting averages," "total days of sunshine," "per cent of time up," etc. Unless such values (i.e. average, total, per cent) are explicitly stored in the database, the user must rely on another processing facility in order to arrive at these. In short the Datacomputer lacks a front-end statistical package.

Information at the Datacomputer is stored in a hierarchical structure. Central to the understanding of this structure is the notion of a "container." Fig. 1 shows a typical container structure of a hypothetical database concerning MIT's Project MAC. Fig. 2 shows the actual realization of this description in a Datacomputer file.

```
MAC
------------------------------------------------------------
| PERSONNEL                                                |
| -------------------------------------------------------- |
| | AI                                                   | |
| | ---------------------------------------------------- | |
| | | STUDENTS                  FACS                   | | |
| | | ----------------------    ---------------------  | | |
| | | | STUDENT            |    | FAC               |  | | |
| | | | ------------------ |    | ----------------- |  | | |
| | | | | NAME    YEAR   | |    | | NAME    RANK  | |  | | |
| | | | | ------  ------ | |    | | ------  ------| |  | | |
| | | | | |    |  |    | | |    | | |    |  |    || |  | | |
| | | | | ------  ------ | |    | | ------  ------| |  | | |
| | | | |                | |    | |               | |  | | |
| | | | | STATE   SALARY | |    | | STATE   SALARY| |  | | |
| | | | | ------  ------ | |    | | ------  ------| |  | | |
| | | | | |    |  |    | | |    | | |    |  |    || |  | | |
| | | | | ------  ------ | |    | | ------  ------| |  | | |
| | | | ------------------ |    | ----------------- |  | | |
| | | ----------------------    ---------------------  | | |
| | ---------------------------------------------------- | |
| |                                                      | |
| | ML                                                   | |
| | ---------------------------------------------------- | |
| | | STUDENTS                  FACS                   | | |
| | | ----------------------    ---------------------  | | |
| | | | STUDENT            |    | FAC               |  | | |
| | | | ------------------ |    | ----------------- |  | | |
| | | | | NAME    YEAR   | |    | | NAME    RANK  | |  | | |
| | | | | ------  ------ | |    | | ------  ------| |  | | |
| | | | | |    |  |    | | |    | | |    |  |    || |  | | |
| | | | | ------  ------ | |    | | ------  ------| |  | | |
| | | | |                | |    | |               | |  | | |
| | | | | STATE   SALARY | |    | | STATE   SALARY| |  | | |
| | | | | ------  ------ | |    | | ------  ------| |  | | |
| | | | | |    |  |    | | |    | | |    |  |    || |  | | |
| | | | | ------  ------ | |    | | ------  ------| |  | | |
| | | | ------------------ |    | ----------------- |  | | |
| | | ----------------------    ---------------------  | | |
| | ---------------------------------------------------- | |
| -------------------------------------------------------- |
------------------------------------------------------------
```

FIGURE 1: A container structure

```
FILE MAC LIST
   PERSONNEL STRUCT
        AI STRUCT
           STUDENTS LIST (40)
                STUDENT STRUCT
                   NAME STR (10)
                   YEAR STR (2)
                   STATE STR (10)
                   SALARY STR (3)
                END
           FACS LIST (20)
                FAC STRUCT
                   NAME STR (10)
                   RANK STR (10)
                   STATE STR (10)
                   SALARY STR (3)
                END
        END
        ML STRUCT
           STUDENTS LIST (40)
                STUDENT STRUCT
                   NAME STR (10)
                   YEAR STR (2)
                   STATE STR (10)
                   SALARY STR (3)
                END
           FACS LIST (20)
                FAC STRUCT
                   NAME STR (10)
                   RANK STR (10)
                   STATE STR (10)
                   SALARY STR (3)
                END
        END
   END
```

FIGURE 2: A Datacomputer file description

In fig. 2 we note that the containers come in three types: STRing, STRUCTure, and LIST. An STR contains a fixed length string of ASCII characters (the length of the string of characters is specified inside the parentheses following the STR). Currently the Datacomputer will store only fixed length ASCII strings, but it will eventually support other types of data.

A LIST contains other containers of identical description. Thus a list description of the form:

```
HOSTS LIST
  HOST STR (3)
```

will contain a list of host numbers.

A STRUCT contains other containers but not necessarily of identical description. A STRUCT is useful in making logical connections between items in a database as well as providing a naming mechanism (the two concepts are of course interrelated).

All the above is a reflection of information organization of a file, not of the file itself. Conceptually we may think of the following file:

```
BALLPLAYERS LIST
   BALLPLAYER STRUCT
      NAME STR (15)
      BATTING-AVERAGE STR (3)
   END
```

as being a list of ballplayers and their associated batting averages. An entry in such a list (e.g. "   Babe Ruth320") would be referred to as a list member while each field in the entry (i.e. the name field (Babe Ruth) and the batting average field (320)) are referred to as list member elements.

The basic statement used for retrieval by content from the Datacomputer is the FOR statement. Logically speaking, the FOR statement is a universal quantifier over the list-members of a list:

$$\forall(X \in \mathrm{LISTX}),\ P(X)$$

where P(x) is a set of connected predicates that each list member must satisfy before being added to an output list. The predicates that the Datacomputer understands are EQ (equal or identity), NE (not equal or not identity), LT (less than), LE (less than or equal to), GT (greater than), and GE (greater than or equal to). The logical connectives that it recognizes are AND, OR, and NOT.

The FOR statement in reality is a bit more complicated than the above. Specifically, a user programming in Datalanguage must explicitly describe the format of the output list, which for efficiency should match the list-members that are output. Additionally a user may specify which STRs or STRUCTs within a list he wishes to see from each list-member that satisfies the given predicate. For example, we may wish to know the names of the ballplayers who have a batting average greater than 300, but we may not be interested in seeing the batting average itself; in totality the Datalanguage program for such a request would take the form:

```
CREATE OUTPUT PORT LIST
 NOM STR (15);

FOR OUTPUT.NOM,BALLPLAYERS.BALLPLAYER
   WITH BATTING-AVERAGE GT '300'
        NOM=NAME;
END;
```

The first two lines above would create an appropriate output port with only one field (the name field to be returned); note that the names that are given to the different containers need not match the names from the file that are to be retrieved. The last three lines effectively say: for every list-member of the ballplayers list with batting average element greater than 300, add the NAME element to the output list OUTPUT.

MUDDLE

The Muddle programming language is a direct outgrowth of LISP. It was designed by members of the MIT Artificial Intelligence Laboratory and the Programming Technology Division of MIT Project MAC as an environment within which PLANNER and Planner-like languages might run. Its chief advantages over LISP are more data types, more readable syntax, ease of extensibility, network interfacing primitives, and a base for advanced graphics work.

Muddle has been running at MIT-DMS for the past three years. During this period it has been continually augmented; this process continues as new areas of research are identified and explored.

Beyond the base features of Muddle, the DMS implementation provides access to a general dynamically-loadable "Muddle Library." This library consists of a set of functions and global data which users may access in building up more complex programs out of other users' previous work. Within this library are such items as a general context-free parser for BNF grammars and a graphics package for display consoles. A Muddle compiler is also implemented in order to speed the processing of debugged code.

III. THE TRANSLATION SYSTEM

The translation system (MDC-Programmer, or MDC for short) is capable of formulating retrieval requests for an arbitrary database at the Datacomputer. MDC must be made aware of the format of each database at the Datacomputer from which a user may want to retrieve data. This is done by creating a file containing a model (description) of the particular Datacomputer file's structure. This model is created and stored at the Dynamic Modeling System. A user who wishes to use the retrieval system need only identify to it the local DMS file where the database model is stored. The model encompasses, in addition to all the information found in a Datacomputer file description, a few other items which are used to provide a more convenient user interface. The process of creating the model file is relatively easy and straightforward, and it need only be done once for each new Datacomputer database from which we wish to retrieve information. Appendix 1 contains the format of such a model along with a sample model file.

Given that a Datacomputer file and its corresponding DMS model file exist, MDC-Programmer when loaded will connect to the Datacomputer (using the standard network Initial Connection Protocol [11]), perform the LOGIN for the current user, and open the appropriate file for reading. If successful in all three, MDC will build a series of functions bearing the simple names of all the containers in the Datacomputer file model.

The functions that are available to the user (beyond the primitives of Muddle and any functions that he may have defined or used from the library) are these:

(a) A set of functions bearing the simple names of all the containers. These functions are used to create the predicate that the output list members must satisfy.

(b) A series of functions that move a pointer inside the model of the database; the position of the pointer is crucial in determining what actions the set of functions in (a) are to perform. The functions that manipulate the pointer are shown here in muddle syntax:

```
<PTOP>  ;"Position the pointer at top of model."
<R>     ;"Move pointer one position to the right."
<L>     ;"Move pointer one position to the left."
<DR>    ;"Move pointer down one level to the right."
<DL>    ;"Move pointer down one level to the left."
<UR>    ;"Move pointer up one level to the right."
<UL>    ;"Move pointer up one level to the left."
```

(c) A REQUEST function that is used to specify the associated <u>LIST</u> member elements that are to be returned for each <u>LIST</u> member satisfying the predicate.

(d) An EXECUTE command that will return to the user a Muddle channel from which the information that has been retrieved is to be read. The EXECUTE command alone actually interacts with the Datacomputer. It starts a compilation process that generates

Datalanguage code according to the specifications made by the above sets of functions, transmits that code to the Datacomputer, and returns a channel on which the requested data is waiting to be read.

(e) A set of "convenience commands" which allow the user to enter different modes, specify a change of file from the one currently being processed, display the substructure that the pointer points to, display or suppress the set of Datacomputer control responses, etc.

It should be noted that the position of the pointer determines the actions to be taken by the various functions in the set (a) above. Application of one of the functions will send the MDC-Programmer through an exhaustive search of that part of the file model currently at the pointer. The search restricts the values of the data returned from each container bearing the name of the function to the values specified in the argument of the function. For example, the application of the function:

<STATE ("MASS")>

for a file with the description of that in figure 3 will have the following results depending on whether the pointer is at ptr1, ptr2, or ptr3:

(a) If at ptr1, this requires all the values of the field STATE within the list MAC to be "   MASS" (the padding is necessary because the data is stored in fixed length fields; MDC-Programmer will pad appropriately for the user). This would be used for example in a request such as "Which people at Project Mac come from Massachusetts?"

(b) If at ptr2, this requires all the values of the field STATE within the AI group to be "   MASS", as in "Which people from the AI group are from Massachusetts?"

(c) If at ptr3, this requires all the values of the field STATE for the students within the AI group to be Massachusetts.

```
FILE MAC LIST
   PERSONNEL STRUCT            <===ptr1
        AI STRUCT              <===ptr2
           STUDENTS LIST (40)  <===ptr3
                STUDENT STRUCT
                   NAME STR (10)
                   YEAR STR (2)
                   STATE STR (10)
                   SALARY STR (3)
                END
           FACS LIST (20)
                FAC STRUCT
                   NAME STR (10)
                   RANK STR (10)
                   STATE STR (10)
                   SALARY STR (3)
                END
        END
        ML STRUCT
           STUDENTS LIST (40)
                STUDENT STRUCT
                   NAME STR (10)
                   YEAR STR (2)
                   STATE STR (10)
                   SALARY STR (3)
                END
           FACS LIST (20)
                FAC STRUCT
                   NAME STR (10)
                   RANK STR (10)
                   STATE STR (10)
                   SALARY STR (3)
                END
        END
   END
```

FIGURE 3: A Datacomputer file description with pointers

The actions of these functions also depend on the type of container that the function name refers to (the containers being the ones unambiguously located by the above procedure). In the pointer examples, all the containers were STRs and the action taken was as described. However if the container turns out to be a STRUCT, the arguments of the function are passed (in order) to each container within the STRUCT. This procedure is recursive, allowing STRUCTs to be embedded within other STRUCTs. For the description in fig. 2, an application of the function

```
<STUDENT (("JOHN")("73"))>
```

is equivalent to the application of the functions

```
<NAME ("JOHN")>
<YEAR ("73")>
```

Note that any previous statement about other fields that place restrictions on the LIST members that are to be returned remain unaffected. If we wished to specify those students named John with salaries equal to 500 (irrespective of their YEAR or STATE), this would be done in the following manner by using a null list in the position of the YEAR and STATE fields:

```
<STUDENT (("JOHN")()()(500))>
```

If the function name refers to a LIST, the arguments are passed down again as with STRUCT, but this time to the one (there can be only one) container that the list encloses.

Strictly speaking, functions bearing the names of LISTs and STRUCTs are not necessary. However, they are important, because it is in terms of these containers (as well as the STRs) that the user thinks about his data; they provide an additional convenience of making one function call instead of several; and finally they allow the user to resolve simple name ambiguity without moving the position of the pointer.

The set of container name functions can also take arguments involving the predicates that the Datacomputer understands (in the above examples the implicit predicate was EQ), linked together by the connective 'OR'. Thus application of the following function is acceptable:

```
<STUDENT (("JOHN" OR "JACK")
          (LT "73" OR GE "75"))>
```

This would return those students named Jack or John who graduated before 73 or who will graduate after 74.

The REQUEST command is used to specify which elements of each LIST-member from the output list are to be returned. The format of the REQUEST command is as follows:

```
<REQUEST <name> <name> ...>
```

where <name> refers to the names of the containers that are to be returned. These simple names are resolved in terms of the description pointer as was done with the container name functions. If <name> refers unambiguously to an STR, the STR values are returned; if <name> refers unambiguously to a STRUCT, all elements within that STRUCT are returned; finally if <name> refers unambiguously to a LIST, all the elements of each list member are returned.

The EXECUTE command, applied simply as

<EXECUTE>

starts the actual retrieval process. This process is a two pass algorithm. The first pass builds an appropriate port description for output, and the second pass uses that description in the formulation of an appropriate Datalanguage request statement. Both passes are recursive, allowing file models to have arbitrary embedding of containers and thus capable of processing any file that could conceivably be stored at the Datacomputer (for version 0/9).

PROCESS-PROCESS COMMUNICATION

Communication with the Datacomputer is done through the user-server paradigm discussed earlier in this memo. The Datacomputer will sit passively, interpreting Datalanguage commands and sending out information to the user site specifying the actions that are being taken as a result of these commands. This set of information messages is sent out one line at a time. The first five characters of a response line are a message code intended to be used by a foreign program, while the remainder of the line is the natural language equivalent of this code for human interpretation.

The class of responses sent by the Datacomputer fall broadly into three areas:

(a) informational messages

(b) synchronization messages

(c) error messages

The first character of each response line determines the class within which each message falls. Error messages are further subcategorized according to severity, so that the user site may take appropriate action. Typical synchronization messages are "waiting for datalanguage input," "waiting for data," etc. Some informational messages are "adding node to table," "execution complete," etc. Error messages can be either compilation or execution errors. In the latter case, a series of messages will follow, describing the actions taken. These actions may vary from "temporary ports flushed" to "crashing user job." Each error message is then followed by a synchronization message which will allow the two processes to get back into step after having lost control through an unexpected event.

COMPARISON WITH PREVIOUS SYSTEMS

Two efforts along similar lines preceded this one. These were SURRET [12] written by the author, and SMART written by Hal Murray at Computer Corporation of America.

Every twenty minutes, a program at the Dynamic Modeling System wakes up to record the statuses of the different hosts across the network. The program collects this information and stores it in the SURVEY database at the Datacomputer. This information proved to be of interest to the general network community, so a task-specific Muddle-to-Datalanguage programmer was written for retrieving this information. It was the success and usefulness of this facility that brought about considerations for a general Muddle-to-Datalanguage programmer.

SMART was a prompt-response system for generating datalanguage code from the set of responses made by the user. There are two chief difficulties with the whole concept of a prompt-response system. One is that as the number of containers in a file tend to increase, the system becomes an increasingly more difficult interface for the user, as he may have to be prompted unnecessarily for a large number of fields. The second difficulty is that the prompt-response is not a consistent embedding of the system within a larger software environment; such an approach fragmentizes resources as opposed to unifying them into more powerful facilities.

SMART is not a fully general system. Specifically it will not handle any files with embedded lists within them. Its usefulness is thus restricted to a very small subset of the potentially rich information structures that the Datacomputer is capable of handling. In addition, making SMART intelligent enough to retrieve information from a new file description requires about three days of system programmer time. MDC-Programmer improves this performance in two respects. (1) The task of acquiring the capabilities to process a new file does not require the intervention of a systems programmer. It is simple enough so that any user who knows the file structure can create the model file. (2) This process for creating the file model takes on the order of minutes as opposed to days.

SCENARIO

A simple scenario of MDC-Programmer in use is outlined below. It is a transcript of an actual session made at the Dynamic Modeling System. In this session, retrieval of information is made from two files: PEOPLE as given in the example database in appendix 1, and SURVEY as given in figure 4. The former file was created for testing purposes whereas the latter is a "real" file which is accessed quite often to get information regarding the past performance of various hosts on the network. The underlined lines below represent those that have been typed in by the user; the indented lines have been included for explanatory purposes; the lines beginning with a period or semi-colon are responses from the Datacomputer; all other lines are what Muddle prints or returns as a result of function application.

```
LISTENING-AT-LEVEL 1 PROCESS 1

    Message from Muddle indicating it is listening for commands.
<FLOAD "SAB;M-DC">$
```

Above function will load MDC (the $ sign typed in by the user starts the evaluation by Muddle).

```
Muddle to Datalanguage Translator
Please type name of local file containing
Datacomputer file model:
"SAB;PEOPLE FILE"$
```

MDC greeting message followed by a string typed in by the user identifying the pathname of the file containing the model of the Datacomputer file PEOPLE.

```
;J150 21-05-74 0034:43  FCRUN: HERE WE GO
;J200 21-05-74 0034:43  RHRUN: READY FOR REQUEST
.I210 21-05-74 0034:43  LAGC: READING NEW DL BUFFER
```

Datacomputer greeting message; it is now ready to receive commands---first the user must be logged in.

```
LOGIN NAME PLEASE:
```

Typed out by MDC to prompt user for login name.

```
"MIT.DMS.SURVEY"$
;J209 21-05-74 0035:09  RHRUN: EXECUTION COMPLETE
;J200 21-05-74 0035:10  RHRUN: READY FOR REQUEST
.I210 21-05-74 0035:10  LAGC: READING NEW DL BUFFER
```

Login is completed. Note that the user is currently in UNSOAK mode, i.e. the responses of the Datacomputer are displayed at his console. He can vary this mode (to one in which the Datacomputer responses are not displayed) by application of the function SOAK.

```
;U000 21-05-74 0035:25  DHKD: ADDING PUNCTUATION
;J209 21-05-74 0035:28  RHRUN: EXECUTION COMPLETE
;J200 21-05-74 0035:28  RHRUN: READY FOR REQUEST
.I210 21-05-74 0035:28  LAGC: READING NEW DL BUFFER
"DONE"
```

The four Datacomputer lines are in response to the opening of the PEOPLE file; the open was successful; the last line was returned by Muddle indicating that loading of MDC has been completed.

```
<CONTEXT>$
```

CONTEXT is a function which will show the file model to the user below the point where his pointer is currently located. Immediately following loading, the pointer will always point to the top of the model. After printing the model, CONTEXT will then return the pathname of the pointer.

```
(PEOPLE
 LIST
 ()
 (PERSON
  STRUCT
  ()
  (NAME STR ())
  (ADDRESS STR ())
  (CITY STR ())
  (STATE STR ())
  (ZIP STR ())
  (DEPENDENTS
   LIST
   ()
   (DEPENDENT STRUCT () (NAME STR ()) (AGE STR ())))))
"PEOPLE"
```

```
<TERSE>$
"DONE"
```

Enters the user into terse mode; i.e. after applying the container name functions or the REQUEST function or the EXECUTE function, the model will not be printed but rather only the pointer pathname will be returned.

```
<REQUEST (NAME)>$
"PEOPLE"
```

Request is made for all names within the file; since the pointer is currently at the top of the model, all the names of the children as well as the adults will be returned.

```
<SET CH <EXECUTE>>$
;J209 21-05-74 0046:07  RHRUN: EXECUTION COMPLETE
;J200 21-05-74 0046:07  RHRUN: READY FOR REQUEST
.I210 21-05-74 0046:08  LAGC: READING NEW DL BUFFER
```

This set of Datacomputer responses is made in response to the creation of an output port. In reality the entire Datalanguage program has been to the Datacomputer, but resynchronization occurs at the next request. The following code was sent to the Datacomputer:

```
 "CREATE L2 TEMP PORT LIST
     ST1 STRUCT
        S2 STR (15)
        L1 LIST (0, 2), D='*'
```

```
            S1 STR(15)
        END ;"
```

followed by the request:

```
" FOR L2.ST1,PEOPLE.PERSON
      S2=NAME;
      FOR L1.S1,PEOPLE.PERSON.DEPENDENTS.DEPENDENT
            S1=NAME;
      END;
    END;"
```

and Muddle returns:

```
#CHANNEL [4 "READ" -1 -1 "NET" 0 5688 13893763 "NET" 4127 4118
23748359936 <ERROR END-OF-FILE!-ERRORS> 0 0 0 0 10 ""]
```

the channel returned by the EXECUTE command.

```
<FILECOPY .CH .OUTCHAN>$
```

FILECOPY is a Muddle function which copies data from one channel to another; in this case it is copying the data coming from the Datacomputer to the user console output channel, giving:

```
     BILL STORM                                 *
     ALICE FALL      JILL FALL                  *
   SCOTT SUMMER    MARY SUMMER                  *
 138
```

The above are the names returned; the "*" was specified in the generated Datalanguage code to separate the occurrences of inner list members. In this manner we know that Jill Fall is a dependent and that Mary Summer is a dependent. The 138 at the end is a count of the number of characters that have been received over the channel; it is returned by FILECOPY. The next request will be to ask for the address of Scott Summer. Here however things will be done in SOAK mode so that the Datacomputer responses will no longer appear.

```
<SOAK>$
"DONE"

<NAME ("SCOTT SUMMER")>$
"PEOPLE"

<REQUEST (STATE ADDRESS)>$
"PEOPLE"
```

```
<SET CH <EXECUTE>>$
#CHANNEL [4 "READ" -1 -1 "NET" 0 6512 13893763 "NET" 4127 4118
23748359936 <ERROR END-OF-FILE!-ERRORS> 0 0 0 0 10 ""]

<FILECOPY .CH .OUTCHAN>$
MA          9 BOW STREET
23
```

9 BOW STREET and MA were found as the address of Scott Summer in the order requested: first the state, then the street address. The character count is again returned by FILECOPY. Note that for these examples we did not need our pointer functions (primarily because of the simplicity of the file); performance of these functions is shown below.

```
<CVAL>$
"PEOPLE"
```

Clear all values; the file model now looks like it did when we first loaded MDC.

```
<DR>$
"PEOPLE.PERSON"

<DR>$
"PEOPLE.PERSON.NAME"

<R 5>$
"PEOPLE.PERSON.DEPENDENTS"

<CONTEXT>$
(DEPENDENTS
 LIST
 ()
 (DEPENDENT
  STRUC
  ()
  (NAME STR ())
  (AGE STR ())))
"PEOPLE.PERSON.DEPENDENTS"
```

```
(Y74Q1 LIST ()
 (LOGTRY STRUCT #FALSE()
  (DAY STR () #FALSE() 2 !"0 T T 1 31)
  (MONTH STR () #FALSE() 2 !"0 T T 1 12)
  (YEAR STR () #FALSE() 2 !"0 T T 73 74)
  (HRMIN STR () #FALSE() 4 !"0 T #FALSE() #FALSE() #FALSE())
  (HOST STR () #FALSE() 3 !"0 T T 0 6)
  (STATUS STR () #FALSE() 1 #FALSE() #FALSE() T 0 6)
  (SOC STR () #FALSE() 3 #FALSE() #FALSE() #FALSE() #FALSE()
#FALSE())
  (SCHED STR () #FALSE() 1 #FALSE() #FALSE() #FALSE() #FALSE()
#FALSE())
  (RESTIME STR () #FALSE() 3 !"0 T #FALSE() #FALSE() #FALSE())))
```

FIGURE 4: SURVEY database model

```
<NEWFILE "SAB;SURVEY FILE">$
```

A change from the file from which information is to be retrieved is done by the NEWFILE command; the argument to the command must be the pathname of the DMS file where the new Datacomputer file model is stored.

```
"Y74Q1"

<CONTEXT>$
(Y74Q1
 LIST
 ()
 (LOGTRY
  STRUCT
  ()
  (DAY STR ())
  (MONTH STR ())
  (YEAR STR ())
  (HRMIN STR ())
  (HOST STR ())
  (STATUS STR ())
  (SOC STR ())
  (SCHED STR ())
  (RESTIME STR ())))
"Y74Q1"
```

This survey file contains all the Survey information for the first quarter of 1974.

```
<HOST (31)>$
"Y74Q1"

<HRMIN (GT 500 AND LT 600)>$
"Y74Q1"

<MONTH (1)>$
"Y74Q1"

<DAY (LT 4)>$
"Y74Q1"

<REQUEST (LOGTRY)>$
"Y74Q1"

<SET CH <EXECUTE>>$
```

The request was for all the information on host 31 (CCA) for the time period of 5 o'clock to 6 o'clock from January 1 to January 3.

```
#CHANNEL [4 "READ" -1 -1 "NET" 0 7280 13893763 "NET" 4127 4118
23748359936 <ERROR END-OF-FILE!-ERRORS> 0 0 0 0 10 ""]

<FILECOPY .CH .OUTCHAN>$
010174051203150012031
010174053203150012034
010174055203150012031
020174051003150012032
020174053003150012038
020174055003150012036
030174051103150012025
030174053103140012000
030174055103150012029
189
```

The first six characters of each line are the date, followed by four for the time, three for the host number (031), one for the status (5=logger available), three for the socket (001), one for the schedule (2=unknown), and three for the response time in tenths of a second.

```
<CONTEXT>$
(Y74Q1
 LIST
 ()
 (LOGTRY
  STRUCT
  ()
  (DAY STR (LT 4))
  (MONTH STR (1))
  (YEAR STR ())
  (HRMIN STR (GT 500 AND LT 600))
  (HOST STR (31))
  (STATUS STR ())
  (SOC STR ())
  (SCHED STR ())
  (RESTIME STR ())))
"Y74Q1"
```

The datalanguage code for retrieval by content in this example was:

```
" FOR L1.ST1,Y74Q1.LOGTRY
   WITH ( DAY EQ '01' OR DAY EQ '02' OR DAY EQ '03') AND
        ( MONTH EQ '01') AND
        ( HRMIN GT '0500' AND HRMIN LT '0600') AND
        ( HOST EQ '031')
        S1=DAY; S2=MONTH; S3=YEAR; S4=HRMIN;
        S5=HOST; S6=STATUS; S7=SOC; S8=SCHED; S9=RESTIME;
 END; "
```

for a port of description:

```
"CREATE L1 TEMP PORT LIST
   ST1 STRUCT
        S1 STR (2)
        S2 STR (2)
        S3 STR (2)
        S4 STR (4)
        S5 STR (3)
        S6 STR (1)
        S7 STR (3)
        S8 STR (1)
        S9 STR (3)
   END ;"
```

Finally the session is ended by:

```
<DIS>$
"Connection to the Datacomputer has been severed."
```

IV. CONCLUDING REMARKS

MDC has been fully implemented, but it has yet to be documented for the general ARPANET community. I believe that when it is documented and put to use, it will prove to be as useful a programming tool as SURRET was found to be, but a far more powerful one because of its generality.

Continual improvement of the facility must go hand in hand with the development of the Datacomputer. Version 1/0 of the Datacomputer is about to be released, and it includes many features that the former version did not possess. The differences between the two versions however indicate that a major reprogramming of MDC will not be necessary; changes will need to be made, but to augment the facility as opposed to completely changing it.

MDC currently provides both Muddle functions and the retrieval component of the Datacomputer. A major improvement to the system would be the addition of a statistical package. Rather than have this package programmed in Muddle, use should be made of other sites on the network (such as the Multics Consistent System) which already possess quite powerful statistical packages. Taken together such a system would serve as an exemplary model of network resource sharing, while at the same time provide an opportunity to explore some areas of network parallel processing.

REFERENCES

[1] D. Eastlake, et al. ITS 1.5 Reference Manual. Memo Number 161A, Artificial Intelligence Laboratory, MIT. July 1969.

[2] Elliot Smith. The Datacomputer, Version 0/9---A User Manual. Computer Corporation of America, Cambridge, Mass. August 1973.

[3] Richard Winter. Specifications for Datalanguage: Version 0/9. Computer Corporation of America, Cambridge, Mass.

[4] Greg Pfister. A MUDDLE Primer. Document SYS.11.01, Programming Technology Division, Project MAC, MIT. December 1972.

[5] Specifications for the Interconnection of a HOST and an IMP. Report number 1822, Bolt Beranek and Newman Inc., Cambridge, Mass.

[6] F. E. Heart, et al. The Interface Message Processor for the ARPA Computer Network. AFIPS Conf. Proc., volume 36, page 551. May 1970.

[7] C. S. Carr, et al. HOST-HOST Communication Protocol in the ARPA Network. AFIPS Conf. Proc., volume 36, page 589. May 1970.

[8] T. O'Sullivan, et al. TELNET Protocol. ARPA Network Information Center Document 6768. May 1971.

[9] A. McKenzie. File Transfer Protocol. NIC Document 14333.

[10] R. Bressler, et al. Remote Job Entry Protocol. NIC Document 12112. June 1971.

[11] J. Postel. Official Initial Connection Protocol. NIC Document 7107. June 1971.

[12] Safwan Bengelloun. MUDDLE Survey Data Retrieval Programs. Document SR.10.06, Programming Technology Division, Project MAC, MIT. January 1974.

APPENDIX 1: FILE MODELS

The file model used by MDC-Programmer is held in a Muddle list structure (do not confuse with a Datacomputer LIST). Each list has its first object as the name of a container and the second object as the type of the container. The remaining objects in a list are then dependent on the type of container that the list represents:

(a) For LISTs the third object is always an empty list. LISTs will also always have a fourth element which will be a description list.

(b) For STRUCTs, the third element is either #FALSE() or specifies the length of a LIST that may immediately enclose the STRUCT (this is done only for embedded LISTs). The remaining objects of a STRUCT will be one or more description lists.

(c) For STRs, the third object is always an empty list. The fourth object specifies the default value of the field or is a #FALSE(). The fifth object specifies the fixed length of the field. The sixth object gives the padding character, if any, and the seventh gives the direction of the padding (left or right). The eighth object states whether the STR is an inversion key or not; if it is and the STR holds numerical values, the ninth and tenth objects will hold the maximum and the minimum value respectively. The eleventh (optional) object performs the same function as STRUCT's third object, but for the case where lists enclose only a single STR.

Following is the model built for the list PEOPLE that was used in the SCENARIO section.

```
(PEOPLE LIST ()
 (PERSON STRUCT #FALSE()
  (NAME STR () #FALSE() 15 !"  T T #FALSE() #FALSE())
  (ADDRESS STR () #FALSE() 20 !"  T #FALSE() #FALSE() #FALSE())
  (CITY STR () #FALSE() 10 !"  T #FALSE() #FALSE() #FALSE())
  (STATE STR () #FALSE() 2 #FALSE() #FALSE() #FALSE() #FALSE() #FALSE())
  (ZIP STR () #FALSE() 5 #FALSE() #FALSE() #FALSE() #FALSE() #FALSE())
  (DEPENDENTS LIST ()
   (DEPENDENT STRUCT 2
    (NAME STR () #FALSE() 15 !"  T #FALSE() #FALSE() #FALSE())
    (AGE STR () #FALSE() 2 !"0 T #FALSE() #FALSE() #FALSE()))))))
```

APPENDIX 2: PROGRAM ABSTRACTS

The programs comprising MDC reside in the DMS Muddle Library. Each program in the Library has an Abstract, also in the Library, that gives necessary and sufficient information about the program to allow direct use of the program by other programs and to allow maintenance of the Library. This appendix contains the Abstracts for the MDC package itself and for its "ports" -- those programs designed to be called from outside the package, for example from the user's console. For the sake of brevity, Abstracts for the internal programs are not included here.

The various parts of an Abstract are named by Muddle comments (preceded by a semicolon), which briefly describe the following part. Some parts are in turn made up of parts, in hierarchical fashion. The hierarchy is defined by the various brackets used to enclose Muddle objects in the Abstract, and it is further indicated here by indentation.

Each Abstract is a Muddle vector (enclosed in [ ]) whose elements and subelements are vectors, strings (enclosed in "), lists (enclosed in ()), type declarations (enclosed in () and preceded by #DECL), forms (enclosed in <>), and atoms (everything else).

```
[
; "Unique-name"  "MDC!-PACKAGE"
; "Name"  "MDC"
; "Author"  ["SAB" "JIL"]
; "Object-type"  "PACKAGE"
; "Contents" [
  ; "Ports"  [ "CONTEXT" "NEWFILE" "PTOP" "VERBOSE" "TERSE" "SOAK"
        "UNSOAK" "CVAL" "CREQUEST" "R" "L" "DR" "DL" "UL" "UR"
        "REQUEST" "TIME-CONSTANT" "CON" "DIS" "EXECUTE" "INIT"]
  ; "Internal-functions"  ["SIMPLE" "SIMPLIFY" "CDISP" "COMRET"
        "CRESTR" "CVALUE" "CREQ" "PATHNAME" "INITVAL" "REQLIST"
        "SETVAL" "PAD" "EXPAND" "RESOLVE" "ZEROLSP" "GETLL" "GETS"
        "GETST" "NEXTS" "STRINGER" "CRELIST" "CRESTRUC" "GETLEN"
        "MAFISH" "FILIST" "FILIST1" "FILIST2" "COMSTR" "LISTN"
        "COMSTRUC" "COMLIST" "RTIME" "NETINT" "ICP" "CONNECT"
        "GETCODE" "DCERR" "PCON" "EXEC"]
  ; "Data-ports" [
    ; "Data-ports-global"  []
    ; "Data-ports-local"  [#DECL ((TIME.CONSTANT) FIX)
        "TIME.CONSTANT determines how long the ICP should wait for a
        response from the Datacomputer."] ]
  ; "Internal-data" [] ]
; "Category"  ["NETWORK"]
; "Descriptor"  ["NETWORK" "DATALANGUAGE" "DATACOMPUTER" "RETRIEVAL"]
; "External-interactions" [
  ; "Side-effect"  ["I/O" "IDENTIFIER" "DATA" "INTERRUPTS"]
  ; "Variables" [
    ; "Global" [
      ; "Setg'd"  [#DECL ((NIN NOUT) <OR CHANNEL FALSE>)]
      ; "Used"  [#DECL ((NIN NOUT) <OR CHANNEL FALSE> (OUTCHAN) CHANNEL)]]
    ; "Local" [
      ; "Set"  [#DECL (
        (DFD DPTR DSTLST FD NUMLIST PTR SIMPLENAME SUBLIST? SUBVAL? STLST) LIST
        (HIDE SOAK LC N SC STC SYNCF TIME.CONSTANT) FIX
        (DLCODE MESS PORTDESC PORTN S1 TEM1) STRING)]
      ; "Used"  [#DECL (
        (DFD DPTR DSTLST FD NUMLIST PTR SIMPLENAME STLST) LIST
        (ARGS) <SPECIAL ATOM>
        (HIDE LC N SC SOAK STC SYNCF TIME.CONSTANT) FIX
        (DLCODE MESS PORTDESC PORTN S1 TEM1) STRING (AC1 DONE) ACTIVATION
        (CHICP) <SPECIAL <OR CHANNEL FALSE>>
        (INCHAN) <OR CHANNEL FALSE>)]
      ; "Special"  [] ] ]
  ; "Functions called"  [PPRINT RTIME]
  ; "Environment"  [
    ; "Required"  []
    ; "During"  []
    ; "After"  [] ] ]
; "Location"  "LIBRARY"
; "Reference"  ["B.S.E.E. thesis, S. A. Bengelloun, June 1974"]
; "Description"  ["This is a general retrieval program for the
   Datacomputer.  The user manipulates a user-built file model to form
   a request in Datalanguage.  REQUEST then sends the Datalanguage
   program to the Datacomputer.  The information it retrieves can be
   printed on the user's console or written into a file."]
; "Argument"  []
; "Example"  ["See scenario in thesis."]
; "Notes"  []
]
```

```
[
; "Unique-name"  "CONTEXT!-MDC"
; "Name"  "CONTEXT"
; "Author"  "SAB"
; "Object-type"  "FUNCTION"
; "Contents"  []
; "Category"  ["UTILITY"]
; "Descriptor"  ["CONTEXT" "MODEL" "POINTER" "OUTPUT"]
; "External-interactions" [
  ; "Side-effect"  []
  ; "Variables" [
    ; "Global" []
    ; "Local" [
      ; "Set"  []
      ; "Used"  [#DECL ((DPTR) LIST)]
      ; "Special"  [] ] ]
  ; "Functions called"  [PATHNAME PPRINT]
  ; "Environment" [] ]
; "Location"  "MDC"
; "Reference"  ["MDC"]
; "Description"  ["
  CONTEXT displays that portion of the file model which is currently
  to the right of the pointer."]
; "Argument" [
  ; "Template"  [#DECL ("VALUE" STRING) "returns pathname of pointer"]
  ; "Argument-type"  []
  ; "Result-type"  ["STRING"] ]
; "Example"  [<CONTEXT>]
; "Notes"  []
]
```

```
[
; "Unique-name"  "NEWFILE!-MDC"
; "Name"  "NEWFILE"
; "Author"  "SAB"
; "Object-type"  "FUNCTION"
; "Contents"  []
; "Category"  ["DATA-HANDLING"]
; "Descriptor"  ["NEW" "FILE" "MODEL" "BINDING" "CREATION" "FUNCTION"]
; "External-interactions" [
  ; "Side-effect"  ["I/O" "IDENTIFIER" "DATA"]
  ; "Variables" [
    ; "Global" [
      ; "Setg'd"  []
      ; "Used"  [#DECL ((NOUT) <OR CHANNEL FALSE>)] ]
    ; "Local" [
      ; "Set"  [#DECL (
        (DFD DPTR DSTLST FD PTR SIMPLENAME STLST) LIST)]
      ; "Used"  [#DECL (
        (DFD DPTR FD PTR SIMPLENAME) LIST (ARGS) <SPECIAL ATOM>)]
      ; "Special"  [#DECL ((ARGS) ATOM)] ] ]
  ; "Functions called"  [COMRET SIMPLIFY CDISP]
  ; "Environment" [] ]
; "Location"  "MDC"
; "Reference"  ["MDC"]
; "Description"  ["
  Newfile brings in a new file model.  It also creates a series of
  functions which permit the user to manipulate the file model.
  Specifically, each function provides a mechanism for changing the
  value of the list which is associated with its name. "]
; "Argument" [
  ; "Template"  [#DECL ("VALUE" STRING STRING) "arg is local file name"]
  ; "Argument-type"  ["STRING"]
  ; "Result-type" "STRING"]
; "Example"  [
  <NEWFILE "SAB;SURVEY FILE">
  "The argument must be the name of a model file. "
  <MONTH (JUN)>
  "MONTH is a created function that modifies the field MONTH in SURVEY
  FILE. It puts JUN into the list associated with the field MONTH." ]
; "Notes"  []
]
```

```
[
; "Unique-name"  "PTOP!-MDC"
; "Name"  "PTOP"
; "Author"  "SAB"
; "Object-type"  "FUNCTION"
; "Contents"  []
; "Category"  ["PROGRAM-CONTROL"]
; "Descriptor"  ["TOP" "MODEL" "GOTO" "POINTER"]
; "External-interactions" [
  ; "Side-effect"  []
  ; "Variables" [
    ; "Global" []
    ; "Local" [
      ; "Set"  [#DECL ((DPTR DSTLST PTR STLST) LIST)]
      ; "Used"  [#DECL ((DFD FD) LIST)]
      ; "Special"  [] ] ]
  ; "Functions called"  [COMRET]
  ; "Environment" [] ]
; "Location"  "MDC"
; "Reference"  ["MDC"]
; "Description"  ["
  This function moves the pointer to the top of the file model."]
; "Argument" [
  ; "Template"  [#DECL ("VALUE" STRING) "returns pathname of pointer"]
  ; "Argument-type"  []
  ; "Result-type"  ["STRING"] ]
; "Example"  [<PTOP>]
; "Notes"  []
]
```

```
[
; "Unique-name"  "VERBOSE!-MDC"
; "Name"  "VERBOSE"
; "Author"  "SAB"
; "Object-type"  "FUNCTION"
; "Contents"  []
; "Category"  ["DISPLAY"]
; "Descriptor"  ["CONTEXT" "POINTER" "CHANGE" "OUTPUT"]
; "External-interactions" [
  ; "Side-effect"  []
  ; "Variables" [
    ; "Global" []
    ; "Local" [
      ; "Set"  [#DECL ((HIDE) FIX)]
      ; "Used"  []
      ; "Special"  [] ] ]
  ; "Functions called"  []
  ; "Environment" [] ]
; "Location"  "MDC"
; "Reference"  ["MDC"]
; "Description" ["
  This function causes the program to enter verbose mode.
  In this mode the CONTEXT is printed whenever a function returns the
  pointer pathname."]
; "Argument" [
  ; "Template"  [#DECL ("VALUE" STRING) "returns the string 'DONE'"]
  ; "Argument-type"  []
  ; "Result-type"  ["STRING"] ]
; "Example"  [<VERBOSE>]
; "Notes"  []
]
```

```
[
; "Unique-name"  "TERSE!-MDC"
; "Name"  "TERSE"
; "Author"  "SAB"
; "Object-type"  "FUNCTION"
; "Contents"  []
; "Category"  ["DISPLAY"]
; "Descriptor"  ["CONTEXT" "POINTER" "CHANGE" "OUTPUT"]
; "External-interactions" [
  ; "Side-effect"  []
  ; "Variables" [
    ; "Global" []
    ; "Local" [
      ; "Set"  [#DECL ((HIDE) FIX)]
      ; "Used"  []
      ; "Special"  [] ] ]
  ; "Functions called"  []
  ; "Environment" [] ]
; "Location"  "MDC"
; "Reference"  ["MDC"]
; "Description"  ["
  Terse is the opposite of verbose.  In terse mode only the pathname
  of the pointer is printed."]
; "Argument" [
  ; "Template"  [#DECL ("VALUE" STRING) "returns the string 'DONE'"]
  ; "Argument-type"  []
  ; "Result-type"  ["STRING"] ]
; "Example"  [<TERSE>]
; "Notes"  []
]
```

```
[
; "Unique-name"  "SOAK!-MDC"
; "Name"  "SOAK"
; "Author"  "SAB"
; "Object-type"  "FUNCTION"
; "Contents"  []
; "Category"  ["DISPLAY"]
; "Descriptor"  ["OUTPUT" "DATACOMPUTER" "RESPONSE" "CONTROL"]
; "External-interactions" [
  ; "Side-effect"  []
  ; "Variables" [
    ; "Global" []
    ; "Local" [
      ; "Set"  [#DECL ((SOAK) FIX)]
      ; "Used"  []
      ; "Special"  [] ] ]
  ; "Functions called"  []
  ; "Environment" [] ]
; "Location"  "MDC"
; "Reference"  ["MDC"]
; "Description"  ["
  Invoking SOAK inhibits the printing, on the console, of subsequent
  Datacomputer control information sent across the Network
  Connections."]
; "Argument" [
  ; "Template"  [#DECL ("VALUE" STRING) "returns the string 'DONE'"]
  ; "Argument-type"  []
  ; "Result-type"  ["STRING"] ]
; "Example"  [<SOAK>]
; "Notes"  []
]
```

```
[
; "Unique-name"  "UNSOAK!-MDC"
; "Name"  "UNSOAK"
; "Author"  "SAB"
; "Object-type"  "FUNCTION"
; "Contents"  []
; "Category"  ["DISPLAY"]
; "Descriptor"  ["OUTPUT" "DATACOMPUTER" "RESPONSE" "CONTROL"]
; "External-interactions" [
  ; "Side-effect"  []
  ; "Variables" [
    ; "Global" []
    ; "Local" [
      ; "Set"  [#DECL ((SOAK) FIX)]
      ; "Used"  []
      ; "Special"  [] ] ]
  ; "Functions called"  []
  ; "Environment" [] ]
; "Location"  "MDC"
; "Reference"  ["MDC"]
; "Description"  ["
  UNSOAK negates SOAK.  All the control information sent thereafter by
  the Datacomputer is printed on the console."]
; "Argument" [
  ; "Template"  [#DECL ("VALUE" STRING) "returns the string 'DONE'"]
  ; "Argument-type"  []
  ; "Result-type"  ["STRING"] ]
; "Example"  [<UNSOAK>]
; "Notes"  []
]
```

```
[
; "Unique-name"  "CVAL!-MDC"
; "Name"  "CVAL"
; "Author"  "SAB"
; "Object-type"  "FUNCTION"
; "Contents"  []
; "Category"  ["DATA-HANDLING"]
; "Descriptor"  ["CLEAR" "INITIALIZATION" "FIELD" "VALUE" "MODEL"
        "CRITERION" "RESTRICTION"]
; "External-interactions" [
  ; "Side-effect"  []
  ; "Variables" [
    ; "Global" []
    ; "Local" [
      ; "Set"  []
      ; "Used"  [#DECL ((DPTR) LIST (PTR) LIST)]
      ; "Special"  [] ] ]
  ; "Functions called"  [COMRET CVALUE]
  ; "Environment" [] ]
; "Location"  "MDC"
; "Reference"  ["MDC"]
; "Description"  ["
  CVAL clears all the field values from the file model that were set
  by the user with field-named functions."]
; "Argument" [
  ; "Template"  [#DECL ("VALUE" STRING "OPTIONAL" <OR FALSE LIST> LIST)
        "returns pathname of pointer"]
  ; "Argument-type"  ["FALSE" "LIST"]
  ; "Result-type"  ["STRING"] ]
; "Example"  [<CVAL>  ]
; "Notes"  []
]
```

```
[
; "Unique-name"  "CREQUEST!-MDC"
; "Name"  "CREQUEST"
; "Author"  "SAB"
; "Object-type"  "FUNCTION"
; "Contents"  []
; "Category"  ["DATA-HANDLING"]
; "Descriptor"  ["CLEAR" "INITIALIZATION" "REQUEST" "RESULT"]
; "External-interactions" [
  ; "Side-effect"  []
  ; "Variables" [
    ; "Global" []
    ; "Local" [
      ; "Set"  []
      ; "Used"  [#DECL ((DPTR) LIST (PTR) LIST)]
      ; "Special"  [] ] ]
  ; "Functions called"  [COMRET CREQ]
  ; "Environment" [] ]
; "Location"  "MDC"
; "Reference"  ["MDC"]
; "Description"  ["CREQUEST clears all user-set REQUEST fields."]
; "Argument" [
  ; "Template"  [#DECL ("VALUE" STRING "OPTIONAL" <OR FALSE LIST> LIST)
        "returns pathname of the pointer"]
  ; "Argument-type"  ["FALSE" "LIST"]
  ; "Result-type"  ["STRING"] ]
; "Example"  [
  <CREQUEST>
  "The optional arguments are used by internal functions."]
; "Notes"  []
]
```

```
[
; "Unique-name"  "R!-MDC"
; "Name"  "R"
; "Author"  "SAB"
; "Object-type"  "FUNCTION"
; "Contents"  []
; "Category"  ["DATA-HANDLING"]
; "Descriptor"  ["MOVE" "POINTER" "RIGHT" "MODEL" "CONTEXT"]
; "External-interactions" [
  ; "Side-effect"  ["DATA"]
  ; "Variables" [
    ; "Global" []
    ; "Local" [
      ; "Set"  [#DECL ((DPTR PTR) LIST)]
      ; "Used"  [#DECL ((DPTR DSTLST STLST) LIST (HIDE) FIX)]
      ; "Special"  [] ] ]
  ; "Functions called"  [PATHNAME PPRINT]
  ; "Environment" [] ]
; "Location"  "MDC"
; "Reference"  ["MDC"]
; "Description"  ["
  R moves the pointer to the right in the file model."]
; "Argument" [
  ; "Template"  [#DECL ("VALUE" <OR FALSE STRING> "OPTIONAL" FIX)
        "returns pathname of pointer"]
  ; "Argument-type"  ["FIX"]
  ; "Result-type"  ["FALSE" "STRING"] ]
; "Example"  [<R>       "moves the pointer to the right 1 place"
             <R .FIX>  "moves the pointer to the right .FIX places"]
; "Notes"  []
]
```

```
[
; "Unique-name"  "L!-MDC"
; "Name"  "L"
; "Author"  "SAB"
; "Object-type"  "FUNCTION"
; "Contents"  []
; "Category"  ["DATA-HANDLING"]
; "Descriptor"  ["MOVE" "POINTER" "LEFT" "MODEL" "CONTEXT"]
; "External-interactions" [
  ; "Side-effect"  ["DATA"]
  ; "Variables" [
    ; "Global" []
    ; "Local" [
      ; "Set"  [#DECL ((DPTR PTR) LIST)]
      ; "Used"  [#DECL ((DPTR DSTLST STLST) LIST (HIDE) FIX)]
      ; "Special"  [] ] ]
  ; "Functions called"  [PATHNAME PPRINT]
  ; "Environment" [] ]
; "Location"  "MDC"
; "Reference"  ["MDC"]
; "Description"  ["
  L moves the pointer to the left in the file model."]
; "Argument" [
  ; "Template"  [#DECL ("VALUE" <OR FALSE STRING> "OPTIONAL" FIX)
        "returns pathname of the pointer"]
  ; "Argument-type"  ["FIX"]
  ; "Result-type"  ["FALSE" "STRING"] ]
; "Example"  [<L>     "moves pointer to the left 1 place"
             <L .FIX>  "moves pointer to the left .FIX places"]
; "Notes"  []
]
```

```
[
; "Unique-name"  "DR!-MDC"
; "Name"  "DR"
; "Author"  "SAB"
; "Object-type"  "FUNCTION"
; "Contents"  []
; "Category"  ["DATA-HANDLING"]
; "Descriptor"  ["MOVE" "POINTER" "DOWN" "RIGHT" "MODEL" "CONTEXT"]
; "External-interactions" [
  ; "Side-effect"  []
  ; "Variables" [
    ; "Global" []
    ; "Local" [
      ; "Set"  [#DECL ((DPTR DSTLST PTR STLST) LIST)]
      ; "Used"  [#DECL ((HIDE) FIX (DPTR DSTLST PTR STLST) LIST)]
      ; "Special"  [] ] ]
  ; "Functions called"  [PATHNAME PPRINT]
  ; "Environment" [] ]
; "Location"  "MDC"
; "Reference"  ["MDC"]
; "Description"  ["
  DR moves the pointer down and to the right in the file model."]
; "Argument" [
  ; "Template"  [#DECL ("VALUE" <OR FALSE STRING>)
        "returns pathname of the pointer"]
  ; "Argument-type"  []
  ; "Result-type"  ["FALSE" "STRING"] ]
; "Example"  [<DR>    "moves the pointer down one level to the right"]
; "Notes"  []
]
```

```
[
; "Unique-name"  "DL!-MDC"
; "Name"  "DL"
; "Author"  "SAB"
; "Object-type"  "FUNCTION"
; "Contents"  []
; "Category"  ["DATA-HANDLING"]
; "Descriptor"  ["MOVE" "POINTER" "DOWN" "LEFT" "MODEL" "CONTEXT"]
; "External-interactions" [
  ; "Side-effect"  []
  ; "Variables" [
    ; "Global" []
    ; "Local" [
      ; "Set"  []
      ; "Used"  [#DECL ((DPTR STLST) LIST (HIDE) FIX)]
      ; "Special"  [] ] ]
  ; "Functions called"  [PATHNAME PPRINT R DR L]
  ; "Environment" [] ]
; "Location"  "MDC"
; "Reference"  ["MDC"]
; "Description"  ["
  DL moves the pointer down and to the left in the file model."]
; "Argument" [
  ; "Template"  [#DECL ("VALUE" <OR FALSE STRING>)
        "returns pathname of the pointer"]
  ; "Argument-type"  []
  ; "Result-type"  ["FALSE" "STRING"] ]
; "Example"  [<DL>     "moves the pointer down one level to the left"]
; "Notes"  []
]
```

```
[
; "Unique-name"  "UL!-MDC"
; "Name"  "UL"
; "Author"  "SAB"
; "Object-type"  "FUNCTION"
; "Contents"  []
; "Category"  ["DATA-HANDLING"]
; "Descriptor"  ["MOVE" "POINTER" "UP" "LEFT" "MODEL" "CONTEXT"]
; "External-interactions" [
  ; "Side-effect"  []
  ; "Variables" [
    ; "Global" []
    ; "Local" [
      ; "Set"  [#DECL ((DPTR DSTLST PTR STLST) LIST)]
      ; "Used"  [#DECL ((DPTR DSTLST STLST) LIST (HIDE) FIX)]
      ; "Special"  [] ] ]
  ; "Functions called"  [PATHNAME PPRINT]
  ; "Environment" [] ]
; "Location"  "MDC"
; "Reference"  ["MDC"]
; "Description"  ["
  UL moves the pointer up and to the left in the file model."]
; "Argument" [
  ; "Template"  [#DECL ("VALUE" <OR FALSE STRING>)
        "returns pathname of the pointer"]
  ; "Argument-type"  []
  ; "Result-type"  ["FALSE" "STRING"] ]
; "Example"  [<UL>      "moves pointer up one level to the left"]
; "Notes"  []
]
```

```
[
; "Unique-name"  "UR!-MDC"
; "Name"  "UR"
; "Author"  "SAB"
; "Object-type"  "FUNCTION"
; "Contents"  []
; "Category"  ["DATA-HANDLING"]
; "Descriptor"  ["MOVE" "POINTER" "UP" "RIGHT" "MODEL" "CONTEXT"]
; "External-interactions" [
  ; "Side-effect"  ["DATA"]
  ; "Variables" [
    ; "Global" []
    ; "Local" [
      ; "Set"  [#DECL ((DPTR DSTLST PTR STLST) LIST)]
      ; "Used"  [#DECL ((DPTR DSTLST STLST) LIST (HIDE) FIX)]
      ; "Special"  [] ] ]
  ; "Functions called"  [PATHNAME PPRINT]
  ; "Environment" [] ]
; "Location"  "MDC"
; "Reference"  ["MDC"]
; "Description"  ["
  UR moves the pointer up and to the right in the file model."]
; "Argument" [
  ; "Template"  [#DECL ("VALUE" <OR FALSE STRING>)
        "returns pathname of the pointer"]
  ; "Argument-type"  []
  ; "Result-type"  ["FALSE" "STRING"] ]
; "Example"  [<UR>      "moves the pointer up one level to the right"]
; "Notes"  []
]
```

```
[
; "Unique-name"  "REQUEST!-MDC"
; "Name"  "REQUEST"
; "Author"  "SAB"
; "Object-type"  "FUNCTION"
; "Contents" []
; "Category"  ["PROGRAM-CONTROL" "NETWORK"]
; "Descriptor"  ["DATACOMPUTER" "FILE" "FIELD" "OUTPUT" "RESULT"]
; "External-interactions" [
  ; "Side-effect"  []
  ; "Variables" [
    ; "Global" []
    ; "Local" [
      ; "Set"  []
      ; "Used"  [#DECL ((DPTR DSTLST PTR STLST) LIST)]
      ; "Special"  [#DECL ((STK) LIST (N M) FIX)] ] ]
  ; "Functions called"  [COMRET REQLIST]
  ; "Environment" [] ]
; "Location"  "MDC"
; "Reference"  ["MDC"]
; "Description"  [
  "REQUEST sets the request field for information retrieval.
  This field determines what field(s) of the Datacomputer file should
  be output."]
; "Argument" [
  ; "Template"  [#DECL ("VALUE" STRING LIST)
  "Argument 1 is a list of field names.  Returns pointer pathname."]
  ; "Argument-type"  ["LIST"]
  ; "Result-type"  ["STRING"] ]
; "Example"  [<REQUEST (MONTH DAY HOST STATUS)>
  "This example is taken from the SURVEY FILE.  It will cause the
  Datalanguage program to ask for the date, host, and status fields."]
; "Notes"  []
]
```

```
[
; "Unique-name"  "TIME-CONSTANT!-MDC"
; "Name"  "TIME-CONSTANT"
; "Author"  "SAB"
; "Object-type"  "FUNCTION"
; "Contents"  []
; "Category"  ["PROGRAM-CONTROL"]
; "Descriptor"  ["TIMEOUT" "LIMIT" "MODIFICATION" "ICP"]
; "External-interactions" [
  ; "Side-effect"  []
  ; "Variables" [
    ; "Global" []
    ; "Local" [
      ; "Set"  [#DECL ((TIME.CONSTANT) FIX)]
      ; "Used"  []
      ; "Special"  [] ] ]
  ; "Functions called"  []
  ; "Environment" [] ]
; "Location"  "MDC"
; "Reference"  ["MDC"]
; "Description"  ["
  TIME-CONSTANT is used to control the initial connection to the
  Datacomputer.  It determines how long the ICP should wait for
  acknowledgement from a foreign host.  If this function is not
  called, the waiting period will be 20 seconds."]
; "Argument" [
  ; "Template"  [#DECL ("VALUE" FIX <UNSPECIAL FIX>)
        "Argument is number of seconds.  Returns number of seconds."]
  ; "Argument-type"  ["FIX"]
  ; "Result-type"  ["FIX"] ]
; "Example"  [<TIME-CONSTANT 60> "It will wait for 60 seconds."]
; "Notes"  []
]
```

```
[
; "Unique-name"  "CON!-MDC"
; "Name"  "CON"
; "Author"  "SAB"
; "Object-type"  "FUNCTION"
; "Contents"  []
; "Category"  ["I/O"]
; "Descriptor"  ["CONNECTION" "ESTABLISH" "DATACOMPUTER" "ICP"]
; "External-interactions" [
  ; "Side-effect"  ["I/O"]
  ; "Variables" [
    ; "Global" [
      ; "Setg'd"  []
      ; "Used"  [#DECL ((NOUT) <OR CHANNEL FALSE>)] ]
    ; "Local" [
      ; "Set"  [#DECL ((SYNCF) FIX)]
      ; "Used"  [#DECL ((FD) LIST)]
      ; "Special"  [#DECL ((N) <SPECIAL STRING>)] ] ]
  ; "Functions called"  [DIS PCON CONNECT]
  ; "Environment" [] ]
; "Location"  "MDC"
; "Reference"  ["MDC"]
; "Description"  ["
  CON attempts to establish Network channels to/from the Datacomputer."]
; "Argument" [
  ; "Template"  [#DECL ("VALUE" <OR FALSE STRING>)
        "returns the string 'Connection to datacomputer completed.'"]
  ; "Argument-type"  []
  ; "Result-type"  ["FALSE" "STRING"] ]
; "Example"  [<CON> "Invokes connection rites."]
; "Notes"  ["
  Normally CON will not be needed.  <INIT> initializes the program and
  establishes the Network channels."]
]
```

```
[
; "Unique-name"  "DIS!-MDC"
; "Name"  "DIS"
; "Author"  "SAB"
; "Object-type"  "FUNCTION"
; "Contents"  []
; "Category"  ["I/O"]
; "Descriptor"  ["CONNECTION" "TERMINATE" "DISCONNECT" "DATACOMPUTER"]
; "External-interactions" [
  ; "Side-effect"  ["I/O"]
  ; "Variables" [
    ; "Global" [
      ; "Setg'd"  []
      ; "Used"  [#DECL ((NIN NOUT) <OR CHANNEL FALSE>)] ]
    ; "Local" [] ]
  ; "Functions called"  []
  ; "Environment" [] ]
; "Location"  "MDC"
; "Reference"  ["MDC"]
; "Description"  ["
  DIS closes the Network channels to/from the Datacomputer."]
; "Argument" [
  ; "Template"  [#DECL ("VALUE" STRING)
    "returns the string 'Connection to datacomputer has been severed.'"]
  ; "Argument-type"  []
  ; "Result-type"  ["STRING"] ]
; "Example"  [<DIS>]
; "Notes"  []
]
```

```
[
; "Unique-name"  "EXECUTE!-MDC"
; "Name"  "EXECUTE"
; "Author"  "SAB"
; "Object-type"  "FUNCTION"
; "Contents"  []
; "Category"  ["I/O" "NETWORK"]
; "Descriptor"  ["OUTPUT" "DATALANGUAGE" "SEND" "PROGRAM"]
; "External-interactions" [
  ; "Side-effect"  ["I/O"]
  ; "Variables" [
    ; "Global" [
      ; "Setg'd"  []
      ; "Used"  [#DECL ((NOUT) <OR CHANNEL FALSE>)] ]
    ; "Local" [
      ; "Set"  []
      ; "Used"  [#DECL ((PORTN) STRING (SYNCF) FIX)]
      ; "Special"  [#DECL ((ACl) ACTIVATION)] ] ]
  ; "Functions called"  [EXEC PCON]
  ; "Environment" [] ]
; "Location"  "MDC"
; "Reference"  ["MDC"]
; "Description"  ["EXECUTE executes the REQUEST to the Datacomputer."]
; "Argument" [
  ; "Template"  [#DECL ("VALUE" <OR CHANNEL FALSE>)]
  ; "Argument-type"  []
  ; "Result-type"  ["CHANNEL" "FALSE"] ]
; "Example"  [<FILECOPY <EXECUTE> .OUTCHAN>
        "prints the information retrieved on the user's console"]
; "Notes"  []
]
```

```
[
; "Unique-name"  "INIT!-MDC"
; "Name"  "INIT"
; "Author"  "SAB"
; "Object-type"  "FUNCTION"
; "Contents"  []
; "Category"  ["I/O"]
; "Descriptor"  ["INITIALIZATION" "SETUP" "CONNECTION" "ESTABLISH"
       "ICP" "DATACOMPUTER" "LOGIN" "MODEL" "FILE"]
; "External-interactions" [
  ; "Side-effect"  ["I/O"]
  ; "Variables" [
    ; "Global" []
    ; "Local" [
      ; "Set"  [#DECL (
        (DFD DPTR DSTLST FD PTR SIMPLENAME STLST) LIST
        (HIDE SOAK TIME.CONSTANT) FIX (TEM1) STRING)]
      ; "Used"  [#DECL ((DFD FD) LIST (INCHAN) <OR CHANNEL FALSE>)]
      ; "Special"  [] ] ]
  ; "Functions called"  [CON INITVAL SIMPLIFY CDISP]
  ; "Environment" [] ]
; "Location"  "MDC"
; "Reference"  ["MDC"]
; "Description"  ["
  INIT initializes the program and sets up Network channels to/from
  the Datacomputer."]
; "Argument" [
  ; "Template"  [#DECL ("VALUE" STRING)]
  ; "Argument-type"  []
  ; "Result-type"  ["STRING"] ]
; "Example"  [
  <INIT>
  "It will ask for the name of a local file containing a Datacomputer
  file description and for a login name."
  "SAB;PEOPLE FILE"      "This is a local file."
  "MIT.DMS.SURVEY"       "This is a login name."]
; "Notes"  []
]
```